RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

KIRCHHOFF-INSTITUT FÜR PHYSIK

DISSERTATION

submitted to

the
Joint Faculties for Natural Sciences and Mathematics

of the
Ruprecht-Karls-Universität
Heidelberg, Germany

for the degree of
Doctor of Natural Sciences

presented by
M.S. Felix Schürmann
born in Kassel, Germany

Date of oral examination: 8.6.2005

Exploring Liquid Computing in a Hardware Adaptation:

Construction and Operation of a Neural Network Experiment

Referees: Prof. Dr. Karlheinz Meier
Prof. Dr. Norbert Herrmann

Erkundung von Liquid Computing in einer Hardwareadaptierung : Konstruktion und Betrieb eines neuronalen Netzwerkexperiments - Die zukünftige Steigerung von Rechenleistung basiert auf Miniaturisierung, hoher Integration und Parallelisierung. Allerdings entstehen bei der Annäherung an Nanometerstrukturen neue Herausforderungen bezüglich der Verläßlichkeit der Bauelemente, der Leistungsaufnahme und der Konnektivität. Diese Aspekte finden in den momentan vorherrschenden Mikroprozessorimplementierungen noch nicht genügend Berücksichtigung. Um so wichtiger ist es, die Erforschung alternativer Rechenarchitekturen und -strategien voranzutreiben, die große Anzahlen unzuverläßiger Bauelemente nutzen können und dennoch nur eine moderate Leistungsaufnahme haben. Diese Doktorarbeit beschreibt die Konstruktion eines Experiments, welches es ermöglicht, Adaptierungen von künstlichen neuronalen Netzwerkparadigmen in Silizium bezüglich ihrer Anwendbarkeit, effizienten Leistungsaufnahme und Fehlertoleranz zu untersuchen. Der hier vorgestellte Aufbau besteht aus peripherer Elektronik, programmierbarer Logik und Software, um einen gemischt digital-analogen CMOS Mikrochip zu betreiben, auf dem ein flexibeles Perceptron mit 256 McCulloch-Pitts Neuronen implementiert ist. Mit Hilfe dieses *neuronalen Netzwerkexperiments* wird eine kürzlich veröffentlichte Strategie untersucht, mittels derer rekurrente Netzwerktopologien nutzbar gemacht werden können. Die hier präsentierte erstmalige Adaptierung von *Liquid Computing* an ein neuronales Netwerk auf CMOS-Basis bestätigt dessen vermutete Eignung für eine Hardwareadaptierung. Dabei wird nicht nur die Machbarkeit demonstriert, sondern auch die Toleranz gegenüber Substratvariationen und die Robustheit gegenüber Fehlern, die während des Betriebs auftauchen.

Exploring Liquid Computing in a Hardware Adaptation: Construction and Operation of a Neural Network Experiment - Future increases in computing power strongly rely on miniaturization, large scale integration, and parallelization. Yet, approaching the nanometer realm poses new challenges in terms of device reliability, power dissipation, and connectivity—issues that have been of lesser concern in today's prevailing microprocessor implementations. It is therefore necessary to pursue the research on alternative computing architectures and strategies that can make use of large numbers of unreliable devices and only have a moderate power consumption. This thesis describes the construction of an experiment dedicated to exploring silicon adaptations of artificial neural network paradigms for their general applicability, power efficiency, and fault-tolerance. The presented setup comprises peripheral electronics, programmable logic, and software to accommodate a mixed-signal CMOS microchip implementing a flexible perceptron with 256 McCulloch-Pitts neurons. This *neural network experiment* is used to explore a recent strategy that allows to access the power of recurrent network topologies. While it has been conjectured that this *liquid computing* is suited for hardware implementations, this first time adaptation to a CMOS neural network affirms this claim. Not only feasibility but also tolerance to substrate variations and robustness to faults during operation are demonstrated.

Contents

Introduction **1**

1 Hardware Neural Network Concepts **5**
1.1 Foundations . . . 5
1.1.1 Historical Origins of ANNs – A Hardware's Perspective . . . 5
1.1.2 VLSI Variety . . . 7
1.1.3 Present Trends and Future . . . 9
1.2 Design Considerations . . . 10
1.2.1 General . . . 10
1.2.2 Network Model Implications . . . 10
1.2.3 VLSI Implications . . . 15
1.3 Realization . . . 18
1.3.1 A Neural Network Experiment . . . 18
1.3.2 ANN ASICs . . . 19
1.3.3 Experimental Framework . . . 20
1.3.4 Suitable Training Approaches . . . 21

2 Computing without Stable States **23**
2.1 Recurrent Neural Networks . . . 23
2.1.1 Finite Automata and Neural Networks . . . 23
2.1.2 The Liquid State Machine Approach . . . 24
2.2 Exploring Liquid Computing in Hardware . . . 27
2.2.1 Motivation . . . 27
2.2.2 *liquid*HAGEN . . . 28
2.3 Liquid Dynamics . . . 31
2.3.1 Input Driven Networks . . . 31
2.3.2 The Edge of Chaos . . . 33
2.3.3 Performance Measures . . . 35

3 The HAGEN Prototype Implementation **39**
3.1 Learning from the Predecessor ASIC . . . 39
3.2 HAGEN Overview . . . 42
3.3 Network Block . . . 47
3.3.1 A Network Cycle . . . 48
3.3.2 Elementary Synapse . . . 49
3.3.3 Neuron . . . 53
3.3.4 Weight Storage . . . 58
3.4 Digital-to-Analog Converters . . . 60

3.5 Interface . . . 61
3.5.1 Physical Layer . . . 62
3.5.2 Logical Layer . . . 68
3.6 Power Considerations . . . 70
3.6.1 Power Consumption of a Network Block . . . 70
3.6.2 The HAGEN Prototype . . . 71
3.7 Beyond the HAGEN Prototype . . . 72
3.7.1 Minor Modifications . . . 73
3.7.2 Scaling . . . 73
3.7.3 Outlook . . . 75

4 Experimental Framework **77**
4.1 Substrates . . . 77
4.1.1 ANN ASICs . . . 77
4.1.2 Dedicated Peripheral Electronics . . . 78
4.1.3 PCI-based Mixed-Signal FPGA Adapter . . . 83
4.1.4 General Purpose Computer . . . 87
4.1.5 Advancing the Framework . . . 88
4.2 Programmable Logic Design . . . 91
4.2.1 Overview . . . 91
4.2.2 HAGEN State Machine . . . 93
4.2.3 Extensions . . . 94
4.3 Control Software (HANNEE) . . . 96
4.3.1 Overview . . . 96
4.3.2 Hardware Abstraction Layer . . . 98
4.3.3 Input/Output Pattern Management and Operation Modes . . . 99
4.4 Applications . . . 102
4.4.1 Hardware-in-the-loop . . . 102
4.4.2 Precalculated Weights . . . 104
4.4.3 Liquid Computing . . . 106

5 Experimental Results **111**
5.1 General . . . 111
5.2 Exploring a Hardware Liquid . . . 111
5.2.1 Initial Experiments . . . 111
5.2.2 Observing the Edge of Chaos . . . 115
5.2.3 Scaling Behavior . . . 121
5.3 Fault-Tolerance . . . 123
5.3.1 Robustness to Substrate Variations . . . 123
5.3.2 Graceful Degradation with Time . . . 125
5.3.3 Graceful Degradation with Individual Synapse Faults . . . 131

Discussion **135**

A Liquid Computing Supplements **i**
A.1 Linear Regression Training . . . i
A.2 Mean-Field Theory for Input Driven Networks . . . ii

B HAGEN Prototype Supplements **v**
B.1 Inter-Block Routing vi
B.2 Bonding Diagram vii
B.3 Interface viii
B.3.1 Command-Address Symbols viii
B.3.2 Slow-Control Registers A and B ix
B.4 Clock Pattern x
B.4.1 Preamble-Pattern x
B.4.2 A Loopable Clock Pattern x
B.5 HAGEN Measurements xii

C Experimental Framework Supplements **xiii**
C.1 Peripheral Electronics xiv
C.2 Pattern Handling xv

D Variable Network Resources **xix**

E Supplementary Liquid Computing Experiments **xxv**

Acronyms **xxxi**

Bibliography **xxxiii**

Introduction

Unraveling the principles of biological nervous systems has shown to be a long lasting and enormously challenging endeavor. To date it seems that only a unified effort of disciplines ranging from biology to medicine and from physics to engineering will lead to a complete picture [53, 137].

While this laborious process may still need decades in order to converge, along the way many useful results and insights will be gained. It was over 60 years ago that McCulloch and Pitts [134] used a mathematical model to give an abstract description of neural activity. On the one hand, this formalism made it possible to assess the (computational) capabilities and limitations of networks of neurons and encouraged others to conceptualize the biological findings. On the other hand, the research on artificial neural networks (ANN) became a topic of its own[1]. The most prominent example for the former is the strongly biologically motivated *perceptron* by Rosenblatt [173], later fully understood in its restrictions [140] and shown to have a *universal approximation property* if used in multiple layers [97]. An important achievement of the ANN research was the rediscovery[2] of the gradient-descent *back-propagation algorithm* [178] which opened the field for technical applications.

The history of hardware adaptations of neural networks reflects these different directions: Early implementations like the *SNARC* by Minsky [138] and the *MARK I Perceptron* by Rosenblatt [80] and more recently Mead [135] with his *silicon retina* aimed at mimicking biological systems. In parallel, a multitude of hardware implementations were dedicated to merely accelerate the training and operation of ANNs. Primarily, this was in reaction to the shortcomings of available computing power for adequate software simulations in the late eighties and early nineties. The approaches ranged from digital arithmetic accelerators to stand-alone analog implementations. The prevailing technology used is the *very large scale integration* (VLSI) of *complementary metal oxide semiconductor* (CMOS) devices, also the basis for modern digital microprocessors. An acceleration of the computation by dedicated ANN circuits therefore has to arise from architectural rather than from technological advantages. Yet, the designs of modern microprocessors incorporate instruction and data level parallelism as well as highly optimized non-standard logic that uses the analog properties of the substrate [43]. Due to the flexibility of software and the ease of porting it from one processor generation to the next, allowing immediate utilization of technological advances, present research activity is dominated by software implementations of ANNs.

However, architectural advantages that can be achieved with dedicated circuits are not limited to mere acceleration. Other performance aspects are of increasing importance: An unresolved and critical issue is the power dissipation of integrated circuits. The speed of modern microprocessors is already governed and limited by their thermal power management. Reducing the supply volt-

[1] An elaborate selection of important publications can be found in the collection edited by Anderson and Rosenfeld [6].

[2] The back-propagation algorithm was independently discovered several times: 1969 by Bryson and Ho [32], 1974 by Werbos [222], and 1985 by Parker [159] and LeCun [120], when it was finally widely spread in the neural network community.

ages along with the shrinking of the feature size may increase the power efficiency of the active components, but the increased operation frequencies, larger off-currents, and higher component densities in effect worsen the power dissipation. The miniaturization of the structures, furthermore, is accompanied by the susceptibility to imperfections in the substrate and to variations in the manufacturing process which reduce the yield. Ultimately, manufacturing processes in the nanometer-realm will rely on principles such as self-assembly which is likely not only to produce unreliable devices but also to limit the interconnectivity and require some regularity. These challenges are anticipated by the semiconductor industry in their bi-annual semiconductor roadmap [104].

The future of semiconductors therefore relies on the research of alternative computing architectures that incorporate features such as sparseness of activity, reconfigurability, and fault-tolerance. From observations of biological nervous systems, one can conclude that it is indeed possible to realize these properties in a single system. The goal of contemporary research on hardware implementations of neural networks should therefore aim to answer the question of how these properties can be efficiently realized in an artificial system.

Part of the answer lies in the technical realization of the artificial system; and part in the learning strategy that actually configures the neural architecture. This thesis describes the construction and operation of a neural network experiment and is dedicated to start exploring both parts of the answer.

In the current setup, the experiment allows to assess a network paradigm based on a flexible perceptron with about 33,000 synapses and 256 McCulloch-Pitts neurons for its efficient realization in a standard CMOS technology. The appropriate concepts and implementations of the neural network model in mixed-signal VLSI have been developed in [187] and have received partial contributions by this thesis work. The ANN microchip is accommodated by a framework comprised of peripheral electronics, programmable logic, and software running on a microprocessor. This modularity is the key to do research on a wide variety of questions: those that deal with the feasibility of specific network paradigms, those that are concerned with the exploration of training strategies, and others that have a specific application in mind. The neural network experiment is a conjoint effort with several other researchers[3].

In [86] the neural network experiment has been used to extensively explore learning with evolutionary algorithms. The approach relies on iteratively testing weight configurations and—if appropriately adapted—it can profit from the parallelism inherent to the hardware and its fast reconfigurability and operation. Competitive results in commonly used pattern classification benchmark problems have been achieved. The hardware-in-the-loop operation, which tests the effect of a synaptic weight change on the actual ANN hardware, furthermore allows one to cope with variations of the substrate.

While those investigations use feed-forward topologies where the activity is propagated from input neurons to the dedicated output neurons in a pipelined fashion, biological nervous systems usually also have long or short-range *recurrent* connections. May the precise purpose be difficult to pinpoint, the overall effect is clear: Recurrent connections make the network response depend on its history, allowing perhaps regulatory cycles as well as short-term memories. Specifically, the essential difference is that feed-forward networks are static mappings from an input to an output, while recurrent networks are dynamical systems.

For the processing of time varying inputs, recurrent neural network are promising whereby configuring the weights appropriately proves to a difficult task [106]. In 2001, Maass et al. [131] and Jaeger [105] published independently a strategy that is fundamentally different to other learn-

[3]This thesis concentrates on the electronics; for a software perspective see [86]. The programmable logic will be detailed in the Ph.D. thesis of T. Schmitz, yet to be published.

ing strategies for recurrent networks to date in that it does not change the synaptic weights of the recurrent neural network at all but rather draws them once randomly from a given distribution. Instead, only an observer is trained that continuously interprets the transient behavior of the recurrent neural network. Maass et al. and Jaeger showed that in many cases a linear readout suffices to predict outputs that are non-linear functions of the input.

Intuitively, this strategy may be viewed as an input constantly perturbing a dynamical system; the observable state of this system is then interpreted by the readout. The internal interactions within the dynamical system will evoke changing trajectories for different inputs and time courses. If the interactions in the system are appropriate, non-linear projections of the input are observable in the transient dynamics which are available for linear readout. According to the terminology of Maass et al. this strategy is called *liquid computing*.

Different types of recurrent neural networks have been proposed to be configurable as adequate[4] dynamical systems [131, 105, 21], yet all of them have been implemented in software. However, there are good reasons to adapt the strategy for a hardware implementation: First of all, a physically realized dynamical systems frees one from having to compute the non-linear projections and to handle the desired high-dimensionality explicitly. Furthermore, the published simulation experiments showed that essentially random synaptic weights can accommodate appropriate dynamics. This may be used to hide device variations of the substrate in the weight distribution or it may even enrich the dynamics of the system. Lastly, the robustness with which randomly generated recurrent neural networks yield adequate dynamics gives rise to the assumption that this property is not emerging from the individual synaptic connections or neurons but rather from the distributed interaction of several of them. This promises an inherent tolerance to faults throughout operation.

In this thesis the strategy of Bertschinger et al. [21] is adopted to realize a hardware adaptation of liquid computing—to the knowledge of the author this is the first reporting. This hardware implementation allows to test the propositions of robustness to substrate variations and the tolerance to faults in a physical experiment and to assess liquid computing with recurrent neural networks as a possible paradigm for future computing architectures.

The thesis is partitioned into five chapters: The first chapter will motivate the design of the neural network experiment while the second chapter introduces liquid computing and its implementation in detail. Chapters three and four describe the used ANN microchip and the experimental framework along with the contributions to both. Finally, chapter five will present the experimental data collected on liquid computing.

[4] Other dynamical systems may similarly be considered as long as they provide certain properties; for details see Ch. 2 and [132].

Chapter 1

Hardware Neural Network Concepts

Since the early days of artificial neural networks, hardware implementations have accompanied the research. Recently, the ever increasing performance of general-purpose digital microcomputers and the ease of their programming challenge the enormous effort of dedicated hardware development. Yet, the ongoing miniaturization of computing substrates to the nanometer realm make it necessary to explore architectures that can cope with unreliable devices, interconnect limitations, and a moderate power dissipation. The inherent parallelism and robustness of neural networks in combination with hardware-friendly paradigms and adequate training strategies offer a promising approach. In this chapter a combined hardware-software approach is presented that allows to assess these propositions in a physical experiment.

1.1 Foundations

1.1.1 Historical Origins of ANNs – A Hardware's Perspective

While the *summing amplifier* by Swartzel [196] implementing an electrical threshold device preceeded the seminal article of 1943 by McCulloch and Pitts [134] which presents neurons as binary decision units, the first dedicated neurocomputer was realized by Minsky and Edmonds in 1951 [138]: The *Stochastic Neural-Analog Reinforcement Calculator* (SNARC) was an electromechanical implementation of 40 cells that—with a certain probability—transmitted incoming electrical pulses. The individual transmission probabilities were realized by multivibrator circuits that could be adjusted by potentiometers. With the help of magnetic clutches these adjustments could be initiated by the network pulses themselves which allowed an ongoing reinforcement learning. SNARC was tested on a series of maze-running experiments, where at a number of vertices a 'direction-decision' had to be performed to ultimately arrive at the exit. Possible advantages of a dedicated parallel hardware implementation for the fault-tolerance have already been observed by Minsky (p. 4-39 in [138]):

> *"Perhaps most impressive was the fact that at any time during operation, the random net of Snarcs could be rearranged, wires pulled out, tubes removed, and even fuses blown, and yet, if not too many snarcs were inactivated, the machine would, again*

after a period of confusion, adapt to the new situation and find paths to the preferred vertices." [Marvin Minsky, 1954]

While this was the first hardware implementation of a pulse-based artificial neural network, it was primarily a toy demonstrator for Minsky's theoretical work on reinforcement learning.

It was Rosenblatt in 1957 [173] who extended McCulloch-Pitts networks, which needed to be explicitly constructed for each problem, to networks with variable connection weights. The so-called *perceptron* could be trained with reinforcement learning to classify patterns and was even capable of generalizing on previously unseen patterns. In the following years Rosenblatt performed numerical simulations [174] on an IBM 704 mainframe computer [99] and in a parallel effort with co-workers he developed the *MARK I Perceptron* hardware with automatic learning [80]. Even though the actual MARK I hardware was not remarkably faster than the simulations [23], the first realization of a 'perceiving' machine and the ease of varying the stimuli, i.e., stimuli could be provided with overhead slides, attracted a lot of interest. Moreover, Rosenblatt anticipated a much higher performance gain of hardware implementations for larger network architectures where the parallelism would pay off [174]. Technically, the MARK I had a 20x20 grid of photo-receptors, each hand-wired to up to 40 of the 512 *associator units* which is shown in Fig. 1.1. These were responsible for thresholding their input signals, summing them and generating a weighted output signal via a dc-motor driven potentiometer which represented the synapses. The associator units then were connected to one of up to 8 threshold units (relays driven by dc amplifiers).

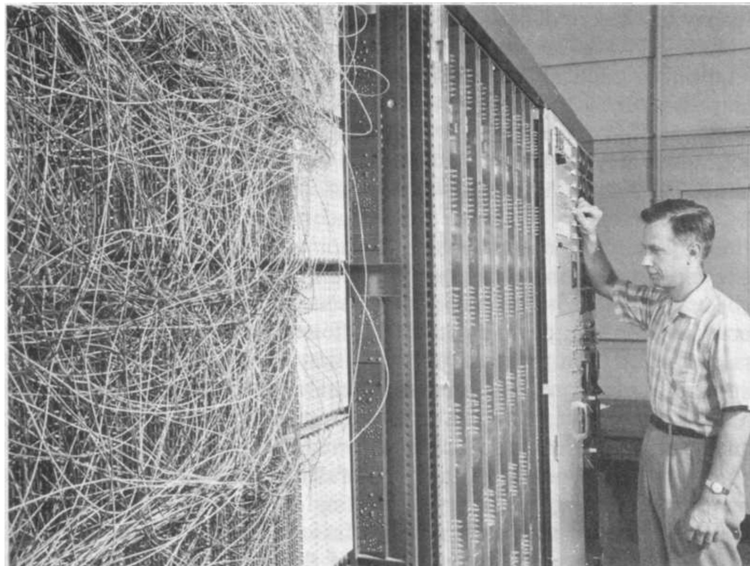


Figure 1.1: The MARK I Perceptron: in front the patch board for connecting the 400 photo-receptors with up to 40 of the 512 associator units each. Photograph taken from [82].

The success of the MARK I drew increasing interest to the field of neural network hardware, and in 1960 Widrow and Hoff published a supervised learning rule along with a hardware implementation of a threshold unit with 16 adjustable weights, the *adaptive linear* (ADALINE) [225]. The initial ADALINE element used multi-aperture magnetic cores [41, 27] for the weight storage and a human operator needed to manually adjust the gains according to a mechanized procedure (knobby ADALINE). Widrow demonstrated that it is possible to build networks from theses elements, the multiple ADALINE (MADALINE), and even commercialized improved versions with automatic electronic learning with his Memistor company [224]. The Widrow-Hoff learning

procedure—being a local, error-correction strategy—is also known as the *least-mean-squares rule* or *delta-rule* and is the basis for the renowned *back-propagation algorithm*.

With the 1969 book by Minsky and Papert [140], a rigorous mathematical description of a perceptron's capabilities was given, and especially, its limits. While it was already known by Rosenblatt and co-workers that a perceptron is not able to correctly distinguish between linearly non-separable patterns, Minsky and Papert proved that this limitation is indeed a general inability to compute a certain set of logical functions[1]. In a generalization they showed that furthermore certain types of pattern distinctions cannot be made by realistic perceptrons, i.e., perceptrons that have a limited fan-in. Their (wrong) conjecture that these inabilities are fundamental even to more complex perceptron architectures (e.g. multi-layer perceptrons), further accelerated the renunciation of perceptrons, which had already started in the mid 60s [82]. For the following 15 years, research funding was directed from neural networks to areas like artificial intelligence, and even though theoretical progress was still made, hardware implementations were increasingly neglected.

The early 80s brought new innovations and approaches to the field of neural networks, e.g. 1982 *self-organizing maps* by Kohonen [114] or 1983 the *neocognitron* by Fukushima [64], and attracted a new group of researchers: physicists. Initially, Hopfield drew the connection between his proposed network model and the Ising model of spin-glass physics in 1982 [93]. He was followed by Kirkpatrick with the *Boltzmann machine* [110] and Geman with *simulated annealing* [68]. In 1984 Hopfield expanded his model and proposed an electrical circuit for its realization [94]. He was followed by researchers proposing an electro-optical [55] and electronic implementation based on MNOS/CCD [180]. Meanwhile, the availability of general purpose computers promoted software simulations, e.g., the *MARK I & II* packages by Hecht-Nielsen, but at the same time raised the desire for faster simulation speeds. This was the impetus for the development of general purpose digital neural network accelerators such as the *MARK III & IV* 1984-1986 [82].

Finally, the renaissance of the artificial neural network research began with the 1986 book edited by Rumelhart and McClelland [179]. The authors used the accumulated knowledge about neural processing to formulate a framework for a connection-based, distributed and highly-parallel approach to neural computing. With this book the back-propagation algorithm became widely known [178] and previously intractable problems suddenly became solvable. In 1989 Hornik, Stinchcombe and White even proved that multi-layer perceptrons are universal function approximators [97].

While previous hardware implementations experimented with all different kinds of substrates, the inherent parallelism of CMOS electronics in conjunction with ever advancing tools and fabrication let VLSI become the prevailing choice for the implementation of neural processing systems. In 1989 Mead [135] showed impressively that CMOS VLSI is not restricted to digital implementations but, rather, is well suited to directly model neural functionality by physical properties of the substrate (*silicon retina* [206], *electronic cochlea* [128]). Henceforth, the race was on between digital, analog and mixed-signal implementations.

1.1.2 VLSI Variety

The late 1980s and the early 1990s brought a wave of new approaches to accelerate neural computing. On the one hand, many commercially available parallel computing platforms were evaluated for their suitability to accelerate neural networks, e.g., implementations on the Connection Machine 1990 by Singer [203], transputer based solutions by Vuurpijl 1992 [221], or a CRAY YM-P

[1] And even worse: The percentage of Boolean functions $f:\{0,1\}^n \rightarrow \{0,1\}$ that are linearly separable goes to 0 for $n \rightarrow \infty$ [228, 1].

supercomputer adaptation 1993 by Leung and Setiono [123]. On the other hand, many semiconductor companies took on the developments of research groups for dedicated neural network chips to make neural computing available on a large scale—among them Intel, Siemens, IBM, Hitachi, Philips, and others. A concise overview can be found in [126]. A selection of neural network chips and systems built from it is given in Tab. 1.1.

Many more hardware implementations were and still are developed in university laboratories and are described in several overview articles [90, 100, 126, 83, 16, 34, 192, 20]. Digital implementations of neurocomputing hardware are described in [19, 191, 237]. Purely analog approaches are reviewed in [52, 214]. As shown in [170], fuzzy systems are very similar to neural networks which led to fuzzy hardware implementation be included in an overview [171].

In order to understand the diversity of approaches in hardware neural networks it is useful to categorize them. Several authors have proposed classification schemes which either can be applied to neural hardware or were explicitely postulated for it. An example of the former is the categorization of the computing architecture according to Flynn 1972 [60] in the instruction and data level parallelism, e.g., multiple-instruction multiple-data (MIMD). Nordström and Svensson, on the other hand, proposed to divide neural hardware by the degree of parallelism [156]. Other suggested to classify it by the biological evidence of the realization [72, 192] or by the training type [214].

A simple and coarse, yet useful, scheme was described by Heemskerk [83] in 1995 which goes back to Rückert [177]. Even though it does not factor in the purpose or biological relevance of the neural device, it clarifies major differences. As a major premise, he differentiates between *neurochips* and *neurocomputers*. This recognizes that reconfigurability, training, and interfacing are an integral part of a neurocomputer but need not necessarily be contained in a neurochip. A neurocomputer then can be comprised either of *standard chips* or dedicated *neurochips*. The category built from standard chips is divided into systems comprising a *single (sequential) processor* supported by a dedicated *accelerator* or systems comprised of *multiprocessors*. The category of dedicated neurochips is divided into *analog* and *digital* solutions or a combination of both (*hybrid* or *mixed-signal*).

It is understood that there are potential performance increases when going from the category of sequential processors with accelerator to the category of dedicated mixed-signal neurochips [72, 177, 83, 19]. Tab. 1.1 shows the classification scheme according to [83] which originates from Rückert [177] and lists prominent examples.

<table>
<tr><td rowspan="5">neuro-computers</td><td rowspan="2">standard chips</td><td>seq. + accelerator</td><td>ANZA plus (Motorola 68020, 68881) [82]</td></tr>
<tr><td>multiprocessor</td><td>Mark IV (Motorola 68020) [82]</td></tr>
<tr><td rowspan="3">neuro chips</td><td>analog</td><td>Mod2 (Intel ETANN) [145, 91]</td></tr>
<tr><td>digital</td><td>Siemens SYNAPSE-1 (MA-16) [169]</td></tr>
<tr><td>mixed-signal</td><td>ANNA [25]</td></tr>
</table>

Table 1.1: Classification of neurocomputers with prominent examples following Heemskerk [83].

It can be observed that the interest in building dedicated neurocomputers declined in the mid 90s. The reasons for this were manifold and still challenge todays neural hardware development:

- First of all, in order to be efficient, neurocomputers need to be optimized for a certain neural network model. For a hardware implementation, the dedication to a model means a limited flexibility which in turn complicates or even prevents the exploration of new neural network paradigms [83]. As long as several paradigms are competing, investments to develop full-blown neurocomputers do not seem appealing.

- A second, important reason is the steady growth of computing power in conventional computers and *digital signal processors* (DSPs) which is due to an ever higher device count (semiconductor roadmap 2003 [103]). That this indeed translates into an increased performance for simulated neural networks was shown in [8]. Architectural advancements in microprocessor design such as the integration of vector units [121] together with optimizing compilers let desktop computers become so powerful that they are well suited for simulating neural networks of limited size. This development is promoted by the observation that for a lot of research and applications the available software performance is already satisfactory. The expected advantage of a hardware solution by further exploiting parallelism therefore might not be needed [192] or not be relevant due to accompanying pre- and post-processing not making use of further parallelization (Amdahl's law [4]).
- Lastly, the possible performance gain of dedicated hardware solutions compared to software implementations comes along with a substantially increased design effort. For mixed-signal solutions, which are expected to have the highest performance potential, the effort is even higher because of the poorly automated design process of full-custom analog circuitry. The resulting longer time-to-market and higher financial investment at a reduced flexibility burden the development of dedicated neural hardware [171, 133].

1.1.3 Present Trends and Future

It was however commercial interest, not research on hardware neural networks itself, which abruptly ended. A number of implementations of different paradigms is present in recent surveys [214, 20], special issues [37, 168, 125], as well as in conference proceedings, and journals [69, 164, 35]. Furthermore, the aspect of closely mimicking biological neural systems in CMOS as started by Mead [135] yields an increasing number of spiking neural network implementations, e.g. [230, 65, 73, 190, 185].

The competition with general purpose microprocessors and digital signal processors most affected the research on dedicated digital neural hardware [157]. This is because digital neural networks are considered to be essentially hardware accelerators [146], and therefore they are in direct competition. Nonetheless, dedicated digital implementations are still considered for applications due to a problem-specific performance advantage. For example, a new level-1 trigger based on neural networks for the H1 experiment at the HERA particle accelerator is proposed [44]. The currently observed renewed attention for digital implementations, especially those in programmable logic devices, is mainly caused by advances in the design tools and the so-called *hardware/software co-design* (HW/SW co-design) [229]. A prominent example are embedded systems where different types of resources are present (e.g. microprocessor cores, programmable logic, and signal processors) and an optimal apriori partitioning of the design becomes difficult. The idea of the HW/SW co-design is to allow the developer to easily assess the resulting performance with the help of a fully automated design environment. If applied to the development of neural hardware, the best use could be made of the parallel resources in the programmable logic and the flexibility of soft-programmable microprocessors while reducing the design effort [171, 237, 61]. Successful implementations are already reported [172, 160].

The automation of analog and mixed-signal integrated circuit design is unlike harder and a topic of research itself [71, 124, 117]. These types of designs therefore are quite costly in terms of design time and engineering expenses. According to [115] an analog solution therefore needs to provide an increase in efficiency (i.e. performance, size, or energy consumption) of at least an order of magnitude over digital ones as to become attractive for commercial use. Immediate applications arise for the realization of extremely low power consumption and robust devices, e.g.

implantable bio-medical devices [122, 199]. Others, such as nonlinear channel equalization in telecommunications, additionally demand extremely high speed and are shown to be realizable with neural networks [147, 161, 197, 54].

However, the main reasons for contemporary research on adapting neural architectures and principles for an efficient hardware implementation arise from the expected future of VLSI substrates: While the semiconductor roadmap [103] describes[2] in detail the expected steps of CMOS miniaturization, and in this regard predicts the increase of microprocessor performance, a whole chapter is dedicated to emergent devices and architectures [104] in order to take on the challenges concerning the power dissipation and unreliability of individual devices. The miniaturization towards the nanometer-realm furthermore will raise the problem of interconnecting the small structures. The construction of these shrunk structures will likely rely on self-assembling and will hence require some kind of regularity of the structures. The economical use of resources in terms of using as few devices as possible, on the other hand, will be less important due to the realizable integration. Consequently, computing architectures have to be developed that make use of the large scale of integration by sparsely using the individual devices simultaneously. Similarly, tolerance to faults will be required: those resulting from manufacturing as well as those exhibited throughout operation. Biological neural systems prove that these paradigms can be realized, thus the challenge is to find suitable adaptations for available and future hardware substrates.

1.2 Design Considerations

1.2.1 General

The network model that is motivated and described in the following sections has been proposed in [186, 187]. It aims at an efficient implementation by a standard CMOS process and incorporates aspects of low power consumption, scalability, and flexibility.

1.2.2 Network Model Implications

If one reverts to the analog nature of the elementary components present in the CMOS substrate, one can profit from a multitude of properties inherent to the transistor. Not only functions such as non-linear amplification, thresholding, and a multitude more when used in combination with other transistors [218, 67] but also the possibility of low power operation allow a dramatic increase in efficiency compared to using transistors merely as binary switches in digital circuits. Yet, an efficient mapping between the analog properties and the network model is crucial and not trivial. A key aspect is to choose a hardware-friendly network model [142], i.e., a model that explicitly allows an efficient representation by physical quantities [218, 20]. On the lowest level, efficient implementations of threshold devices have been developed, e.g. the *Neuron-MOS* transistor [201] suited for CMOS, and neural networks proposed to be implemented with single-electron transistors [74]; an overview can be found in [20]. Furthermore, efficient mappings of more complex properties and functions such as sigmoid transfer functions, normalization, multiplication have been realized [219].

The very feature of continuous quantities in analog VLSI comes at a cost: There are no discrete states the analog signals may be restored to, which consequently makes the design more susceptible to noise and offsets [181]. Binary logic in digital systems is the strict opposite and copes with much higher noise levels. Consequently, binary signals can be stored, buffered, distributed,

[2] Indeed, the roadmap is a collaborate effort of the semiconductor industry and rather a detailed plan than a mere prognosis; it is thus often called self-fulfilling.

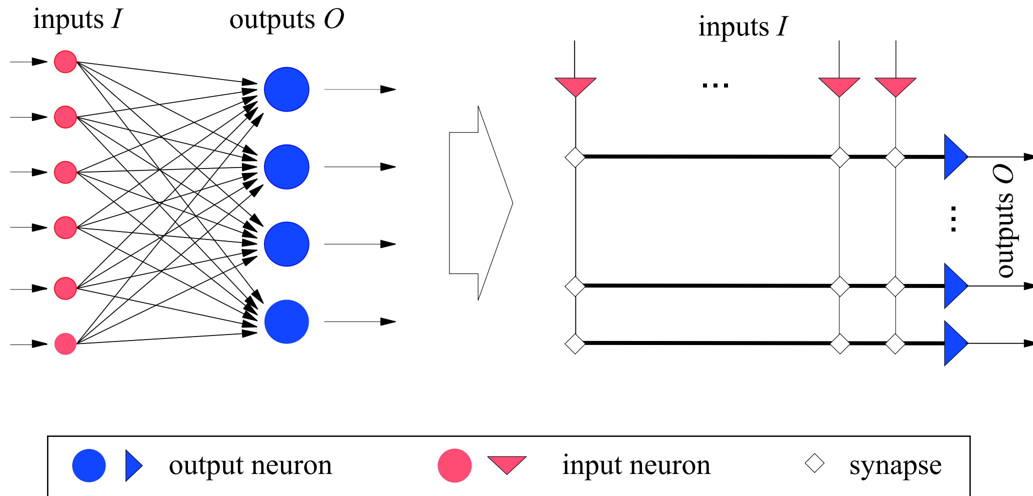


Figure 1.2: Fully-connected feed-forward perceptrons can be efficiently mapped to an array-based structure. Adapted from [184].

and transmitted much simpler than analog signals. Digital implementations therefore will be a good choice if high signal-to-noise ratios are required [219]; this will be important for long-range transportation of network activity.

One solution is to choose a low-connectivity network model, e.g. *cellular neural networks* [38, 176, 37], were connections only need to be made to nearest-neighbors. The more promising approach though is to combine analog and digital signal processing: The computationally intensive parts are to be implemented in analog VLSI, preferably in a regular, array-based structure [115]. Digital circuitry then will have to ensure a reliable and routable communication as to allow the scaling of the neural network.

A fan-in limited, analog, array processing with digital interface provides an efficient way for vector-matrix multiplication and is suited for many neural network paradigms, from feed-forward perceptrons ([25] or c.f. Fig. 1.2) to support vector machines [69]. Yet, the details of the elementary analog cells exhibit relevant differences. These will be especially important if the design is to be adapted to smaller feature size technologies. The ability to cope with imperfections and variations in gain and offset is a challenge for the analog circuits, the training, and ultimately the underlying network model. In contrast to off-chip learning, where the synaptic weights are calculated according to an idealized model of the neurons and weights, chip-in-the-loop approaches or on-chip training can deal with the actual characteristics and mismatches of the neural network substrate [142, 214].

A Network of Network Blocks The model proposed in [186, 187] utilizes McCulloch-Pitts neurons and arranges them to a set of fully-connected, single-layer perceptrons in rectangular blocks with binary inputs as shown in Fig. 1.2. In the following, this is referred to as a *network block*. The network block's response is modeled by the standard perceptron formula, where I,O are the binary inputs and outputs, ω the synaptic weights, and $\Theta(x)$ is the Heaviside step function:

$$O_i = g(\sum_j \omega_{ij} I_j), \quad g(x) = \Theta(x), \ \omega \in [-1,1] \ I,O \in \{0,1\}. \tag{1.1}$$

In addition to the straight-forward rectangular structure, the choice of McCulloch-Pitts neurons yields a natural border between the analog and digital implementation: the synapse array and the

input stage of the output neurons can be implemented efficiently in analog VLSI; the input neurons are merely digital switches just as the outputs of the output neurons are digital signals. The right side of Fig. 1.3 illustrates this partitioning with dark gray (analog) and light gray shading (digital).

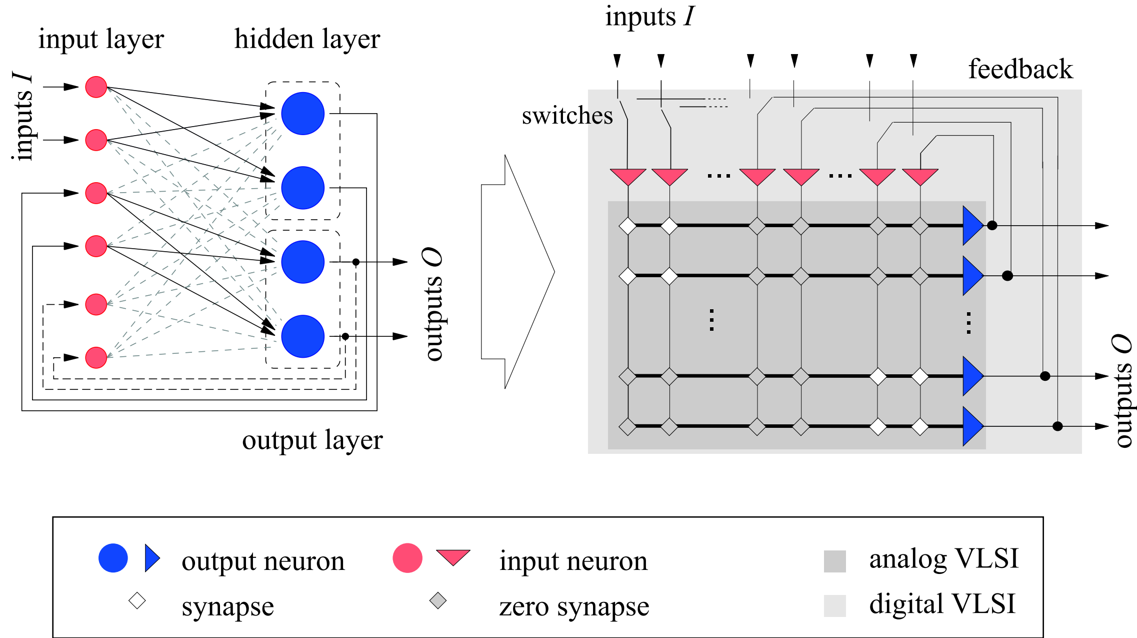


Figure 1.3: Multiple layers or even recurrent connections can be realized in the array structure if switchable connections allow to feed back the output to the input layer while setting appropriate synaptic weights to zero. Adapted from [184].

Having mentioned the inherent limit to linearly separable classification, a single-layer perceptron paradigm would restrict the neural network in its generality[3]. If extended by feedback connections, i.e., connections to transport the binary signals of the output neurons back to the input layer, the presented fully-connected single-layer perceptron can be configured to represent a multi-layer perceptron or recurrent network (this is illustrated by Fig. 1.3). To allow this behavior, it is necessary to configure the feedback connections and to set appropriate weights to zero in order to restrict the input signals to activate only synapses of certain output neurons. The necessary condition for this to work properly is the time-discretized operation of the network which allows to evaluate the various layers of a network in consecutive loops. One of these loops is called a *network cycle*. The evaluation of one layer consequently has to be assigned a time Δt; the network then operates at a frequency of:

$$f_{\text{net}} = \frac{1}{\Delta t}. \tag{1.2}$$

With this, the network response yields:

$$O(t+\Delta t)_i = \Theta\Big(\sum_j \omega_{ij} I(t)_j + \sum_k \omega_{ik} O(t)_k\Big). \tag{1.3}$$

[3]Interestingly, many real-world benchmarks problems have a large proportion of linearly separable data which makes single-layer perceptrons applicable. Furthermore, experiments with the presented ANN ASIC show that a majority voting between independently trained single-layer perceptrons can improve classification performance beyond the limit given by the amount of linearly separable data [86].

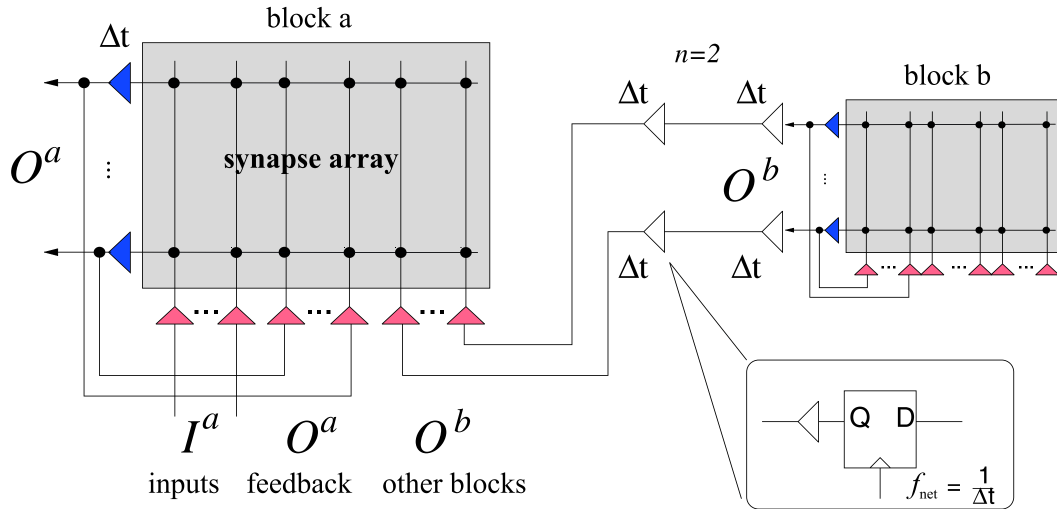


Figure 1.4: Multiple network blocks can be operated synchronously when the propagation delay is an integer multiple of the network frequency: The absolute delay may be large. Figure taken from [184].

Of course, the inputs to one network block are not restricted to come from the same block only. They rather can be provided from other, similar network blocks as shown in Fig. 1.4. If those network blocks are operated at the same network frequency, the network response for one block (here, block *a*) can be consistently described by the following equation (the superscripts determine the block):

$$O(t+\Delta t)_i^a = \Theta\left(\sum_j \omega_{ij} I(t)_j^a + \sum_k \omega'_{ik} O(t)_k^a + \sum_l \omega''_{il} O(t-n\Delta t)_l^b \ldots\right). \quad (1.4)$$

The last term shows an exemplary connection from a second network block b (O_l^b).

An intriguing feature is that—as long as the propagation delay from distant blocks is an integer multiple of the network frequency—the absolute delay is irrelevant for a synchronous operation. It is this property that allows the scaling to large networks.

Variable Network Resources Since some perceptron training algorithms require continuous neuron activation functions (e.g., the prominent back-propagation algorithm [178]), many perceptron implementations do not restrict themselves to binary inputs/outputs. Unfortunately, the multiplier circuit then needs to be incorporated into each synapse if the full parallelism is to be exploited. Several examples with fully analog multipliers in the synapses are found in literature: ETANN [91], BELLCORE [3], and others [182, 144]. The largest problem of multiplier circuits is their susceptibility to fixed-pattern noise which results in offsets and gain errors. Another obvious disadvantage is the area penalty of synapses that need to accommodate the weight storage and the multiplier circuitry. Furthermore, it is difficult for analog multiplier circuits to achieve simultaneously low power consumption, high dynamic range, and a good noise resistance [78]. An alternative strategy is to use discretized multi-value inputs and have a simplified multiplier circuit in the synapse, e.g., a multiplying digital-to-analog converter (MDAC) in the AT&T ANNA [25] or weights stored in a binary weighted fashion [70] which releases the design from the need for data-rate analog-to-digital conversion (ADC).

Indeed, the latter concept can be successfully adopted for the binary inputs propagated here: A straightforward way to feed multi-valued signals is to consider n inputs as a group encoding

an n bit integer. The integer value then needs to be translated into an analog network activity according to the significance of the representing bits. The least significant bit (LSB) induces ω, the next bits 2ω, 2^2ω and so on. The most significant bit (MSB) induces 2^{n-1}ω into the network:

$$\omega_i = 2^i \omega \quad \text{for} \quad 0 \leq i < n. \tag{1.5}$$

Essentially, this is a digital-to-analog conversion (DAC). Depending on the weight quanta ω and the number of bits n, the maximum induced activity of a multi-valued input is $(2^n - 1)\omega$.

Similar to the pooling of binary inputs for multi-valued signals, it is possible to group McCulloch-Pitts output neurons to act as an m bit integer output in the range of $[0, 2^m - 1]$. The task to be performed by this group of neurons is to measure the analog network activity present at their inputs and represent this activity as an integer. Basically, they have to perform an analog-to-digital conversion. Although a single neuron only performs a threshold discrimination, it is possible to configure a recurrent topology with self-inhibiting feedback in order to operate similar to a successive approximation ADC [215].

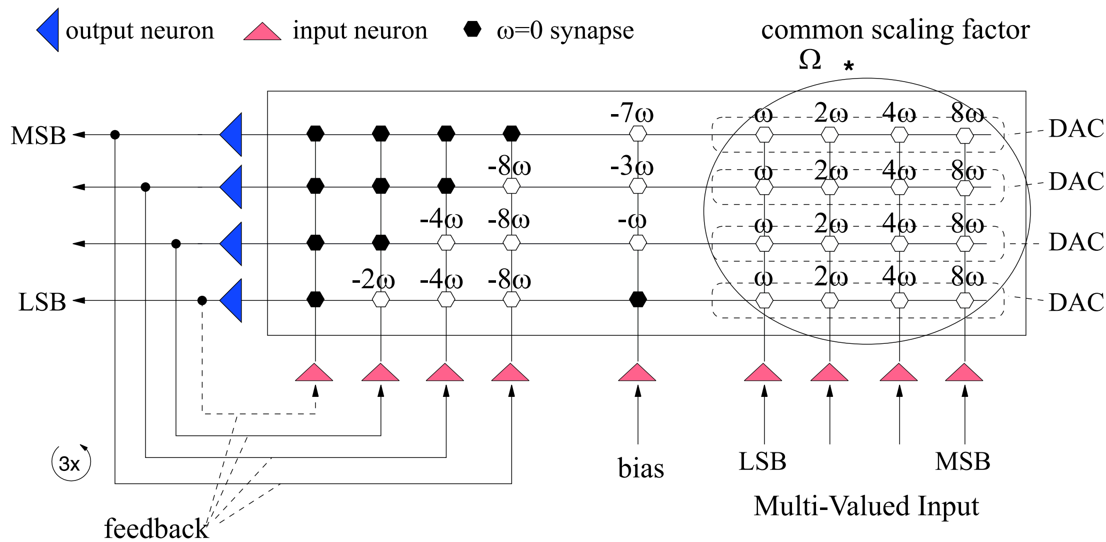


Figure 1.5: Network configuration of a 4-bit input being connected to a 4-bit ADC by a 4-to-4-bit synapse of weight Ω.

Fig. 1.5 depicts an exemplary configuration of a network that has a 4 bit multi-value input and extracts the activity by a 4 bit output neuron. It is to be noted that the activity induced to the pooled output neurons needs to be the same (for the same multi-value input) and has to stay constant over the course of the successive approximation.

In particular, the analog-to-digital conversion is achieved by the following procedure: One binary input is used as a bias (constantly activated) and adjusts the threshold values for the participating bits according to their significance, i.e., for the LSB 0, for the next bit ω and for the MSB $(2^m - 1)\omega$. After one network cycle, the MSB has reached its final state and subtracts $2^m\omega$ from all other bit lines if active. After the second cycle, the second most significant bit is stable and will go on to adjust the lower significant bit lines and so on. This process is completed after m network cycles. The easiest way to ensure stability of the solution is to prohibit feedback connections of bits to themselves or higher significant bits, i.e., the upper left and the diagonal of the weight matrix need to be zero.

Even though the relative weights are fixed for the pool of synapses inducing a n bit input to the neurons performing the m bit successive approximation, it is possible to change their absolute

value by introducing a common scaling factor $\Omega \in [-1,1]$. Essentially, this pool of synapses then can be considered an n-to-m bit synapse of weight Ω; the case $n = m = 1$ is the elementary synapse and $\Omega = \omega$.

If the number of bits for the inputs is chosen equal to the number of bits used in the output representation , the outputs can be directly fed to other inputs and arbitrary topologies of these so-called *variable network resources* can be realized in the network block architecture. On the other hand, the number of bits for the input and output do not have to be the same; a prominent case is $n \neq 1$ and $m = 1$ where multi-bit inputs are fed to a single McCulloch-Pitts neuron. Yet, there is no restriction to the combination of multi-valued inputs of different bit resolution.

Binary inputs therefore are not a restriction for interfacing multi-valued inputs, quite to the contrary, they reduce synapse complexity which allows smaller synapse circuits. This is an important issue for the reconfigurability of the network topology; some implementations even limit the achievable network topologies as to minimize the amount of unused synapses, i.e. area consumption [182]. As it can be seen from Fig. 1.5, synapses indeed remain unused, i.e., are set to zero. Similarly, Fig. 1.3 illustrates that strictly layered feed-forward topologies as well leave parts of the synapse array unused. An immediate consequence is that it is desirable to have small individual synapses in terms of area coverage and, most importantly, that inactive synapses do not contribute to the power consumption.

Experimental results for variable network resources are shown in appendix D.

1.2.3 VLSI Implications

The most important implication of binary inputs is the major simplification they introduce to the VLSI design: in order to accumulate the postsynaptic activity only the sum of the active weights needs to be calculated, and not the sum of weights multiplied by the value of the respective inputs (multiply-accumulate). An analog summation, furthermore, has to deal only with additive offset mismatches, whereas an analog multiplication is affected by offsets in the inputs and additionally a mismatch in the gain [115]. A purely additive offset can easily be compensated for since it is constant, i.e., it is independent of the input data and the weight value. This is not true for the gain and offset errors in a multiplication which affects training performance even in on-chip setups [49, 142]. Circuit precautions may be introduced to compensate these effects, but those increase the circuit complexity [33, 9, 59].

The choice for McCulloch-Pitts neurons reduces the evaluation of the summed activity to a comparison task. If the activity is above a certain threshold, the neuron will fire, otherwise it remains silent. There are different ways to efficiently implement a summation, two possibilities are: First, if the weights are represented by charges, a capacitor can be used to accumulate them. Second, if currents are used to encode the weights, a property ensured by Kirchhoff's current law can be exploited: Currents that flow through the same node sum up. The thresholding, finally, is easily implemented by a voltage comparator.

Charge-Sharing Neuron Implementing the network evaluation by the charge-sharing approach is straight forward and can be realized quite efficiently. Fig. 1.6 illustrates the schematic setup. First of all, a capacitor in each synapse is required to store its weight. It can be easily programmed by charging the capacitor to a certain potential. If several of these weight capacitors[4] are connected, charges will flow between them until a common potential is achieved, i.e., the weight

[4]Since it is not desirable to recharge the weight capacitors after each leveling, one additional capacitor per synapse should be used.

averaging is performed. In order to have positive and negative weights, an intermediate potential is chosen to be zero.

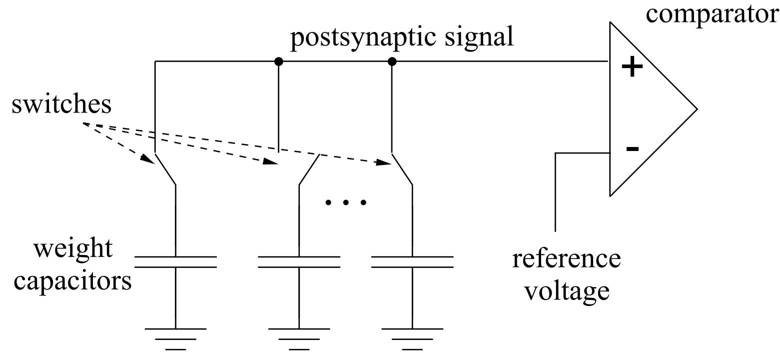


Figure 1.6: Operation principle of a charge-sharing neuron.

If the same potential is used as the reference voltage at the voltage comparator in the neuron, the activity evaluation can be realized in two phases: First, the postsynaptic line gets precharged to the threshold voltage which essentially resets the neuron. In a second phase, the weight capacitors in the individual synapses are connected to the postsynaptic line in accordance to the input. In effect, the potential of the postsynaptic line will yield:

$$V_i^{\text{postsyn}} = \frac{\sum_{j=1}^{N} I_j Q_j}{\sum_{j=1}^{N} I_j C_j}, \quad I \in \{0,1\}, \tag{1.6}$$

where N is the number of inputs per neuron, C_i are the capacitances of the weight capacitors, and Q_i the amount of charge stored on each. Finally, if the postsynaptic voltage is higher than the reference voltage, the neuron fires.

In practice, Eq. 1.6 exhibits the drawback that the influence of one weight is dependent on the number of simultaneously activated synapses. While in the ideal case it can be shown that this does not cause a premature flip in sign, already the effect on the amplitude may lead to inaccurate fire decisions. In case there are precalculated weights to be used, e.g. for variable network resources, two neurons each are combined to alternately switch which keeps the total capacitance constant.

This neuron principle has been tested in an initial mixed-signal prototype ASIC [186]. It not only allowed to evaluate the performance of this neuron implementation but also made it possible to test the network block architecture. Details on the realization of this concept are given in Sec. 1.3.

Current-Sum Neuron In contrast to the charge-sharing neuron, the current-summation can be easily combined with a differential approach, which makes the zero reference obsolete and eliminates one noise source. Fig. 1.7 shows a simplified visualization of the setup realized in a second mixed-signal ANN ASIC [187]; it is described in detail in Ch. 3. The amplitude of the weight is stored in an analog current memory and the sign is provided by a switch directing the current to one of the postsynaptic branches, I_+ or I_-. The currents on the postsynaptic branches are the sum of the active (as determined by the on-signal from the input neuron) synapses of respective sign:

$$I_{\text{neg}} = \sum_{i}^{N} |\omega_i| I_{\text{syn}}^{\max}, \quad \forall i \in \{j| -1 \leq \omega_j < 0\}, \tag{1.7}$$

$$I_{\text{pos}} = \sum_{i}^{N} |\omega_i| I_{\text{syn}}^{\max}, \quad \forall i \in \{j| 1 \geq \omega_j > 0\}, \tag{1.8}$$

with $I_{\mathrm{syn}}^{\mathrm{max}}$ being the maximum current sunk by a synapse. An inactive synapse, triggered by an inactive input neuron, gets connected to a third current line, I_{park}. This minimizes potential changes in the output node of a synapse and accelerates the operation.

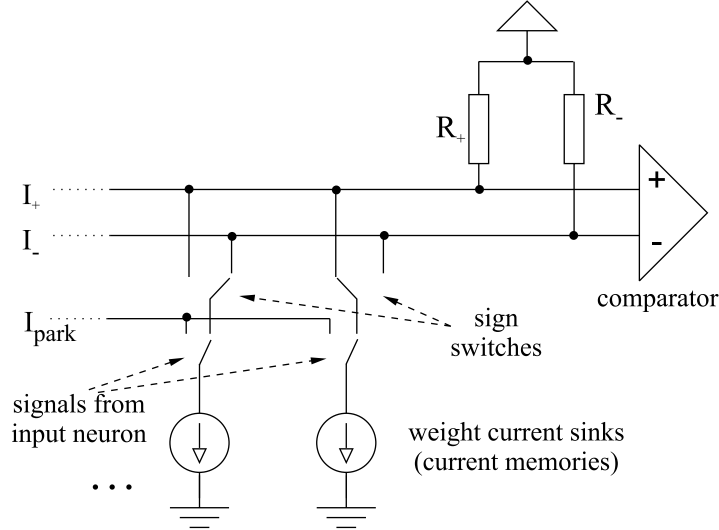


Figure 1.7: Operation principle of a differential current-sum neuron.

Before the comparison in the output neuron is performed, the excitatory and inhibitory currents are converted to voltages by resistors (R_+ and R_-) according to Ohm's law:

$$U_{\mathrm{neg}} = V_{\mathrm{dd}} - R \cdot I_{\mathrm{neg}}, \tag{1.9}$$

$$U_{\mathrm{pos}} = V_{\mathrm{dd}} - R \cdot I_{\mathrm{pos}}. \tag{1.10}$$

Weight Storage In order to perform a parallel evaluation of the postsynaptic activity, the weights need to be present in each synapse. And since the major feature of neural networks is their learning capability, the weight values need to be variable (during training) yet viable (during recall). Different approaches therefore have been utilized in earlier neurochips, such as classical EEPROM-based weight storage [91] or advanced versions that extend the underlying floating-gate principles to allow simultaneous weight update and read out in order to allow incremental online learning [47]. This way, the latter approach tries to account for longer programming times and higher programming voltages (above V_{dd}) by only doing small weight changes. Even though a successful adaptation to standard CMOS technology [48] has been shown, usually dedicated non-standard CMOS processes are required that accommodate the required voltages.

In a setup where the learning algorithm is not to be implemented on-chip and rapid weight changes may occur in training (e.g. evolutionary chip-in-the loop training), it seems more sensible to employ a split weight storage: In the synapses an easily changeable working copy of the weight is provided which allows a sufficiently long recall phase but eventually needs to be refreshed. The ANN ASIC based on the charge-sharing neuron showed that the leakage currents of the utilized CMOS process technology are such that capacitances in the 50-100 fF range, which can be efficiently realized by gate capacitances and allow small synapses, are viable enough to be trained and finally used during recall phase [186]. Nevertheless, after a certain time the weights need to be refreshed. This can be realized by introducing an digital weight storage as to provide a fast and long-lasting memory, and digital-to-analog converters can be used to accordingly load the weight capacitors in the synapses.

While in the ANN ASIC based on charge-sharing the long-term weight storage as well as the DA conversion was done off-chip, the on-die integration of the DACs has advantages: it reduces the interface of the overall chip to digital communication which allows a fast and noise-resistant operation. The current-sum prototype therefore integrates the DAC on-die, yet the digital long term storage there as well remains off-chip; this simplifies the exploration of different training strategies which can more easily be varied in software or programmable logic.

The reduced susceptibility to noise of the on-chip generation of the weights increases the analog representation of the weights. The reporting in literature on the minimal weight resolution requirements for training neural networks differs and ranges from 2 to 16 bits [142], whereby a 16 bit accuracy is only necessary for classical back-propagation training [11]. Especially, chip-in-the-loop strategies which only operate networks in forward propagation require much less accuracy and content themselves with as few as 2 to 3 bits [58, 56]. Compared to numerical tasks in computer algebra even the 16 bits for back-propagation are low requirements; according to Vittoz [218] perception-type tasks, which neural networks are dedicated for, seem to have an inherently lower requirement for precision than tasks aiming at the perfect restoration of information.

For some common benchmark problems theoretical worst-case limits for the minimal weight resolution have been calculated independently of the training strategy. These results show that they are problem-specific, yet well below 16 bits; using less rigid bounds the examination yields 8 to 12 required bits and less if a small percentage of misclassifications is allowed [7].

1.3 Realization

1.3.1 A Neural Network Experiment

The various hardware and software components presented in this thesis may be understood as parts of a *neural network experiment*, the main goal of which is the research on hardware adaptations of neural architectures by providing a framework that allows to readily exchange the substrate, the training and operation strategy, as well as the application.

This thesis will give a detailed description of the realization of the presented network paradigm and the experimental framework to assess its capabilities. Throughout the design and construction, a major emphasis has been put on interchangeability and reusability of its components. This modularity will allow the extension of the experiment to other network paradigms: A neurochip aiming at an efficient silicon adaptation of integrate and fire neurons [185] is currently under development and will initially be operated using the infrastructure presented here.

Design and construction of this neural network experiment are a collaborate effort with current and former researchers of the Electronic Vision(s) group contributing ASIC design, PCB development, logic programming, software engineering, and testing. The names of the contributing researchers are given where appropriate.

Within the scope of this thesis, hardware and software components have been developed which contribute to the experiment. This comprises the participation in the design of the ANN ASIC presented in Ch. 3, the development of peripheral electronics detailed in Ch. 4, and contributions to a modular software framework described in Sec. 4.3. Similarly, a substantial effort was dedicated to the testing of the individual components and their integration to an overall system in order to accomplish several operational platforms for different neural network experiments[5].

[5]Due to the modularity of the experimental framework it is even possible to use the infrastructure for other mixed-signal applications. Other projects of the Electronic Vision(s) group, such as research on evolvable hardware [117] or development of optical sensors with logarithmic response [28] employ some of the infrastructure.

Lastly, the experimental application pursued in this thesis adapts a recent strategy to operate recurrent neural networks to a hardware implementation. Since this paradigm does not rely on stable states, it promises some inherent robustness to substrate variations and faults throughout operation. It is explained in detail in Ch. 2 and experimental results are presented in Ch. 5.

1.3.2 ANN ASICs

Central to the neural network experiment are the ANN ASICs. Currently, two different mixed-signal ANN ASICs can be accommodated: One that implements a single network block of charge-sharing neurons, the *EvoOpt*[6] chip [186]. And a second which implements four interconnected network blocks based on the current-summing neuron [187, 184]. The latter—the *HAGEN*[7] prototype—is explained in detail in Ch. 3. Tab. 1.2 lists the features of both ANN ASICs.

Property	EvoOpt Prototype	HAGEN Prototype
process features	0.35 μm, 1 poly, 3 metal	0.35 μm, 1 poly, 3 metal
die/core size [mm^2]	2.5×2.5 / 2×2	4.1×3 / 3.6×2.5
synapse array/network block size [mm^2]	0.7×0.7 / 0.7×0.9	1.1×0.8 / 1.5×1
synapse size [μm^2]	10×10	8.7×12
blocks/output neurons/synapses	1/64/4096	4/256/32768
supply voltage	3.3 V	3.3 V
max. network frequency f_{net}	100 MHz	50 MHz
CPS	0.4 Teracps max.	1.64 Teracps max.
CUPS	16 Megaweigths/s max.	400 Megaweigths/s max.
weight resolution	up to 6 bit (measured)	10 bit (nominal) + sign

Table 1.2: Nominal specifications of the ANN ASIC prototypes.

Both ASICs are prototypes; the implementation therefore is primarily limited by the available silicon area. This area limit directly translates into a limitation in the number of available synapses. Nevertheless, the fundamental assumptions of the proposed network concept can be evaluated.

The EvoOpt prototype for example allowed to assess the following aspects:

- implementation of an efficient synapse for the use with charge-sharing neurons [186];
- trainability of multi-layer and/or recurrent topologies with chip-in-the-loop strategies in the presence of hardware non-idealities [88];
- variable network resources trained by an evolutionary algorithm [193].

Since both ASICs are designed in the same process technology, the feasibility of capacitive weight storage shown in the EvoOpt design allowed to optimize the synapses for the current-sum implementation.

A major goal of the current-sum approach, finally, was to realize a better accuracy while maintaining the successful concept of network blocks. With the HAGEN prototype therefore the following has been achieved in addition:

[6] The name EvoOpt is derived from *Evolution* and *Optical*; it resembles the fact that indeed two designs were submitted on the same die: a neural network suited for the training by evolutionary algorithms and a optical sensor test design.

[7] The acronym stands for Heidelberg Analog Evolvable Neural network; an obsolete internal name of the ASIC is *ANNA* (Analog Neural Network Array); yet it is reminiscent in some internal naming schemes.

- implementation of an efficient and accurate synapse for use with current-summing neurons [187];
- evaluation of the mixed-mode operation principle for a fast and accurate weight storage and activity evaluation by the neurons [187];
- demonstration of low power operation with a high-speed interface;
- evaluation of the mixed-signal approach for encapsulating the analog VLSI into blocks while only using digital signals for the communication with the outside [184];
- competitive trainability on standard classification benchmarks by specialized evolutionary algorithms [89, 87];
- feasibility of scalable networks by interconnecting several network blocks [87, 195, 57];
- variable network resources using precalculated weights [194].

1.3.3 Experimental Framework

A first realization of the experimental framework is based on a custom-built PCI-based adapter that hosts a programmable logic in form of a *field programmable gate array* (FPGA), a local memory, and circuits for mixed-signal testing [18]. This adapter allows to interface various ANN ASICs and a general purpose computer (see Fig. 1.8); these components comprise the *substrate level*.

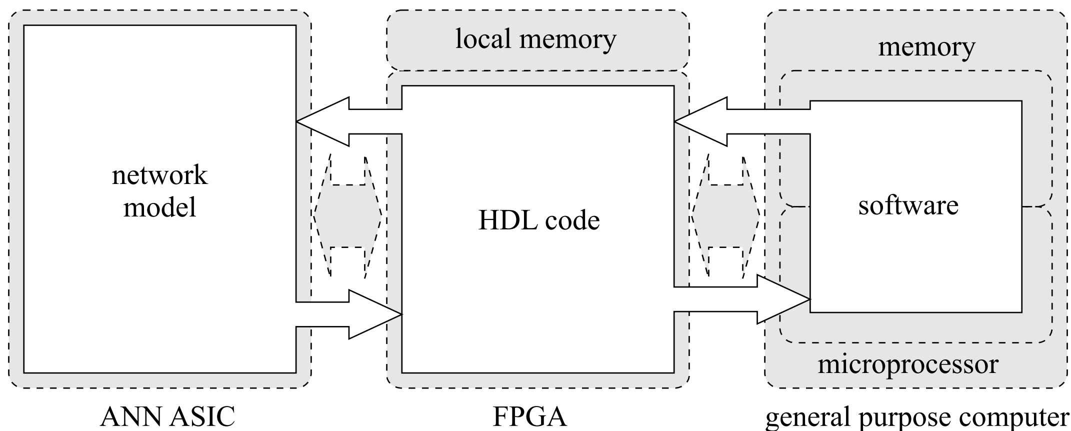


Figure 1.8: Components of the experimental framework and a possible realization. The actual implementation (gray boxes) and their physical interaction is mostly independent from the functionality (white boxes).

Not only on the substrate level but also on the *functional level*, a modular approach has been followed: The neural network has an abstract representation given by its modular network blocks, that allows the configuration, training, and operation the network independent of its actual hardware realization. This separation is realized in software as well as in the programming of the reconfigurable logic (as given by the code in a *hardware description language* (HDL)). In software, a *hardware abstraction layer* is responsible for accommodating the interaction with a hardware organized in the network block structure. Similarly, the HDL code is separated into an entity operating the respective ANN ASIC and other entities serving a general purpose such as interfacing

the software, operating a local memory, or logic supporting the algorithmics of the software. The realization of the substrates as well as details on the functional implementation are given in Ch. 4.

Due to its modular implementation, the experimental framework allows a rapid extension and migration of functionality between substrates. Examples for this are the following achievements:

- While the work on the experimental framework started with the development of the charge-sharing based ANN ASIC, the successor ASIC HAGEN was integrated by essentially only changing the chip-specific parts of the peripheral electronics, the interfacing by the programmable logic, and the low-level chip-representation in the software. The PCI-based FPGA adapter as well as major parts of the software, especially the training strategies, are reused. Details are described in Ch. 4. The integration of any future type of ANN ASIC which is organized in the same block structure will be similarly straight forward.

- Certain parts of an evolutionary training strategy have been migrated from the software to the HDL code to accelerate the training [189]. More details can be found in Sec. 4.2.3.

- A HDL entity that augments the network model implemented by the HAGEN prototype to emulate spike-based computing has been tested in software [31] before it was realized in the programmable logic. More details can be found in Sec. 4.2.3.

- A distributed system has been developed that integrates all substrates of the experimental framework, i.e., the ANN ASIC, programmable logic, and a microprocessor, on a single PCB [77]. Even though the electrical implementation changed, a new type of FPGA has been used, and the general purpose Intel-based PC has been replaced by an embedded PowerPC core, most of the HDL code and the software is reused (c.f. Sec. 4.1.5). This advanced implementation of the neural network experiment will be used to accommodate future ANN ASICs, e.g. [185].

1.3.4 Suitable Training Approaches

Essential to the operation of a neural network is the training: the functionality of a neural network is determined by adequately configured topologies and weights. Several lines of research are pursued that concern the feasibility of successful training in general, training strategies that are especially suited for the chosen network model, as well as training strategies that generate fault-tolerant architectures. To accommodate these different approaches, a modular software platform has been developed that allows a flexible yet efficient operation of the hardware and thus forms the basis for the operation of the neural network experiment. An introduction and description with emphasis on technical details is given in Sec. 4.3; a complementary description of other parts of the software can be found in [86].

The lines of research that are currently pursued comprise the following:

- An approach that was considered first—since it is especially suited to cope with the properties of an analog hardware—is the hardware-in-the-loop approach with an evolutionary training strategy. In terms of an easy usability for an end-user, this training strategy is the most advanced; it is shortly described in Sec. 4.4.1 since it is used to obtain some of the results in Ch. 5. A concise description can be found in [86].

- Another way to operate the ANN hardware is to use precalculated weights that can originate from any kind of algorithm. This approach relies on the knowledge of fixed-pattern effects in the synapses and neurons. A method to infer these values is introduced in [86] and shortly

described in Sec. 4.4.2. Precalculated weights are used for experiments on the variable network resources results presented in appendix D.

- Lastly, in the following a computing paradigm is described and adopted that uses transient states of a highly recurrent neural network. It essentially uses randomly generated weights and thus promises to cope with substrate variations and offer tolerance to faults throughout operation. The theoretical foundations are introduced in the next chapter and experimental results are presented in Ch. 5.

Chapter 2

Computing without Stable States

Inherent to the network model of the neural network experiment is the configurability allowing for a multitude of topologies. While the prototype implementation is extensively studied for feed-forward architectures in [86], here, recurrent topologies are investigated. The latter are considered most adequate for computations on time series, but their theoretical tractability and the feasibility of the training remain difficult. Recently, a strategy independently described by Maass et al. [131] and Jaeger [105] promises to harness the capabilities of recurrent neural networks without the difficulty in training: The recurrent neural network is used as a dynamical system perturbed by input; only a simple readout is trained to extract information from the transient behavior.

2.1 Recurrent Neural Networks

2.1.1 Finite Automata and Neural Networks

The 1943 article of McCulloch and Pitts [134], in many regards, laid the foundation for the research on neural networks. On the one hand, it modeled elementary aspects of neurons by simple threshold devices which allowed to assess the computational power of neural networks, especially aspects of computing logic predicates. On the other hand, it is considered to be the seminal paper of automata theory and neural networks. It was Kleene in 1956 [112] who interpreted the publication of McCulloch and Pitts and explicitely used the term finite automaton for a recurrent neural network. Later on Minsky [139] published a book in which he proved that there exists an equivalent recurrent neural network to each finite state machine. Indeed, a recurrent neural network is a realization of a finite state machine of the Mealy type [92], i.e., a finite automaton receiving input and producing output that is dependent on its internal state and the input. Fig. 2.1 shows a visualization of this.

Thinking of neural networks as being finite state machines is a powerful concept if they are to operate on time varying inputs. Compared to feed-forward networks which need to be augmented by some kind of memory—e.g. a tapped delay line—in order to have time varying inputs transformed to static snapshots, recurrent networks provide built-in memory. They are capable of responding temporally to an input by changing their internal states. Depending on the type of

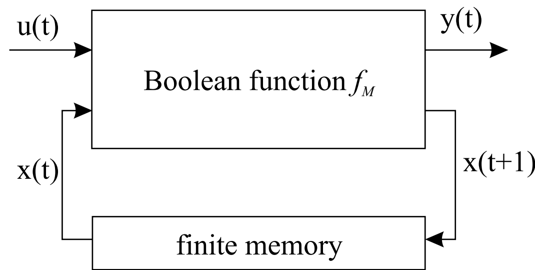


Figure 2.1: Schematic diagram of a Mealy type finite state machine implemented as a sequential machine. The output *y(t)* at some discrete time *t* is dependent on the input $u(t)$, the internal state $x(t)$, and the boolean function f_M realized by the machine. Figure after [152].

feedback, they are supposed to implement memory more efficiently than specifically constructed finite memories used for the temporal processing with feed-forward networks [81].

Unfortunately, the knowledge that there exists a recurrent neural network which is equivalent to any finite state machine[1] does not lead to a straight-forward way to construct it. Thus, the question may be rather whether a given recurrent neural network can be trained to simulate a specific automaton.

First successes were achieved by Hopfield [93] who applied methods from statistical mechanics to train a specific type of recurrent networks. The fully-connected, symmetric weights used by Hopfield guarantee that the network dynamics exhibit fix-point attractors which represent stable states in which information can be stored. Despite this achievement, the research on training recurrent neural networks is only preliminary compared to successes achieved with feed-forward architectures. Unlike the popular back-propagation algorithm for multilayer perceptrons, there exists no standard training algorithm for recurrent networks [106]. Even though back-propagation has been modified to be applicable to recurrent networks (*back-propagation through time* essentially unfolds the circular connections with the help of multiple copies and redirected connections) [223], results in practice are non-trivial to achieve since a convergence to a local error minimum is not guaranteed [106]. Other gradient-descent training methods, such as *real-time recurrent learning* (RTRL) [227] or *extended Kalman filtering* (EKF) [204], improve the training success but are computationally more expensive and eventually are similarly difficult to control [106, 81].

All these approaches have in common that they store information on past inputs in stable states. In case the of Hopfield networks it is even necessary to prevent new inputs from being presented before a stable state has been reached which prevents real-time computation on continuous time series.

2.1.2 The Liquid State Machine Approach

Independently, in 2001 Maass et al. [131] and Jaeger [105] proposed a strategy for time series computing with recurrent neural networks that does not rely on stable states. Instead, all states—except for a ground state with no activity—are transient and information on past inputs has to be extracted from the trajectories of the network dynamics. For this extraction one or more 'observers' and not the recurrent network itself are trained. Once set up, the recurrent neural network essentially represents a dynamical system that is perturbed by the input. If the network dynamics

[1]While this statement and proof originate from Minsky [139] in 1967, it was later on proven that even all Turing machines—essentially automata with infinite memory—may be simulated by fully connected recurrent neural networks built on neurons with sigmoid activation function [202].

are appropriately chosen, Maass et al. and Jaeger demonstrated that it suffices to train a memory-less[2] and even linear readout, e.g. a single-layer perceptron, to extract non-trivial computations from the recurrent network. Compared to training approaches that need to modify the network dynamics, training a memory-less readout has the immediate advantage of ensured convergence (if there exists a solution) and robustness of the training [154]. In the following, the terminology of Maass et al. is followed who coined the term *liquid computing* for this type of computation; proper reference to the work of Jaeger is given in context.

Theoretical Framework The findings of Maass et al. [131] and Jaeger [105] raise the question of what exactly can be inferred from the transient dynamics of a recurrent network? A theoretical answer to this was given by Maass et al. [131]: Similar to the Church-Turing hypothesis which guarantees universal computing powers to Turing machines, it can be proven that—under idealized conditions—the class of machines comprised of a dynamical system and a memory-less readout is universal for computations on time series. This is to say that any[3] mapping F from an input stream $u(t)$ to an output stream $y(t)$, which in engineering terms is referred to as a *filter*, can be approximated by a machine of this class to any desired degree of accuracy. According to the terminology introduced by Maass et al. such a machine is called *liquid state machine* (LSM).

A representative M of this class of liquid state machines provides a filter[4] $L^M : U^n \to (\mathbb{R}^\mathbb{R})^m$, called the *liquid filter* or *liquid*, that maps an input function $u(t)$ onto an output function $x^M(t)$

$$x^M(t) = (L^M u)(t), \tag{2.1}$$

and a memory-less map $f^M : \mathbb{R}^m \to \mathbb{R}$ that evaluates this output function at each time t to yield a target output $y(t)$ according to

$$y(t) = f^M(x^M(t)). \tag{2.2}$$

The output result $x^M(t)$ as defined by Eq. 2.1 of a liquid filter L^M operating at some input u at a given time t is referred to as the *liquid state*. Fig. 2.2 illustrates the components of a liquid state machine.

In order to prove that only a finite number of basis filters (from which L^M is comprised) is required to approximate any filter F that maps $u(t)$ to $y(t)$, two requirements need to be met [131]: First, the set of basis filters $\mathcal{B}$ has to provide a point-wise *separation property*; in other words, for any two functions $u(\cdot), v(\cdot) \in U^n$ with $u(s) \neq v(s)$ at some time $s \leq t$ there exists a basis filter $B \in \mathcal{B}$ with $(Bu)(t) \neq (Bv)(t)$. A trivial example for a class of filters is the set of delay lines or the set of all linear filters. Second, the readout has to have the *universal approximation property* as for example provided by multilayer perceptrons [97].

Practical Implications While the theorem proven by Maass et al. [131] provides a mathematical framework to properly describe this type of machine it does not give practical hints to the actual implementation. In a worst case scenario, e.g., a setup in which the filter is chosen to be comprised of delay lines, it is completely up to the readout to compute the target function. In this case merely

[2]The term memory-less readout is used in the sense that the readout at some time t has no access to information prior to t.

[3]Precisely, there are two restrictions to the type of filters that can be approximated: First, the filter F has to be time-invariant which means that a time shift in the input stream can be expressed by the same time shift of the output function. Second, filters are excluded which require an exact knowledge about the infinite history of the input stream in order to compute the output at some time t. For the latter the notion of *fading memory* was introduced in [26]. Natschläger et al. [154] argue that the class of computable filters comprises all relevant ones for biological and engineering applications.

[4]With $\mathbb{R}^\mathbb{R}$ being the set of real-valued functions, U a special subset thereof (c.f. [131]), and n, m the dimensionality of input and output functions respectively.

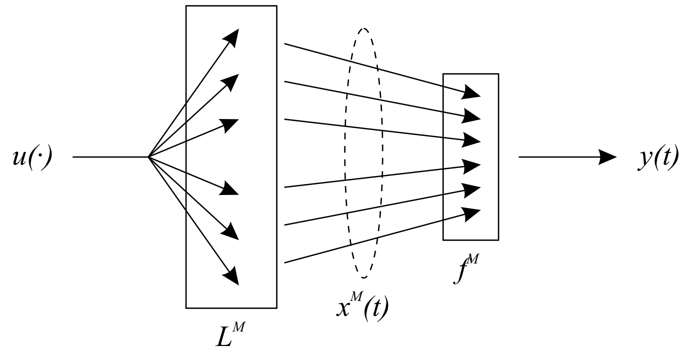


Figure 2.2: Schematic diagram of a liquid state machine mapping the input stream $u(t)$ to some target function $y(t)$. In order to do this, the input stream $u(t)$ is injected to the liquid filter L^M which generates at each time t a liquid state $x(t)$. This is the input to a memory-less mapping f^M which generates the target function $y(t)$. Figure after [131].

the classical approach to time series computing by feed-forward networks is recovered (with the difference that the recurrent network forms the finite memory and the readout takes on the role of the universal approximator realized by the feed-forward network).

On the other hand, the simulation experiments performed by Maass et al. and Jaeger showed that if the liquid filter is realized by a recurrent neural network it can be surprisingly simple to extract various computations by only linear readouts. Maass and his co-workers examined biologically inspired, continuous time neural microcircuits of leaky integrate and fire (I&F) neurons [131, 132]. In a benchmark of speech pattern recognition proposed in [95, 96] they showed competitive performance of an LSM comprised of a liquid with 135 I&F neurons and readout using a single I&F neuron for each digit trained by a linear regression [132]. Jaeger, on the other hand, examined time-discrete recurrent networks of sigmoid neurons with linear readout and showed how to do system identification and time series prediction [105].

The restriction to a linear readout allows the conclusion that the liquid indeed does more than a mere temporal integration if the LSM successfully reproduces target values that are non-linear in the input. Such a boosting of the capabilities of a linear readout is known to be achievable by augmenting a (linear) readout with a preprocessing that computes non-linear combinations of the input or that projects the input into a high-dimensional space [175, 216]. According to a mathematical theorem [40], the probability for a pattern classification problem to be linearly separable grows with the dimensionality of the space cast to by a non-linear mapping. While *support vector machines* (SVM) [24] require specifically constructed kernel functions in order to contract the non-linear mapping and linear readout in a dual space representation, preventing the dimensionality of the intermediate feature space from affecting the computational performance (see e.g. [36]), in the liquid approach the kernel is realized by the liquid and the dimensionality naturally arises from the non-linearity of the neurons and the complex dynamics of the recurrent topology.

One of the most important findings from the experiments of Maass et al. [131] and Jaeger [105] is that temporal integration and kernel-like boosting capabilities can be achieved robustly by almost randomly created networks. Yet, they observed that it is necessary to ensure that the network dynamics exhibit a property which Maass et al. refer to as *fading memory* and Jaeger as the *echo state property*. The notion of Maass et al. [131] resembles the fact that the filter L^M implemented by the recurrent neural network should only depend on the current input and some finite history of it, just like the original filter F that has to be approximated by the LSM (see

footnote 3). Similarly, the notion of Jaeger implies that state differences caused by different input histories vanish over time [105]. In certain types of networks, such as the one that will be adapted for a liquid state machine implementation on the ANN ASIC presented here, this can be directly correlated to ordered and chaotic network dynamics, i.e., fading memory is a property of ordered dynamics [21]. This will be described in detail in Sec. 2.3.2.

In their experiments, Maass et al. and Jaeger furthermore were able to demonstrate that the temporal integration and the kernel-like boosting properties of a liquid are generic; in other words, the same liquid can be used to simultaneously extract many different target functions from it by individually trained readouts. Thus, the name *liquid state machine* implies similarity to a finite state machine, yet it is more: Unlike finite state machines, where the states and transitions are specific to the task, a liquid state machine simultaneously contains a multitude of transient states; it is the readout[5] which picks out the relevant ones.

2.2 Exploring Liquid Computing in Hardware

2.2.1 Motivation

The liquid state machine approach offers a promising alternative for harnessing the computational power of recurrent neural networks. By refraining from changing the dynamics, it reduces the training to a robust and well-defined task, still it makes use of the kernel properties inherent to the recurrent dynamics. In order to do so, the readout does not have to be given the fully defined dynamical state of the liquid, rather it may suffice itself with an observable part of the dynamical system: In the approach proposed by Maass et al., the readout only observes the firing condition of the I&F neurons at some time t which is far from fully quantifying the dynamics (the membrane potentials would be needed for that).

This property allows the use of a complex dynamical system without the need of explicitly having available all state information about it. It is therefore possible to realize the complexity (and thus the potential kernel-properties) *in* a physical system as long as the observables are meaningfully dependent on the internal dynamics. While support vector machines have to revert to a mathematical trick to circumvent the explicit representation of the high-dimensionality, the same is achieved if the liquid is realized physically. Consequently, liquid computing has an implicit connection to a hardware realization.

Moreover, liquid computing offers a strategy that may even profit from variations, inaccuracies, and noise as inherent to analog hardware: As long as these effects do not drive the dynamical system towards chaos or keep it at the ground state, spatial and temporal variety of the underlying computing elements can increase the richness of the dynamics. While Maass et al. [132] point out the analogy to the diversity found in biological nervous systems, liquid computing can be considered a hardware-friendly strategy for analog VLSI as well. Yet, not only the variety increases the complexity of the liquid filter but also the bare number of computing elements. Liquid computing therefore offers a way to make immediate use of the large number of available resources in a VLSI system.

The splitting of the computation into a dynamically driven non-linear system which is generic and a simple readout has yet another implication: Not only is it possible to simultaneously read out different target functions, but furthermore each readout has only very low computational demands. With a linear readout, only a scalar product has to be computed in order to get a response from the LSM at some given time t; additionally, this response is given at any time. Unlike attractor

[5]The readout clearly has no analog in the finite state machine picture since usually the finite state machine itself provides the desired output.

networks which need time to converge to an attractor state, the LSM essentially gives online answers. Maass et al. [130] emphasize this 'anytime computing' capability of LSMs in the context of biologically inspired neural microcircuits: They consider the LSM formalism and the computing paradigm therein to be more suited to explain the computational power of neural microcircuits than classical approaches based on sequential Turing machines.

Beyond the biological implications, a computationally inexpensive readout allowing online predictions in real-time (i.e., a response in some ensured finite time) makes liquid computing an interesting strategy for engineering applications. Jaeger [107] for example proposes to use it for channel equalization of high-speed data streams in wireless communication.

Lastly, adaptations of liquid computing in software simulated neural networks by Maass et al. [131] and Jaeger [105] showed that it is essentially possible to obtain suitable liquid dynamics by drawing the topology and weights randomly. Jaeger only controlled the scaling of the weights as to ensure a damping of the dynamics, while Maass et al. restricted the topology by connection probabilities and distributions from which the weights were drawn; the reason for these heuristic restrictions in both cases is to ensure the fading memory property of the liquid. The kernel functionality of the liquid thus is not specifically constructed, rather, it emerges quite robustly. This observation gives rise to the conjecture that the general kernel properties of the liquid do not rely on single, very specific connections or neurons but more on a distributed and redundant representation by populations of neurons. Consequently, it can be expected that a physically realized liquid has some inherent robustness to faults and imperfect substrates.

Implementing the liquid computing approach with the presented ANN ASIC HAGEN allows the investigation of these conjectures in a physical experiment. Ch. 5 will show that suitable liquids can be realized by a recurrent neural network topology provided by the HAGEN prototype. The first set of experiments (Sec. 5.2) is dedicated to exploring the liquid computing adaptation of the type of networks realizable by the hardware. The emphasis will be on the dynamics of the liquid. In the second set (Sec. 5.3), the liquid computing approach is assessed for its qualities in dealing with imperfect substrates—during training as well as during operation. While the generation procedure for adequate liquid filters has been studied earlier in software (see Sec. 2.3 for references and details), to the knowledge of the author, no systematic studies on the effects of hardware variations and the robustness to faults in a hardware realization of an LSM have yet been published.

2.2.2 *liquid*HAGEN

Essential to exploring the promise of a symbiosis between liquid computing and a hardware implementation is the physical realization of the liquid. Only an actual implementation allows direct observation of the interrelationship of hardware peculiarities and liquid dynamics. The readout, on the other hand, in a prototype system may be implemented in software as its functionality does not rely on the underlying substrate. Eventually, a dedicated digital circuit for computing a linear least-squares readout may be used or a perceptron-based readout implemented using a network block provided by the neural network ASIC.

Fig. 2.3 illustrates the *liquid*HAGEN setup. As shown, the liquid filter L is physically realized by a recurrent neural network on the HAGEN prototype. The readouts as well as the management of the input stream is done in software. Similarly, the generation of the liquid, i.e., the procedure to configure the recurrent network appropriately, is realized in software. This provides the flexibility to conveniently explore the liquid dynamics as described in Sec. 2.3.2. Details of the software implementation can be found in Sec. 4.4.3.

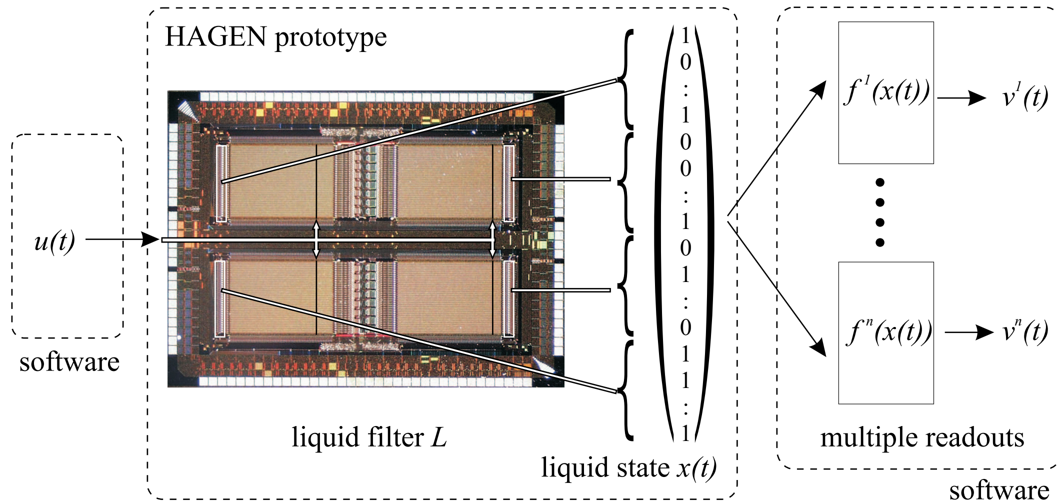


Figure 2.3: Schematic illustration of a liquid state machine comprised of a liquid realized on the ANN ASIC HAGEN and a readout implemented in software.

Realizing the Liquid Due to the nature of the network model provided by the HAGEN prototype, the liquid is a time-discrete recurrent neural network with McCulloch-Pitts neurons (see Ch. 1). Since the HAGEN prototype has four network blocks of 64 neurons each (see Ch. 3), the liquid state $x(t)$ accordingly is a vector of 256 binary digits encoding the firing state of each of the output neurons on some time interval $[t, t+\Delta t]$ with $\Delta t = 1/f_{\text{net}}$ according to Eq. 1.2. This binary vector then can be used by one or more readouts implemented in software.

The binary input neurons of the network blocks naturally suggest that the input time series is comprised of time-discrete binary values, i.e. $u(t) \in \{0,1\}^n$, where t is a multiple of the inverse network frequency, as above. With the variable network resource of Sec. 1.2.2 this can easily be extended to quasi-continuous time-discrete input streams. The time-discretization on the other hand only gives an upper limit to the maximum input frequency according to the sampling theorem. More important concerning the time scale is the following consideration: The liquid dynamics have to be such that its temporal integration capabilities, i.e. the short term memory, correlate with the temporal features in the input. As the next section and the experiments in Sec. 5.1 will show, there is a certain memory capability inherent to the type of liquid on the HAGEN prototype used here; the short term memory together with f_{net} fix the time scale of distinguishable input features in the input time series.

It turns out that the absolute time scale is important if a specific application is intended: If the hardware implementation of the liquid is used to interface some physical real-time data source, the time scales will have to match. The experiments presented here, which are meant as a proof of principle performed to analyze the substrate, are set up such that the input stream $u(t)$ is a recorded time-series. This allows the use of an arbitrary mapping between the actual time series and the time-discrete operation of the HAGEN prototype.

The realizable liquids are limited by the configurability and dynamic range of the utilized HAGEN prototype (it is discussed in detail in Ch. 3). Each of the four blocks is comprised of 128 inputs fully connected with 64 McCulloch-Pitts neurons. Since the feedback connections of the left two blocks are such that all 64 neurons can be fed to the inputs and an arbitrary[6] number of

[6]Precisely, only a finite number can be realized due to the decaying weights (c.f. Sec. 3.3.4); yet, if the network inputs and outputs are buffered while the weights get refreshed, an infinite number of cycles is possible.

network cycles can be performed, essentially any type of time-discrete recurrent network of up to 64 neurons can be implemented (the right two blocks have 32 feedback connections to itself). If all feedback connections are activated, the topology is completely determined by the weight matrix as described in Sec. 1.2.2. The remaining 64 inputs of each network block can be used to connect the input $u(t)$ or interconnect outputs of the other four network blocks of the HAGEN prototype. For simplicity in the technical realization, these inter-block connections are fixed, i.e., not necessarily all output neurons of one block can be interconnected to the inputs of another. Details of the routing scheme are given in Sec. 3.2.

In effect, not all conceivable recurrent topologies of 256 neurons can be realized by the HAGEN prototype, especially not fully connected ones. In [31] a more balanced connection scheme has been realized and applied within the *liquid*HAGEN setup. However, the restrictions imposed by the fixed routing scheme are negligible which can be understood by the sparse connectivity architecture resulting from the adopted network generation procedure.

Linear Readout On a given liquid L^M, a memory-less readout f^M has to be trained in order to allow the liquid state machine to compute a desired filter F that maps an input stream $u(t)$ to an output target stream $y(t)$. As discussed earlier, it is not guaranteed that this target function can be extracted easily from the liquid by the means of a simple linear readout which is trained by a well-defined least-squares regression. Whether or not it is possible is dependent on the kernel-properties of the liquid. But this very quality of the liquid is best assessed if the readout is kept linear: If a target function can be computed by the LSM which is a non-linear function of the input, it can immediately be concluded that the liquid is more than a mere temporal integrator.

In order to account for the fact that indeed the LSM may not be able to compute the target output $y(t)$, it is sensible to assign the name *target prediction*, $v(t)$, to the output generated by a certain readout. As illustrated in Fig. 2.3, there can be several simultaneous predictions for different target functions, each of which is generated by a separate readout. Especially, if the readout is implemented by a linear classifier, the target prediction of the LSM is basically a scalar product of the weights determined by a linear regression and the liquid state $x(t)$; it can be written as

$$v^k(t) = \begin{cases} 1 & \text{if} \quad \sum_i w_i^k x_i(t) + b^k \geq 0 \\ 0 & \text{if} \quad \sum_i w_i^k x_i(t) + b^k < 0 \end{cases} \tag{2.3}$$

where k is the number of the readout, w_i^k the ith weight of the readout k, and the sum goes over all entries of the liquid state vector, $1 \leq i \leq m$. The threshold b^k is added for stability reasons to the linear regression [167] and is computed as the $(m+1)$th weight for a constantly activated vector element, $x_{m+1} = 1$.

All weights w_i^k can be determined simultaneously in a straight forward fashion by a least-squares linear regression. The respective overdetermined set of linear equations is obtained in the following way: At times $t_1, t_2, \ldots, t_l$ the vectors $x(t_1), x(t_2), \ldots$ which describe the states of the liquid after the inputs $u(t_1), u(t_2), \ldots$ are taken and together with the outputs $y(t_1), y(t_2), \ldots$ of the K desired target filters F^k one can formulate the following set of linear equations:

$$\begin{pmatrix} x_1(t_1) & x_2(t_1) & \ldots & x_m(t_1) & 1 \\ x_1(t_2) & x_2(t_2) & \ldots & x_m(t_2) & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x_1(t_l) & x_2(t_l) & \ldots & x_m(t_l) & 1 \end{pmatrix} \begin{pmatrix} w_1^1 & \ldots & w_1^K \\ \vdots & & \vdots \\ w_m^1 & \ldots & w_m^K \\ b^1 & \ldots & b^K \end{pmatrix} = \begin{pmatrix} (F^1 u)(t_1) & \ldots & (F^K u)(t_1) \\ \vdots & & \vdots \\ (F^1 u)(t_l) & \ldots & (F^K u)(t_l) \end{pmatrix} \tag{2.4}$$

While the t_j can be arbitrarily sampled (they even do not have to be monotonically increasing) as long as the corresponding states and target values are taken, the number of sampling points l needs to be larger than the dimensionality m of the liquid state $x(t)$, i.e. $l > m$; for $l > m+1$ Eq. 2.4 is overdetermined.

In order to minimize the numerical errors, the least-squares solution to the overdetermined set of linear equations Eq. 2.4 is computed using a QR-factorization with Householder transformations and solving the resulting upper-triangular matrix equation by back-substitution. The adopted procedure is explained in detail in Sec. A.1 of the appendix. Details on the software implementation are given in Sec. 4.4.3.

2.3 Liquid Dynamics

2.3.1 Input Driven Networks

The key to successfully applying the liquid computing approach to a time series computation is the dynamics of the liquid. On the one hand, it has to be sensitive to salient features of the input stream in order to cause different trajectories of the liquid and have a readout distinguish them. On the other hand, Maass et al. [131] and Jaeger [105] empirically observed that the dynamics have to exhibit fading memory: In an experiment where the same liquid is started twice from the same initial state but with input streams $u(t), \hat{u}(t)$ that differ for some time, i.e. $u(t) \neq \hat{u}(t)$ for $t < t_0$, and later on are identical, i.e. $u(t) = \hat{u}(t)$ for $t \geq t_0$, the liquid dynamics should eventually be indistinguishable.

Maass et al. [131] managed to choose inhibitory and excitatory neurons and the distributions from which weights and connections were drawn such that their networks of spiking neurons exhibit a self-sustained activity which neither dies out nor gets out of hand due to an infinite gain in activity. They showed that this type of dynamics can be sensitive to inputs, thus different input streams cause different trajectories of the liquid state, but the trajectories do not diverge. More formally, they define a distance measure, $d(u, \hat{u})$, between two input streams of spikes $u, \hat{u}$ and use the Euclidean norm of the state difference, $|x_u^M(t) - x_{\hat{u}}^M(t)|$, as the distance measure for two liquid states at a time t. Their experiments show that the state distance is asymptotically proportional to the input difference, $|x_u^M(t) - x_{\hat{u}}^M(t)| \sim d(u, \hat{u})$.

Jaeger [105], on the other hand, proposed to ensure non-diverging trajectories in the recurrent network through damping: By choosing the overall scaling of the weights small enough, the activity of his randomly constructed recurrent networks eventually dies out if the excitation by an input is stopped.

Both strategies for generating appropriate liquid dynamics could have been adapted for examining the properties of a recurrent network with McCulloch-Pitts neurons. Indeed, the approach of Maass et al. to generate neural microcircuits on a virtual three-dimensional lattice which then are mapped onto the HAGEN hardware has been prepared in software (see Sec. 4.4.3) but not systematically investigated in this thesis; initial results on liquids generated by this microcircuit approach can be found in [31]. Instead, the simplicity of McCulloch-Pitts neurons allows an analytical approach to the achievable network dynamics. Bertschinger and Natschläger [21] analyzed a special type of randomly connected recurrent neural network with threshold gates, which they call *input driven networks*. Initially, their mean-field theory considered symmetric threshold gates with states $\{-1, 1\}$; recently[7], they extended the analytical description to threshold neurons of the McCulloch-Pitts type (states $\{0, 1\}$) [151, 150].

[7] The experimental adaptation of input driven networks to the McCulloch-Pitts type networks in the *liquid*HAGEN setup [195] overlapped with the extension of the mean-field theory [151].

The construction procedure of an input-driven network is fairly straight forward: Consider a pool of N neurons where each neuron is connected to the input $u(t)$ and therefore it is *input driven*. Furthermore, each neuron has k non-zero incoming connections from other neurons in the pool. The weights of these connections are drawn from a zero-centered Gaussian distribution with variance σ^2. The liquid state $x(t)$ at some time t therefore can be written as:

$$x_i(t) = \Theta\left(\sum_{j=1}^{N} \omega_{ij} x_j(t-1) + u(t)\right), \quad \forall i \in \{1,\ldots,N\}. \tag{2.5}$$

Here, Θ is the Heaviside step function and the number of neurons N is equivalent to the earlier introduced dimension m of the liquid state. For the theoretical analysis in [21], the input signal $u(t)$ is considered to be a time-discrete signal which at each time step is given by

$$u(t) = \overline{u} + \begin{cases} 1 & \text{with probability } r \\ -1 & \text{with probability } 1-r \end{cases} \tag{2.6}$$

The quantity $-\overline{u}$ can be considered to be the common threshold of all neurons.

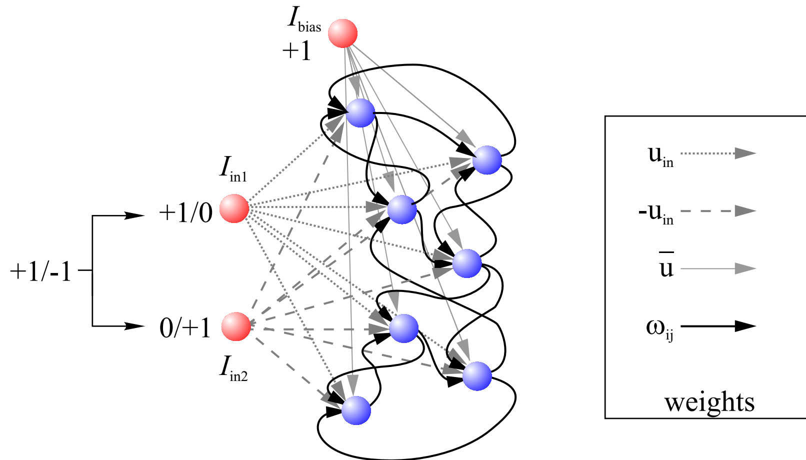


Figure 2.4: Illustration of an input driven network as adopted for the *liquid*HAGEN setup for the toy case $N = 7$, $k = 2$.

This procedure is mapped to the *liquid*HAGEN setup in the following way: The 11 bit weights of the HAGEN prototype are normalized to the interval $[-1,1]$; if a weight is drawn from the Gaussian distribution that exceeds these limits, it is clipped. The input signal $u(t)$ can be fed into the network with the help of three inputs: The $\overline{u}$-contribution is given by a constantly activated bias neuron connecting all N neurons with equal weight $\overline{u}$. The ± 1-signal is fed in by two mutually active neurons connecting all N neurons with weight u_{in} and $-u_{\text{in}}$ respectively

$$u(t) = I_{\text{bias}}\overline{u} + I_{\text{in1}} u_{\text{in}} + I_{\text{in2}}(-u_{\text{in}}), \tag{2.7}$$

where $I_{\text{bias}} = 1$, $I_{\text{in1}}, I_{\text{in2}} \in \{0,1\}$ and $I_{\text{in1}} \neq I_{\text{in2}}$, and $\overline{u} \in [-1,1]$ and $u_{\text{in}} \in [0,1]$. This is illustrated in Fig. 2.4. The here implemented input driven networks are thus a mixture[8] between the ones

[8] The experimental setup allows the use of only a single neuron for the input in order to comply with the analytical description in [151], i.e. I_{in2} always set to 0. Yet, in the *liquid*HAGEN setup the adopted implementation yields better results as will be shown in Sec. 5.2.2.

with $\{-1,1\}$-neurons and ± 1 input described in [21] and the $\{0,1\}$-networks with $\{0,1\}$-input described in [151, 150].

As mentioned in Sec. 2.2.2, for a given neuron not all N other neurons are potential connection candidates. Therefore, the k incoming connections are determined from the physically available connections, i.e., all neurons of the same block plus a predetermined number of neurons from other blocks (c.f. Sec. 3.2 and in Fig. B.1 of the appendix).

2.3.2 The Edge of Chaos

While it can be intuitively understood that input-dominated dynamics exhibit fading memory and chaotic dynamics the separation property, it is not immediately clear for what dynamics a network offers the best kernel-properties. The advantage of considering the simple input driven networks as liquids is the small number of parameters which control the network dynamics. For this type of networks Bertschinger et al. [21] were able to predict a phase transition from ordered to chaotic dynamics analytically and showed that in the vicinity of this transition the kernel qualities of the liquid are best.

Their definition of the ordered and chaotic phase of a discrete dynamical systems adopts the one introduced earlier by Derrida and Pomeau [45]: One considers two (initial) network states of the same liquid which have a certain normalized Hamming distance:

$$d_{ham}(x(t),\hat{x}(t)) := \frac{1}{N}\sum_{i=1}^{N}|x_i(t)-\hat{x}_i(t)|, \tag{2.8}$$

with N being the number of threshold neurons, and $x_i(t)$, $\hat{x}_i(t)$ the components of the liquid state vectors. Receiving the same input, these states map to their next successive states according to the dynamics of Eq. 2.4. If the Hamming distance of these new states tends to increase, this is said to be a sign of chaos. Conversely, a decrease in Hamming distance is a sign of order. This definition of chaos emphasizes the sensitivity to differences in the initial conditions in analogy to the definition of chaos in continuous systems with the help of the Lyapunov exponent [207].

The mean-field theory[9] for $\{0,1\}$-networks of Bertschinger and Natschläger [151, 150] considers the limit of $N \to \infty$ and assumes that all weights are randomly re-generated in each time step. This allows to treat the network activity at some time t as the probability of the network being in that state which is only dependent on the previous state and the input. They derive an analytical update rule for the network activity as well as for the Hamming distance of two network states given a certain input. The Hamming distance at some time t and the expected distance at time $t+1$ form a map which can be analyzed for stable fixed-points. If the only stable fixed-point is given for the state distance being zero, the network is in ordered phase. They eventually derive the following condition for ordered network dynamics:

$$\mathrm{P_{bf}} \leq \frac{1}{k}, \tag{2.9}$$

where $\mathrm{P_{bf}}$ denotes the averaged (over the inputs and the network activity) probability that a single neuron will change its output upon a single bit flip in its k inputs. In Sec. A.2 of the appendix it is shown in more detail how this probability is derived by summing over the appropriate individual bit flip probabilities.

For the discussion here a more qualitatively interpretation is sufficient: As long as the dependency of a neuron's output from its k incoming connections is sufficiently small compared to its

[9] Essentially, the mean-field theory of Derrida and Pomeau [45] for random networks of automata is adopted and extended with a driving input.

dependency on the input u it is likely to follow the input and therefore will exhibit ordered dynamics. Consequently, if for a given k the variance σ^2 is increased—which causes larger weights for the k incoming connections and thus a larger impact compared to the input signal, i.e. P_{bf} grows—the dynamics will move towards the chaotic regime.

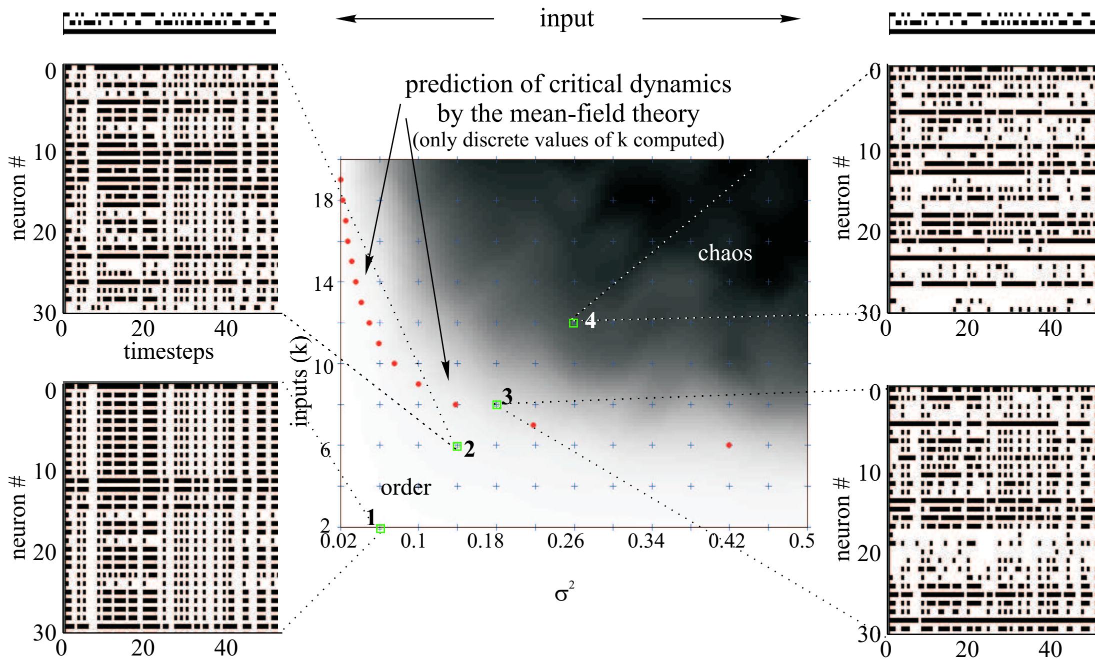


Figure 2.5: The red dots show the theoretical prediction of the k/σ^2-phase transition between ordered and chaotic dynamics according to the mean-field theory of [151]. The exemplary transients of liquid states at the appropriate parameter sets are measured in the *liquid*HAGEN setup. The underlying gray values show the integral Hamming distance for liquid states derived from initially different and eventually identical inputs; see Sec. 2.3.3 and Eq. 2.18. Dark values encode non-decaying separation and indicate chaotic dynamics; actual measurements are indicated with crosses, intermediate values are interpolated. The comparison between the theoretical prediction and the observations in the experiment is discussed in more detail in Sec. 5.2.2.

Despite the limit assumption of $N \to \infty$ and the simplification to a weight regeneration at each time step, the mean-field theory accurately predicts the time development of the Hamming state distances in simulated finitely-sized networks ($\sim 10^2$ neurons) with fixed weights drawn prior to the simulation [21, 151]. Fig. 2.5 shows a phase space diagram for the number of inputs k and the variance of the weight distribution σ^2. The number of inputs k to an individual neuron is discrete, thus the theoretical prediction by the mean-field theory is only computed for integer k and marked by large (red) dots; the values are derived by adopting the Mathematica notebook provided by Natschläger and Bertschinger [149] to match the type of input driven networks used in the *liquid*HAGEN setup. There, a search for σ^2 is implemented that yields equality in Eq. 2.9 for each k and fixing $\overline{u} = 0$; more details are given in Sec. A.2 of the appendix.

The underlying gray values visualize a separation measure which is defined in Sec. 2.3.3; a darker color implies a larger separation if integrating over the time development of the Hamming distance for liquid states derived from initially different and eventually identical inputs, thus yielding a measure for chaotic dynamics. The values are derived from measurements (indicated by blue pluses) in the *liquid*HAGEN setup and recover the predicted transition (note that the color code

is in arbitrary units and does not yield the critical line). Intermediate values are interpolated and may be interpreted as the average number of connections in a network where some neurons have more connections than others.

In order to illustrate the dynamics, in Fig. 2.5 furthermore the transient behavior of liquid states for four liquids generated at the selected parameter sets is shown[10]; the respective data was gathered in the *liquid*HAGEN setup. The hyperbola-like transition in the k/σ^2-diagram is the expected shape considering Eq. 2.9 and that P_{bf} grows with σ^2 as discussed above.

It can be seen that fewer inputs per neurons (small k) or smaller weights (small σ^2) cause the liquid state to follow primarily the input and thus yield ordered dynamics. Respectively, larger numbers of inputs and/or larger weights cause dynamics which only show spurious dependence on the input and are—according to the used definition—chaotic. In terms of fading memory, the vicinity of the phase transition, i.e. so-called *critical dynamics*, provides the best compromise of a memory that is not infinite. It is also clear that the separation property of the liquid increases towards the chaotic regime. Yet, in the chaotic regime, already single bit differences that occurred much earlier in the input or in the initial states already cause different trajectories. It is thus not immediately clear, whether a readout can successfully extract information about more recent input differences. Taking this into account, Bertschinger et al. [21] used the mean-field theory to formulate a measure that aims to predicts the capability of the liquid to separate differences caused by different time courses of the input signals. They were able to show that it peaks along the phase transition from order to chaos.

So far the liquid dynamics have been considered directly. But ultimately, the kernel-properties of the liquid have to be favorably usable by the readout. As it is shown in [21] for simulations and in Sec. 5.2.2 for a liquid implemented in hardware, the capability of a linear readout to predict outputs that are non-linear functions of the input indeed coincides with the predicted critical dynamics. Along with the non-linear separation, the capacity of the readout to extract information from inputs further back in time peaks. For these reasons Bertschinger et al. [21] adopted the term “computation at the edge of chaos” for the observed behavior. This term has been introduced in the context of cellular automata by Langton [118] and Packard [158], where it was similarly observed that in a complex system critical dynamics are favorable for the computation in terms of transmission, storage, and modification of information. On the other hand, not necessarily all types of liquids show exactly this behavior: As is was noted by Maass et al. [129], the computational power of liquid state machines built from spiking neurons tend to peak in a region where chaos has already set in.

2.3.3 Performance Measures

In the course of the liquid experiments of Sec. 5.1 several performance measures will be used. On the one hand, it is necessary to assess the capabilities of the readout. For the correlation of the desired target output with the outcome predicted by the liquid state machine the *mutual information* as introduced by Shannon [198] is used. This can be used to define the *memory capacity* following Jaeger [105]. On the other hand, it is desirable to analyze the dynamics of the liquid directly. In order to do this, the separation of two sufficiently different input streams will be measured.

Mutual Information In the context of transporting a signal across a discrete communications channel Shannon developed his landmark theory of communication [198]. Underlying the theory

[10]Only a 30 bit subset of the 256 bit liquid states for 50 time steps is shown.

is the entropy assigned to a random variable which encodes how much information can on average be gained when the value of this variable becomes known. Equivalently, the entropy can be interpreted as the uncertainty about the random variable before its value is known. The Shannon entropy of a random variable X is defined as:

$$H(X) := -\sum_{x'} p(x') \log_2 p(x'), \tag{2.10}$$

where $p(x')$ is the probability distribution of the random variable X with the index set x'. Furthermore the definition $\log_2 0 := 0$ is implied. When measured using a basis of two, the natural unit for entropy is *bits*.

If there are two discrete random variables X and Y, one can use the entropy definition from above to define the mutual information content of these two variables given by the *mutual information*:

$$MI(X,Y) := H(X:Y) := H(X) + H(Y) - H(X,Y). \tag{2.11}$$

Here, $H(X,Y)$ is the joint entropy given by Eq. 2.10 and summing over the index sets x' and y' for the joint probability distribution $p(x',y')$. Substituting the entropy definition into Eq. 2.11 yields:

$$MI(X,Y) = \sum_{x'} \sum_{y'} p(x',y') \log_2 \frac{p(x',y')}{p(x')p(y')}. \tag{2.12}$$

From Eq. 2.12 it can immediately be seen that the mutual information for two uncorrelated random variables with $p(x',y') = p(x')p(y')$ is zero and thus shows that there is no mutual information between X and Y. For the case $X = Y$ the mutual information is unity.

In a liquid computing setup the mutual information is a suitable measure to assess the correlation between the desired target output y and the predicted output v by the LSM. In liquids of the input driven type with binary input and output used here, the probabilities in the definition of the mutual information between the desired target output y and the predicted output v, $MI(v,y)$, reliably can be estimated by counting the correctly and falsely classified bits for each of the two states $\{0,1\}$. Accordingly, the sums in Eq. 2.12 run over the index sets $v',y' \in \{0,1\}$, $p(v',y') = \Pr\{v(t) = v' \text{ and } y(t) = y'\}$ is the joint probability, and $p(v') = \Pr\{v(t) = v'\}$, $p(y') = \Pr\{y(t) = y'\}$ are the marginal distributions.

Unless explicitely noted, the complete time course is accounted for when computing the mutual information. For the degradation experiments the mutual information is respectively computed for small windows in time:

$$MI_{t_0}^T(v,y) = \sum_{x' \in \{0,1\}} \sum_{y' \in \{0,1\}} p_{t_0}^T(v',y') \log_2 \frac{p_{t_0}^T(v',y')}{p_{t_0}^T(v')p_{t_0}^T(y')}, \tag{2.13}$$

with $p_{t_0}^T(v') := \Pr\{v(t) = v' \text{ for } t \in [t_0, t_0+T]\}$ being the marginal probability determined from a finite window in time. $p_{t_0}^T(y')$ and $p_{t_0}^T(v',y')$ defined accordingly. For $t_0 = 0$ and T being the total length of the time series v, y, the regular mutual information is recovered.

Memory Capacity The mutual information $MI(v,y)$ correlates the prediction of the LSM and the target outcome of any given filter F and input, i.e. $y(t) = (Fu)(t)$. In [153, 21] it has been proposed to consider the same filter yet increasingly shifted in time and analyze the respective mutual information between the predictions and the target.

Let for example the desired target filter be the computation of the parity of the last 3 steps in time:

$$y(t) = (Fu)(t) = \text{PARITY}(u(t), u(t-1), u(t-2)). \tag{2.14}$$

PSfrag replacements

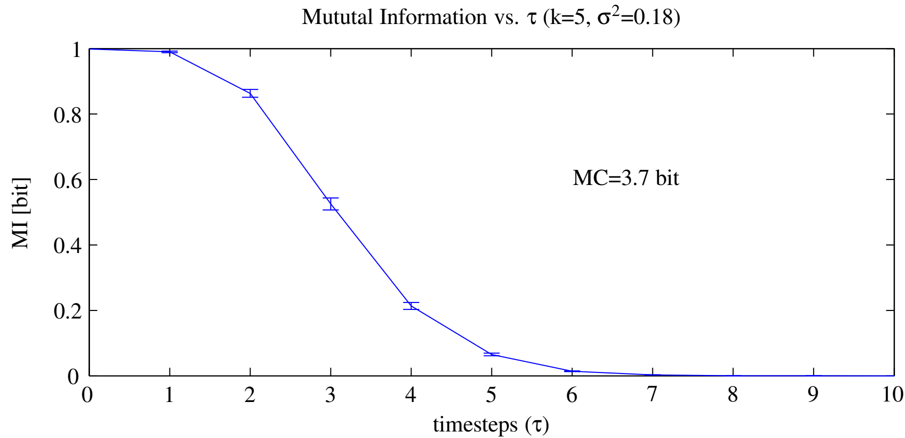


Figure 2.6: Measured mutual information between the predicted and the target output for the 3-bit time-delayed parity task. Due to the discrete input, the connecting lines are merely a guidance for the eye and no fit. The term *memory curve* for this plot resembles the fact that for continuous inputs intermediate values can be measured and the memory capacity is the integral of the curve. In the discrete case the MC is the sum of the measured MIs. At each τ 30 liquids are evaluated and plotted is the mean MI; the limits indicate the standard deviation of the mean. The parameters for the LSM implemented in the *liquid*HAGEN setup were $N = 256$, $k = 5$, $\sigma^2 = 0.18$, and $\overline{u} = 0$. The full experiment is shown in Fig. 5.4.

The shifted target filter then is given by:

$$y_\tau(t) := \text{PARITY}(u(t-\tau), u(t-\tau-1), u(t-\tau-2)), \tag{2.15}$$

for increasing integer delays $\tau \geq 0$. Together, these filters are called the 3-bit time-delayed parity task.

For each value $\tau \geq 0$ a separate readout gets trained. A good performance of the resulting LSM on this task coincides with high values of the mutual informations $MI(v_\tau, y_\tau)$ for increasing τ. In order to achieve this at each time t there has to be information present in the liquid state $x(t)$ of more distant input. Again, this input needs to be non-linearly transformed in order for the linear readout to extract the (delayed) parity. In Fig. 2.6 the experimentally measured mutual information for the 3-bit time-delayed parity task versus increasing τ, the memory curve, is plotted; details of the experiment will be discussed in Sec. 5.2.2. As can be seen, the task of predicting the parity gets increasingly more difficult with increasing τ: The mutual information eventually decays to zero.

Following Jaeger [105], the area under the memory curve can be defined as the *memory capacity* of the liquid. In the case of discrete input, the memory capacity is the sum of the measured values of MI for each τ, measured in bits:

$$MC = \sum_{\tau=0}^{\infty} MI(v_\tau, y_\tau). \tag{2.16}$$

Separation The performance measures presented so far characterize the ability of the readout to extract non-linear information about the input present in the liquid. While this is closely related to the dynamics of the liquid as discussed in Sec. 2.3.2 and shown in Sec. 5.2.2, it is not a direct measure for these dynamics. A simple way to distinguish ordered dynamics from critical ones experimentally is to directly measure the time course of the normalized Hamming distance as

PSfrag replacements

defined by Eq. 2.8 for two network states $x(t)$, $\hat{x}(t)$ arising in the same liquid if driven subsequently by inputs $u(t)$, $\hat{u}(t)$ that initially differ and eventually are identical:

$$u(t) \neq \hat{u}(t) \text{ for } t < t_0 \text{ and } u(t) = \hat{u}(t) \text{ for } t > t_0. \tag{2.17}$$

According to the definition of chaos given in Sec. 2.4, the dynamics of a liquid are said to be ordered if and only if $\lim_{t\to\infty} d_{ham}(x(t), \hat{x}(t)) = 0$ given that the input can be described by Eq. 2.17. Similarly, for asymptotically non-zero values the dynamics are said to be chaotic.

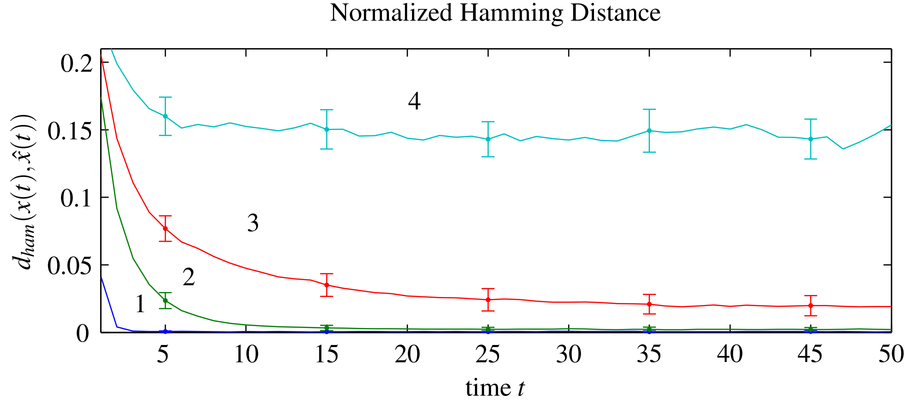


Figure 2.7: Measured transient Hamming distances between network states arising from initially different and eventually identical inputs (the identical input here starts at $t = 0$). If the distances asymptotically go to zero, the network dynamics are ordered; otherwise, the dynamics are said to exhibit chaos. The numbers 1-4 correspond with the appropriate parameter sets shown in Fig. 2.5.

Fig. 2.7 shows the measured hamming distances for liquid states of liquids drawn from different parameter settings k, σ^2 and evaluated in the *liquid*HAGEN setup. The given numbers correspond to the appropriate parameter sets indicated in Fig. 2.5. For each parameter set several liquids were generated and each tested for several pairs of inputs obeying Eq. 2.17. Plotted is the averaged separation (across the input series and liquids) of the liquid states starting at time t_0 at which the inputs become identical. The transients of the distances show that k, σ^2 pairs have been chosen that cause ordered, critical, and chaotic dynamics.

A simple measure expressing the separation property of the network therefore is the area enclosed by these transients and a finite time T that is sufficiently[11] large to cover the asymptotic behavior (the absolute value of the integral is thus arbitrary!):

$$D_T(x(t), \hat{x}(t)) := \sum_{t=t_0}^{T} d_{ham}(x(t), \hat{x}(t)). \tag{2.18}$$

A visualization of this measure can be found in Fig. 2.5. There it is computed for a measured parameter sweep in the *liquid*HAGEN setup. It can be seen that the mean-field theory quite accurately predicts the onset of chaos. This will be discussed in detail in Sec. 5.2.2.

[11] For the liquids examined in Ch. 5, $T = 50$ has been chosen as this value is about 10 times larger than the observed memory capacity.

Chapter 3

The HAGEN Prototype Implementation

The central aspect of the neural network experiment is that the concepts introduced in Ch. 1 are realized in dedicated hardware in order to allow a physical test. In this chapter the implementation details of the prototype ANN ASIC based on current-sum neurons—the HAGEN prototype—are illustrated with references to the predecessor ASIC based on charge-sharing. Special emphasis is put on the neuron and synapse circuits which are the basis of the concept. Furthermore, the operation of the HAGEN prototype in recall and programming phase along with the necessary supplementary circuits are described. Finally, strategies are elucidated to use the proposed concepts beyond a prototype implementation.

3.1 Learning from the Predecessor ASIC

As it was described in Sec. 1.3.2, the neural network concepts based on modular network blocks encapsulating the analog network evaluation by digital I/O have been realized with two prototype ASICs. The first, the EvoOpt prototype, implements a single network block of 64 neurons based on the charge-sharing principle introduced in Sec. 1.2.3. Its complete design is described in [186].

As illustrated in Fig. 3.1, the synapses for the charge-sharing evaluation principle require essentially two capacitances in the synapses. One for the storage of the synapse's weight and one that actually participates in the charge-sharing with all other active synapses of an output neuron. These capacitances are chosen to be 60 fF and 100 fF respectively, which allows the realization of the synapse circuit in only 10 by 10 μm^2 of silicon area. 64 of these synapses have been connected to each of the 64 output neurons which yields a synapse array of 0.7 by 0.7 mm^2. The network core has been simulated for a network frequency of up to 100 MHz. This yields a nominal number of connections per second (CPS)—a common measure for the speed in recall phase of a neural network [90]—of 0.4 Gigaconnections/s.

The left hand side of Fig. 3.2 shows a microphotograph of the prototype ASIC with the dimensions 2.5 by 2.5 mm^2. It can be seen that the design is pad-limited; this primarily arises from the fact that it is quite challenging for a simplistic interface based on single-ended data transmission to provide the necessary data rates, i.e., 64 bits for the input and 64 bits for the output in each network cycle. Therefore, a 16-bit wide interface has been implemented which allows the operation

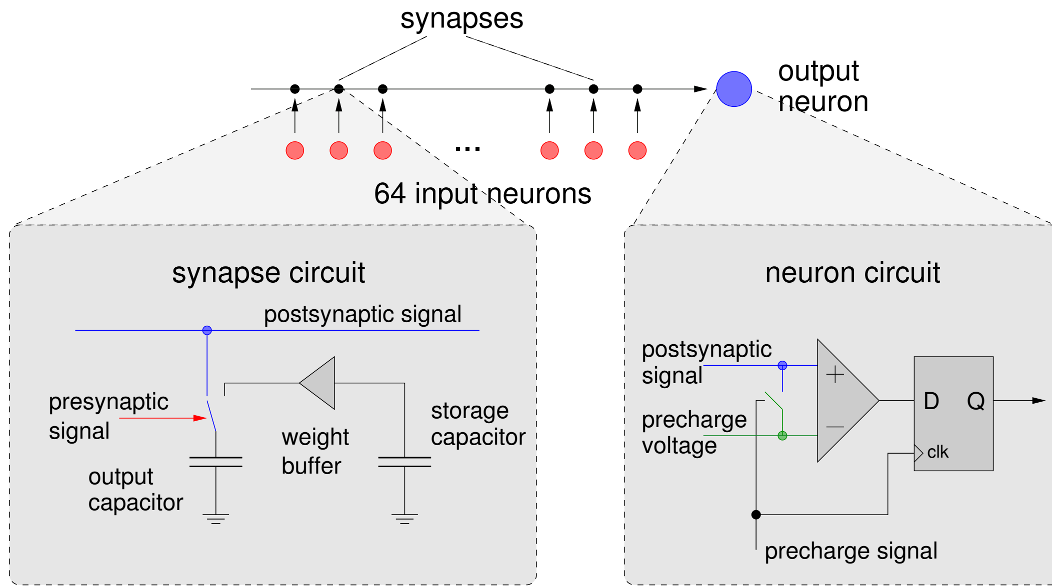


Figure 3.1: Operation principle of the charge-sharing neuron implemented in the EvoOpt ASIC. Figure taken from [186].

of the ASIC at a network frequency of 50 MHz. Consequently, the maximum CPS value is half the nominal value in practice. An overview of the specifications was given in Tab. 1.2 of Sec. 1.3.2.

The right hand side of Fig. 3.2 shows a schematic diagram of the core. It can be seen that indeed the matrix structure of Sec. 1.2 has been realized, but the inputs are fed from two sides into the array. The third side is used for the output neurons and via the last side the weight storage is performed. The weight generation is not integrated to the ASIC itself, rather, the appropriate weight voltages as well as the reference voltage have to be provided by external DACs. While the reference voltage is static, the weight voltage varies from synapse to synapse. A weight storage unit is used to buffer one voltage value per output neuron. This allows to have a fast DAC sequentially generating the voltage values which then get transferred row-wise into the synapse array. In the setup of Ch. 4 a complete weight update can be performed in 250 μs. According to a common measure for the update rate in a neural network, the connection updates per second (CUPS) [90], the ASIC yields 16 Megaweights/s.

Systematic experiments on the 4-bit parity problem [88] as well as the implementation of 3-bit variable network resources [193] were performed successfully with the help of an evolutionary algorithm that employs a hardware-in-the-loop evaluation (see Sec. 4.4.1). Such a setup allows to cope with all kinds of inaccuracies, especially when they are systematic. From the experiments with the EvoOpt prototype several conclusions have been drawn:

1. Multi-layer networks can be successfully realized with the network block structure.
2. The employed process technology allows reasonably sized capacitive weight storage which only needs to be refreshed every 10 to 100 ms.
3. The achievable accuracy of the utilized charge-sharing implementation is about 4-6 bits.

This limitation of the accuracy primarily arises from external sources: First of all, the analog operation is in a quite noisy environment due to the single-ended external interface. Similarly, the reference voltage which is used by the comparators in the output neurons is provided externally.

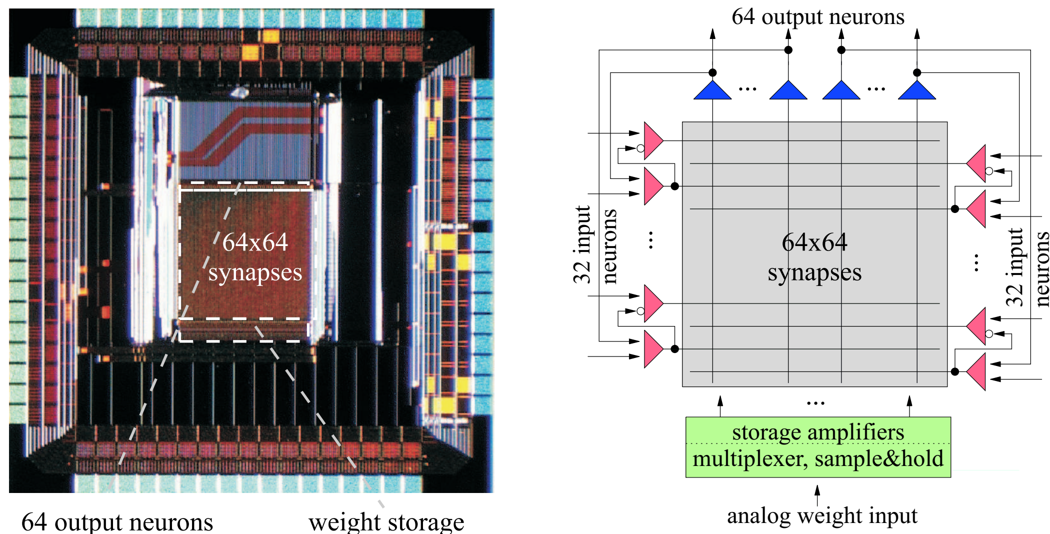


Figure 3.2: Left: Micro photograph of the EvoOpt prototype. **Right:** Schematic diagram of the EvoOpt. Figure adopted from [186].

Since this voltage is also used to precharge the output capacitances in the synapses, it is especially susceptible for timely variations which can be caused by crosstalk. This effect is worsened by the internal conception of the ASIC described in Sec. 1.2.3: The potential increase/decrease caused by the same synapse is dependent on the number of simultaneously active synapses.

The key goal of the HAGEN prototype was to take on the successful block concept of the predecessor ASIC but increase the analog accuracy. Using the current-sum principle of Sec. 1.2.3 allows a number of improvements, which further are combined with a more elaborate interface and weight generation:

- Relinquishing the high voltage swings of the single-ended interface and replacing it by *low voltage differential signalling* (LVDS) reduces the digital noise across the entire ASIC; details are given in Sec. 3.5.

- The differential representation of the postsynaptic activity described in Sec. 1.2.3 abandons the need for an absolute reference voltage during thresholding.

- The weights are represented by currents. Therefore, current memories in each synapse are used to store the desired weight current. By programming those memories with a current it is possible to automatically compensate for most fixed-pattern mismatches in the synapses, such as effective capacitance, threshold voltages, or transconductivity, and allows to operate the synapse in a larger dynamic range. Additionally, systematic effects such as the process corner or temperature are compensated.

- Using currents to represent the weights has furthermore the advantage that a current-steering approach can be employed. This for example allows to maintain a constant potential at the output node of the weight current memories which improves accuracy and accelerates the operation. Simultaneously, the power consumption is kept constant which provides stable temperatures across the synapse array.

- Lastly, the required currents for programming the synapses are generated by on-chip DACs (Sec. 3.4). On the one hand, this simplifies the handling since only digital signals have to be

exchanged with the network block. In effect, this reduces the susceptibility to external noise and accelerates the operation due to minimized parasitic capacitances. On the other hand, the on-chip DACs are the key to scale the architecture to larger arrays of network blocks.

3.2 HAGEN Overview

In the HAGEN prototype four equally sized network blocks with 64 current-sum neurons having 128 inputs each are realized. Fig. 3.3 shows a micro photograph. In contrast to its predecessor ASIC it incorporates the weight generation on-chip and therefore provides a fully digital external interface. The weights are written with a nominal amplitude resolution of 10 bits and a sign. The design of the network blocks allows to be operated at a frequency of $f_{\text{net}} = 50$ MHz which yields a maximum CPS value of 1.64 Teraconnections/s for the HAGEN prototype. Furthermore, a complete weight refresh of all 32768 synapses can be realized in 82 μs which corresponds to a maximum CUPS value of 400 Megaweights/s. Due to its synapses which only cover a silicon area of 8.7 by 12 μm^2, it has a core size of 3.6×2.5 mm. Its dedicated bi-directional double data rate interface realized by custom-built LVDS transceivers yields a net rate of 11.4 Gbit/s. The performance data has already been given in Tab.1.2 of Sec. 1.3.2.

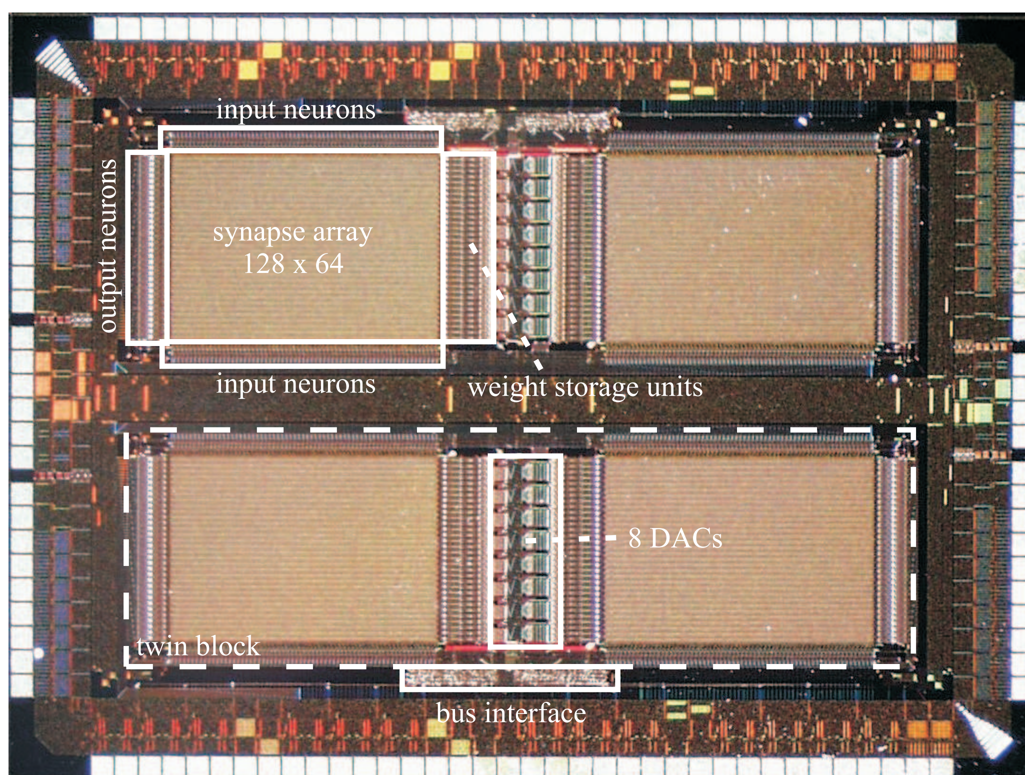


Figure 3.3: Micro photograph of the HAGEN prototype.

The concept and the implementation of the neural network circuitry of the HAGEN prototype originate from Dr. J. Schemmel. Technical details have previously been published in [187] and [184]; here, technical details of selected circuits are discussed and supported by simulations and measurements in order to illustrate the ideas of Ch. 1 and to give a concise description of the operation of the ASIC in the neural network experiment. The full-custom LVDS transceivers, which realize the low-noise and high-speed interface of the HAGEN prototype, were developed

within this thesis work adopting LVDS circuits from a 0.25 μm process [113]; see Sec. 3.5.1 for details.

The HAGEN prototype has been designed and manufactured in the 0.35 μm CSI technology of Austria Mikro Systeme International AG (AMS), which is a complementary metal oxide semiconductor process based on a p-substrate. At the time of manufacturing this 3.3 V process provided 3 metal and 2 poly layers [15] and allowed minimum size transistors of the dimension 0.3 μm by 0.6 μm (length by width) [14]. All metal layers and one poly layer have been utilized for the prototype. It was designed using the AMS HitKit 3.3 in combination with Cadence 4.4.3 using Spectre as the simulator.

Network Block The HAGEN prototype implements the differential current-sum neuron introduced in Sec. 1.2.3. The used adaptation is illustrated in a simplified diagram in Fig. 3.4. There it can be seen that the current memory in each synapse is realized by a voltage controlled current sink. While the weight value is stored as a charge on a 60 fF capacitor, it is programmed by a current which causes the potential on the weight capacitor to assume the appropriate value (see Sec. 3.3.4). The sign of the synapse is stored in a dynamic latch.

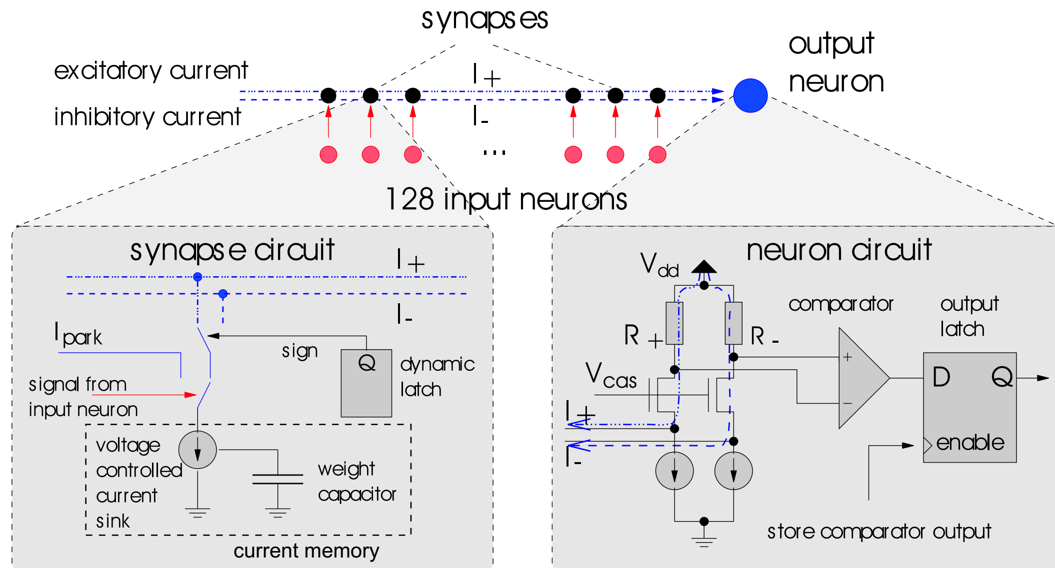


Figure 3.4: Adaptation of the current-sum neuron in the HAGEN prototype. Figure adapted from [187].

The current-to-voltage conversion is realized in each output neuron by transistors operated in the Ohmic region R_+, R_-. The synapses connected to the I_+, I_- branch are isolated from those resistances by the two transistors with their gates connected to V_{cas}. Those cascodes provide a low input impedance to the current sinks in the synapses and essentially maintain a constant potential on the I_+, I_- branch. The current sources in the neuron circuit finally keep the cascodes in saturation. Circuit details of the synapse and neuron will be given in Sec. 3.3.

One network block of the HAGEN prototype realizes 64 of these current-sum neurons having 128 inputs each which totals to 8192 synapses. In addition, the necessary circuitry to program the synapse current memories is as well implemented in each network block.

The block dimensioning is based on the preceeding prototype: Doubling the number of inputs allows not only to fully feedback the block to itself but rather accommodate additional feeds from other blocks or external input. If strictly layered topologies are to be implemented in a single

network block, the weights of unused inputs need to be set to zero. As the power consumption of a synapse is solely given by its programmed weight (c.f. Sec. 3.6), unused synapses only occupy silicon area and have no other negative effect. Yet, if the number of inputs to a block thus is chosen to be much larger than the number of neurons per layer in a 'deep' topology (i.e., many layers), the array-based computing in a single block is not optimal. In those cases it is better to use multiple smaller network blocks, each implementing a single layer.

Network blocks of the dimension 128 by 64 allow to adapt various types of applications [87, 195, 57], or at least the proof of principle, which emphasizes the general-purpose aspect of the hardware. Nevertheless, with a dedicated application in mind the dimensioning may need to be reconsidered. Technical aspects of this are discussed in Sec. 3.7.

Twin Block Two of these network blocks are combined to a *twin block* sharing eight on-chip current DACs. This is possible because of the analog weight storage units which are part of each network block. A schematic diagram of a twin block is shown in Fig. 3.5.

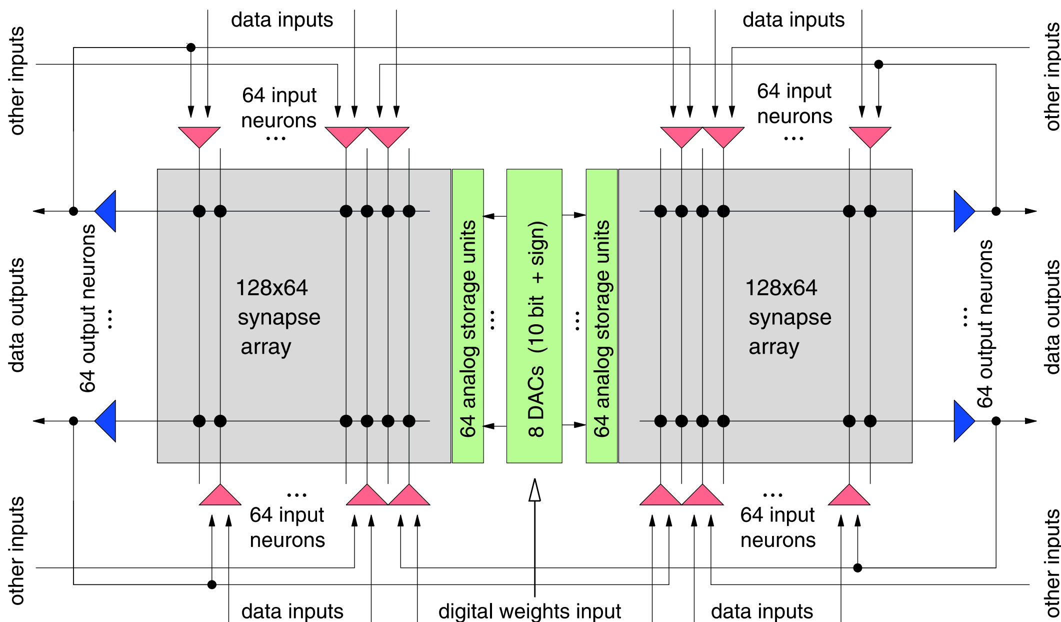


Figure 3.5: Schematic diagram of the upper half of the HAGEN prototype. Shown are two synapse blocks sharing 8 DACs for the weight storage. Figure adapted from [184].

As it will be described in detail in Sec. 3.3.4, these analog weight storage units allow the buffering of a column of weights which at some later point can be simultaneously transferred to the synapse array. By employing this buffer architecture, the generation of the weights can be decoupled from the programming of the synapses. While a column of weights is transferred from the memory cells to the synapse array, the generation of one column's weights can be done in any order. Especially, the degree of parallelism for the digital-to-analog converters can be chosen freely. In order to minimize area consumption, a single DAC would suffice which converts the currents of one column sequentially. To the other end, a DAC for each row would speed up the conversion but would require much more silicon area. Since a network block cannot be used in feed-forward operation while the weights are programmed, it is desirable to minimize the programming time. The compromise that has been chosen for the prototype ASIC is to share eight DACs between two network blocks. Since the DACs are connected to the current memories of

both network blocks, the decoupled weight programming can be optimally utilized: While the DACs are converting the weight values of a column for one network block, the buffered weight column of the other block is transfered to the synapses. With this two-stage design the synaptic weights being programmed are allowed to settle while the DACs are converting the next column of weights of the other block, thus allowing a continuous operation of the DACs. Details on the choice of the DAC architecture are given in Sec. 3.4.

Since the twin block provides the digital-to-analog conversion of the weights, such a unit has a fully digital interface.

Half Chip In contrast to the conceptual building blocks *network block* and *twin block* the now presented entities are of technical nature. Primarily, they are mentioned to document the HAGEN prototype and understand its external interface. The term *half chip* applies to a twin block with all digital and analog circuitry needed to be operated, including a digital bus interface built of standard cells, a bi-directional double data rate LVDS input/output pads, and power pads.

A half chip is a fully functional entity and could have been manufactured[1]. It provides routing for the fixed feedback connections for each of its two network blocks and inter-block connections. Yet, the floor plan of the layout anticipates its use in conjunction with another half chip (mirrored, but otherwise identical) to form a complete HAGEN prototype; accordingly, the two halves are labeled *upper* and *lower*[2].

If the presented concepts are to be scaled towards a larger neural network ASIC, it is likely that slightly modified twin blocks get reused while the remaining circuitry of the half chip will entirely be replaced.

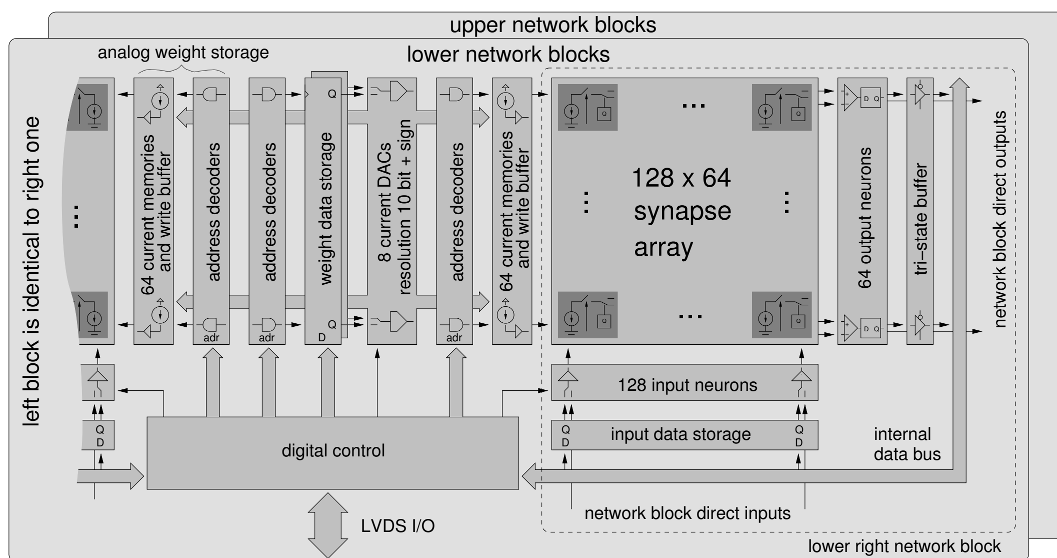


Figure 3.6: Block diagram of the HAGEN prototype. Figure taken from [187].

Full Chip The term *full chip* denotes a complete HAGEN prototype. It consists of two logically independent half chips. Electrically, all digital circuits and I/O pads are independent as well, only

[1]Analog input pads for the three reference voltages, one reference current, and power pads for the analog core would have needed to be added.

[2]The mirroring of the two halves and shared pads prevent the die from being rotational invariant, i.e., the upper and lower half are unambiguously identifiable.

the analog power supply, a few dedicated direct CMOS data pads, and reference signals for the analog circuits are shared between the two half chips. As a consequence, the HAGEN prototype is operated by two separate interfaces (see section 3.5). Fig. 3.6 shows the block diagram of the complete HAGEN prototype.

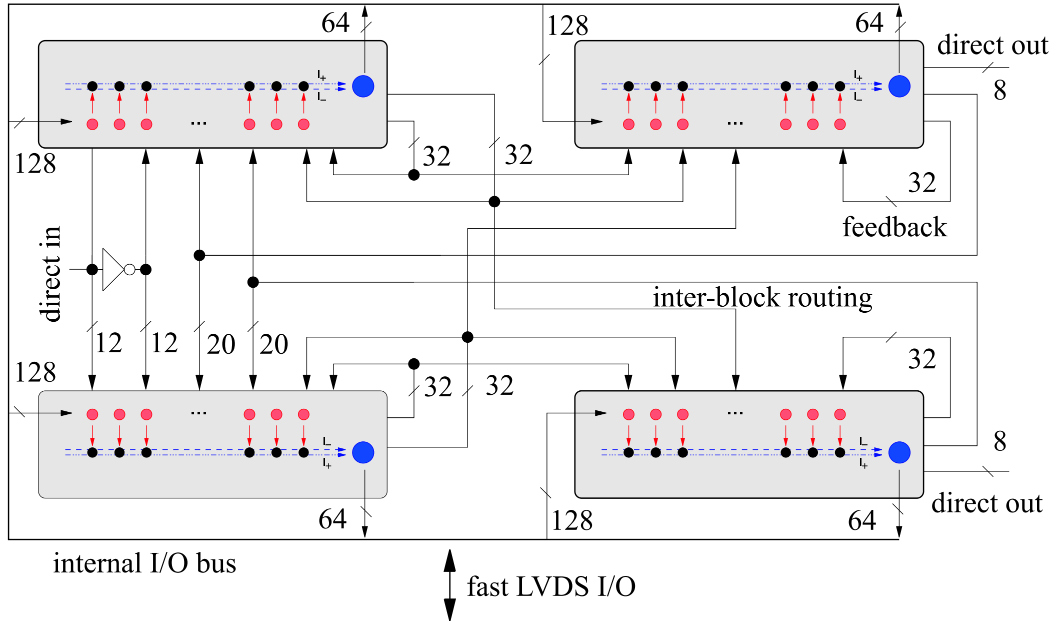


Figure 3.7: Data flow between the four network blocks of the HAGEN prototype. Figure adapted from [187].

In order to allow direct interoperability between the two halves, additional routing has been provided between all four network blocks. Fig. 3.7 shows the implemented intra- and inter-block routing schematically. It is to be noted that the connections are fixed and each input can only be fed by the internal I/O bus or one other connection. An output, on the other hand, can be connected to more than one input. The choice for only one additional connection per input is pragmatic: on the one hand, it allows to feed output signals to the dedicated inputs across the chip readily for the next network cycle. On the other hand, it is straight forward to realize the connections with the limited routing resources of the CMOS process with three metal layers. Because of the symmetry of the weight matrix, the fixed connections are not a restriction within a single network block. Only the maximum number of connections between different blocks is limited. The routing implemented on the HAGEN prototype promotes network architectures that have a predominant data flow from the left network blocks to the right ones: the lower and upper left block for example can each be fully fed back to itself and all of their outputs can be fed to the right blocks (one half of both blocks outputs goes to the upper right block, the other to the lower right block). The backward path from the right blocks to the left allows a maximum of 20 connections each. Basically, the choice was to either ensure a maximum of similarity between the blocks, or to accommodate a larger variety of topologies given the constraints of one additional input source per neuron. A precise documentation on which output connects to which input is given in Fig. B.1 of the appendix.

Especially, data stream processing applications may profit from this architecture. For these kind of application an additional feature has been added: so-called *direct-in* and *direct-out* connections. The direct-in connections are not coming from the outputs of other neurons but originate from CMOS input pads. This makes it possible to directly attach external data sources to the net-

work, e.g. an AD converter. Twelve of these dedicated connections are fed to both left blocks; additionally, the inverse inputs can be presented to the network as well. This allows to have a constant number of inputs active independently of the data; this can be important in networks with the used on/off neurons. Similarly, both right blocks each have eight outputs connected to dedicated CMOS output pads. With this, external data receivers, e.g. DA converters, can be directly connected to the network.

Since all input and output neurons can be configured to connect to the internal data bus, the adopted connection scheme is no limitation for a general connectivity. Via the digital bus interface neurons can get inputs from any source and the output may be distributed anywhere before and after each network cycle. As it will become clear in Sec. 3.5.2 and Sec. 4.2.2, the transfer of 512 input bits and 256 output bits across the external interface requires by construction more time than the execution of a single network cycle. In setups where the full continuous CPS rate is needed, input and output data therefore needs to be fed in and out using the direct I/Os. For these kinds of scenarios the half chip structure of the prototype even allows the required weight refresh cycles to become transparent by having the two halves alternating between programming and recall phase.

3.3 Network Block

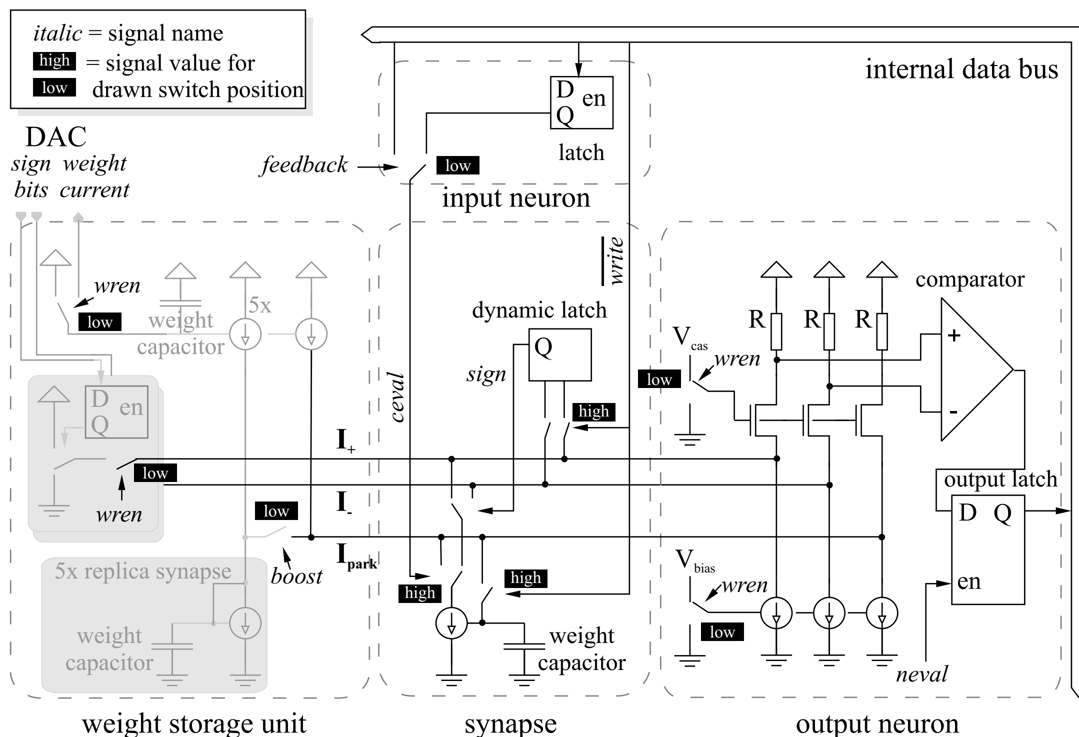


Figure 3.8: Schematic diagram of a network block simplified to one input, one output and one synapse. Shown is the recall phase: the grayed parts are deactivated or disconnected by switches; the switches have assumed position according to the signal stated right next to them.

A detailed diagram of a network block is given in Fig. 3.8. The essential circuits of the network block are shown: the synapse, the output neuron, the input neuron, and the analog weight storage unit. The digital input and output as well as the analog interface to the DAC can be seen. For clarity only one synapse is shown simplifying the diagram to one output and one input. The parts active

in *recall phase* are outlined in black, while parts only active in *programming phase* are outlined in gray. The two phases are differentiated by the *wren* (write enable) signal. In programming phase, the *wren* signal is active which disables the input stage of the output neuron and activates the write buffer of the analog weight storage unit. Details of the weight programming are given in Sec. 3.3.4. In recall phase, the *wren* signal is off, the output neuron gets correctly biased, and most parts of the analog weight storage unit get deactivated.

Independently of the phase, the input neurons are merely switches causing the synapses of the according column to connect their voltage-controlled current sink to one of the postsynaptic signal lines I_+ or I_- or to the park line I_{park}.

3.3.1 A Network Cycle

In recall phase the network block has to evaluate Eq. 1.3. The minimum time Δt needed to do so yields the maximum network frequency f_{net} and is called a *network cycle*. The required time span is comprised of two contributions: First, a correctly signed voltage difference needs to be established at input stage of the output neuron which requires t_{settling}. The settling time is a result of the finite capacitance of the postsynaptic branches: Depending on the activity of the preceeding network cycle surplus charges may need to get dumped by the synapses which takes more time the smaller the synaptic weights are. Second, the resulting voltage difference needs to be amplified to logic levels by the output neuron (t_{compeval}).

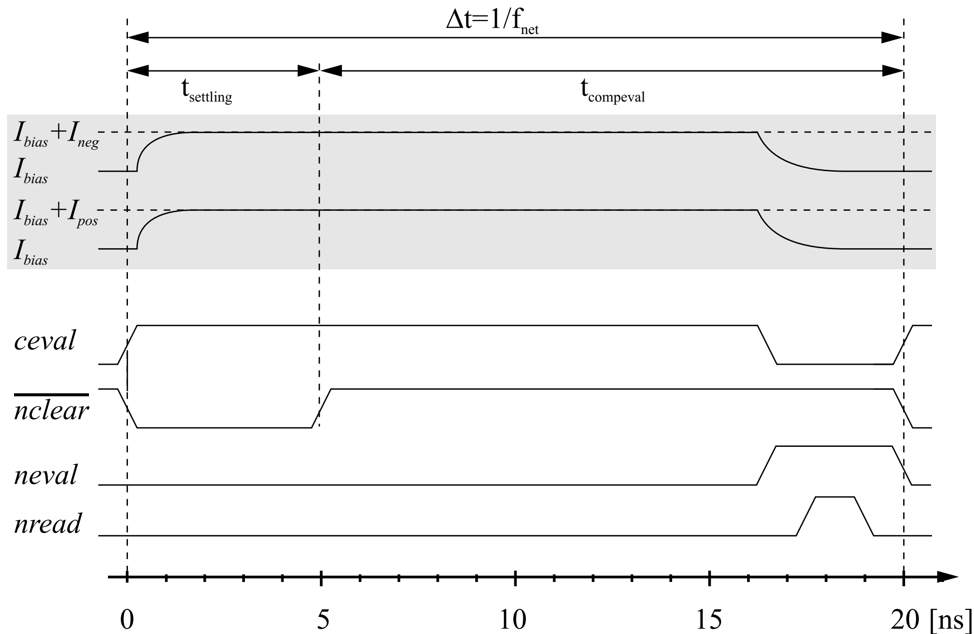


Figure 3.9: Timing diagram of a network cycle. Shown are arbitrary currents on the I_+, I_- branches (gray shaded) and the clock signals used to induce and evaluate this activity. t_{settling} is the time needed to establish a voltage difference in the input stage. t_{compeval} is the time needed by the output neuron to amplify the difference to logic levels. The shown timing allows an f_{net} of 50 Mhz. The clock timing is programmable from the outside of the ASIC.

The diagram of Fig. 3.9 illustrates the timing of a network cycle with 50 MHz. The gray shaded signals represent the postsynaptic signal lines I_+ and I_-. The remaining four signals are

the control signal to activate the synapses (*ceval*)[3] and control the evaluation of the output neuron ($\overline{nclear}$, *neval*, and *nread*). The time course of these four signals is referred to as a *clock pattern*.

The cycle starts with *ceval* which causes the synapses to connect to either I_+ or I_- and sink their earlier programmed current. Simultaneously, $\overline{nclear}$ is set to low to zero the neuron comparator. In the moment $\overline{nclear}$ is released, the comparator will start to amplify the voltage difference present at its input. Essentially, the time span of $\overline{nclear}$ being low constrains the time for establishing an appropriate voltage difference, i.e. the settling time t_{settling}. The *ceval* signal is lowered once the comparator has sufficiently amplified the signal difference in order to prevent a possible interference. The signals *neval* and *nread* finally control the output stages of the output neuron and are explained in detail in Sec. 3.3.3.

To be able to modify the clock pattern, its generation is shifted off-chip: the control signals are transferred in time across the bus interface; this is described in Sec. 3.5. Actual clock patterns for the use in the prototype setup which as well allow a reset of the neurons are given in Sec. B.4 of the appendix.

3.3.2 Elementary Synapse

The synapse essentially is a programmable current memory with the current being proportional to its weight. In Fig. 3.10 the circuit diagram is shown. The circuit is realized by 12 transistors, 5 of which (S4-S6 and S11-S12) are switches needed for the programmability. It was abstained from implementing these switches by transmission gates as to keep the circuit as simple and small as possible; the area occupied by the synapse is only 104.4 μm^2.

The current memory is implemented by S1-S3: S1 is the current sink and the charge on S3 regulates the current. To keep the capacitance of S3 constant, the gate voltage needs to be above the threshold voltage; therefore the diode S2 is introduced having the same W/L ratio as S1. The simultaneously raised output resistance of the synapse is desirable.

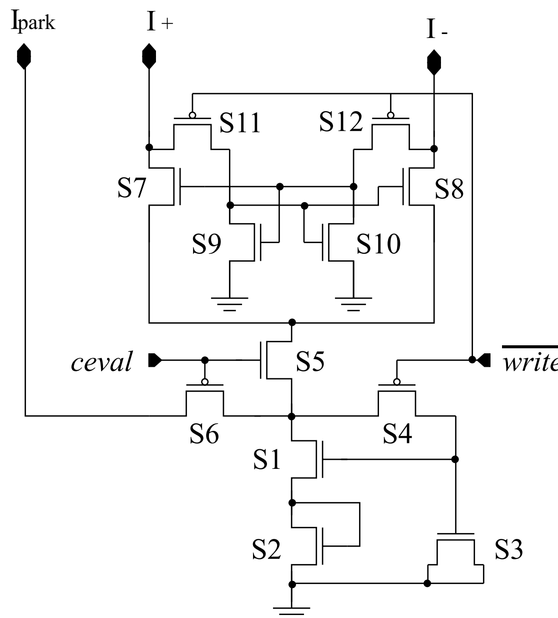


Figure 3.10: Circuit diagram of the synapse. Figure adapted from [184].

[3] *ceval* stands for *column evaluation* and resembles the fact that an input neuron activates all synapses of one column at once.

The *ceval* signal activates and deactivates the synapse with the help of the switches S5 and S6. Depending on their state the current sunk by the synapse is either drawn from I_+/I_- or from I_{park}, i.e., the current is steered. In recall phase *ceval* becomes high each time the input neuron is on. Then the current sink will be connected to either one of the postsynaptic lines I_+ or I_- by S7 or S8. The sign of the synapse is determined by the cross-coupled S9 and S10. When the input neuron is not active, *ceval* is low and the current sink gets connected to the I_{park} line. The I_{park} branch is designed to closely match I_+ and I_- in order to maintain the operating conditions of the current sink as good as possible.

Fig. 3.11 depicts the current-voltage characteristic of the synapse. The plot shows the current sunk by the synapse versus the potential on S3. It is obtained by a DC-analysis using a typical-mean transistor model, having set *ceval* and $\overline{write}$ to high, and providing a realistic load to the postsynaptic branches, i.e., the input stage of the neuron. The swept potential of S3 is provided by an ideal voltage source.

PSfrag replacements

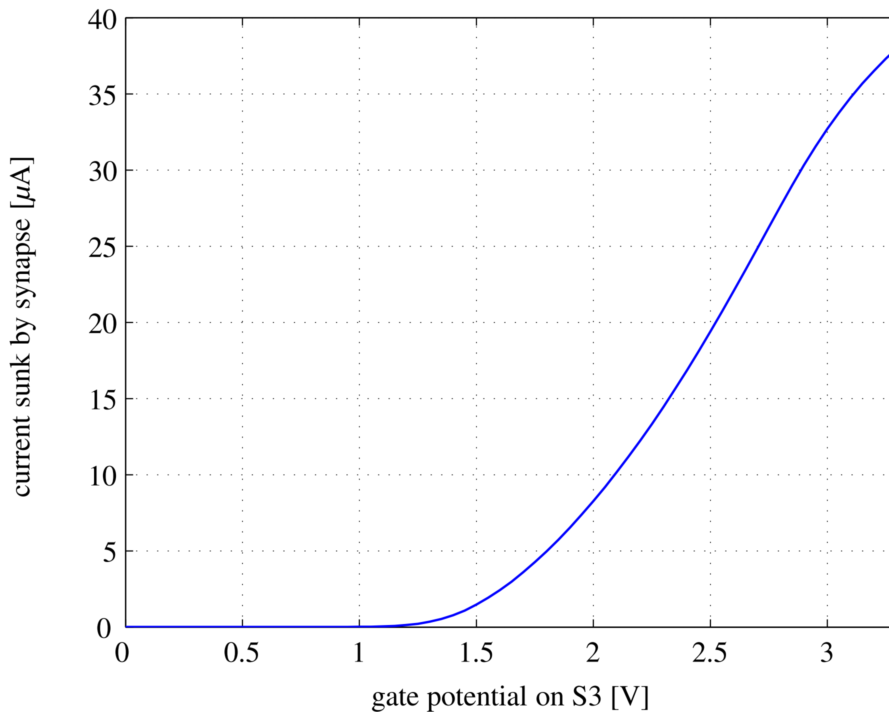


Figure 3.11: Current-voltage characteristic of the synapse. Plotted is the current sunk by the synapse versus the potential U_{gate} on S3. The data is obtained from a DC simulation of a typical mean model with a realistic load, i.e., the input stage of the neuron is attached to the postsynaptic lines I_- and I_+. The potential at the drain of S7 (S8) is lowered by about 80 mV from a U_{gate} = 1 V to a U_{gate} = 3.3 V.

As can be seen, the usable voltage range starts at about 1 V, which is twice the threshold voltage and due to the introduced diode S2. To the other extend, the gate cannot be charged to the full rail voltage of 3.3 V. The reason for this is the threshold voltage of the current source W3 of the weight storage unit (c.f. Sec. 3.3.4 and Fig. 3.17). The remaining maximum usable voltage range therefore is about 1.5 V. The actual current sunk by the synapse is limited by the sizing of S1, S2 (W/L = 1/1) and strongly dependent on the process parameters. A Monte-Carlo analysis

of the expected variations of the transistor parameters yields differences of 25% in the maximum current[4].

Since the voltage-controlled current sink is operated as a current memory, i.e., it is programmed by a current which forces S3 to assume the correct potential (via the switch S4), the non-linear current-voltage characteristic is only of interest for analyzing the effect of noise: While a 30 nA weight difference for large weights is caused by about 1 mV potential difference on the gate, the same weight difference for very small weights is caused by a potential difference of about 30 mV. The actual LSB current increments are determined by the external reference current for the DAC (see Sec. 3.4). The programming is accomplished by the switch S4 which connects the gate of S3 to the drain of S1. Details are described in Sec. 3.3.4.

The theoretical limitation to the accuracy of the here employed capacitive weight storage is ultimately determined by kTC-noise according to thermodynamics [127]. In thermal equilibrium with a reservoir at temperature T, the mean square fluctuation of the energy stored on a capacitor is:

$$\frac{C\overline{\Delta u_c^2}}{2} = \frac{kT}{2}. \tag{3.1}$$

Therefore, the voltage on the capacitor fluctuates with:

$$\overline{\Delta u_c^2} = \frac{kT}{C}. \tag{3.2}$$

Given the dynamic range of the gate voltage of 1.5 V and choosing the extracted capacitance of S3 to be about 60 fF results in a noise at room temperature of about 0.26 mV. This is about a quarter of the minimum 1 mV potential difference for a 30 nA difference in weight and therefore not the limiting factor for an accuracy of 10 bits. Limiting the on-chip DACs to this nominal 10 bit resolution in amplitude has primarily practical reasons:

- Ever smaller currents will require longer times to sink all excess charges on I_+ or I_- after a preceeding network cycle of stronger activity; responsible for this is the capacitance of I_+ and I_- to which even inactive synapses contribute by their gate-drain capacitance of S7, S8 respectively.
- The chosen accuracy can be implemented at an acceptable silicon consumption; in the current implementation about 25 % of the synapse area are used for the capacitance. A similar argument holds for the other circuits that consequently would need to be designed for a higher accuracy: the DACs and output neurons. There, a higher accuracy not only requires more silicon area, but most likely also a larger power consumption.

In practice, the actually realizable accuracy more likely is limited by external noise, i.e., fluctuations in the power supply during the weight programming or activity dependent cross-talk from signal lines such as *ceval*. Additionally, the switching of the current sunk from either I_+/I_- or I_{park} allows charges to flow between the signal branches and the park branch in the transition moment. Depending on the potential difference and programmed weights of the inactive synapses, the current on the signal lines may for a short period be significantly higher than an LSB current. If the settling time is tightly chosen, this may be visible as noise. In order to adjust this later on, the clock pattern generation is shifted of chip (c.f. Sec. 3.3.1).

Lastly, the weights degrade over time due to leakage currents in the switch transistor S4. Basically, there are two contributions: The subthreshold channel leakage of the transistor which in the utilized process technology is given to be about 0.5 pA for a PMOS of the dimensions of

[4] These variations occur in CMOS processes and may be observed for dice from different wafers.

S4 and a V_{DS} of 3.3 V [15]. Since the potential difference of I_{park} and the weight capacitor S3 is much smaller, the second leakage contribution is expected to be dominating: The leakage due to reverse currents across the pn-junction in the diffusion region. This leakage will cause the weights to become larger over time since the capacitor S3 gets charged to V_{dd}. In Sec. 5.3.2 the effect on trained networks is examined.

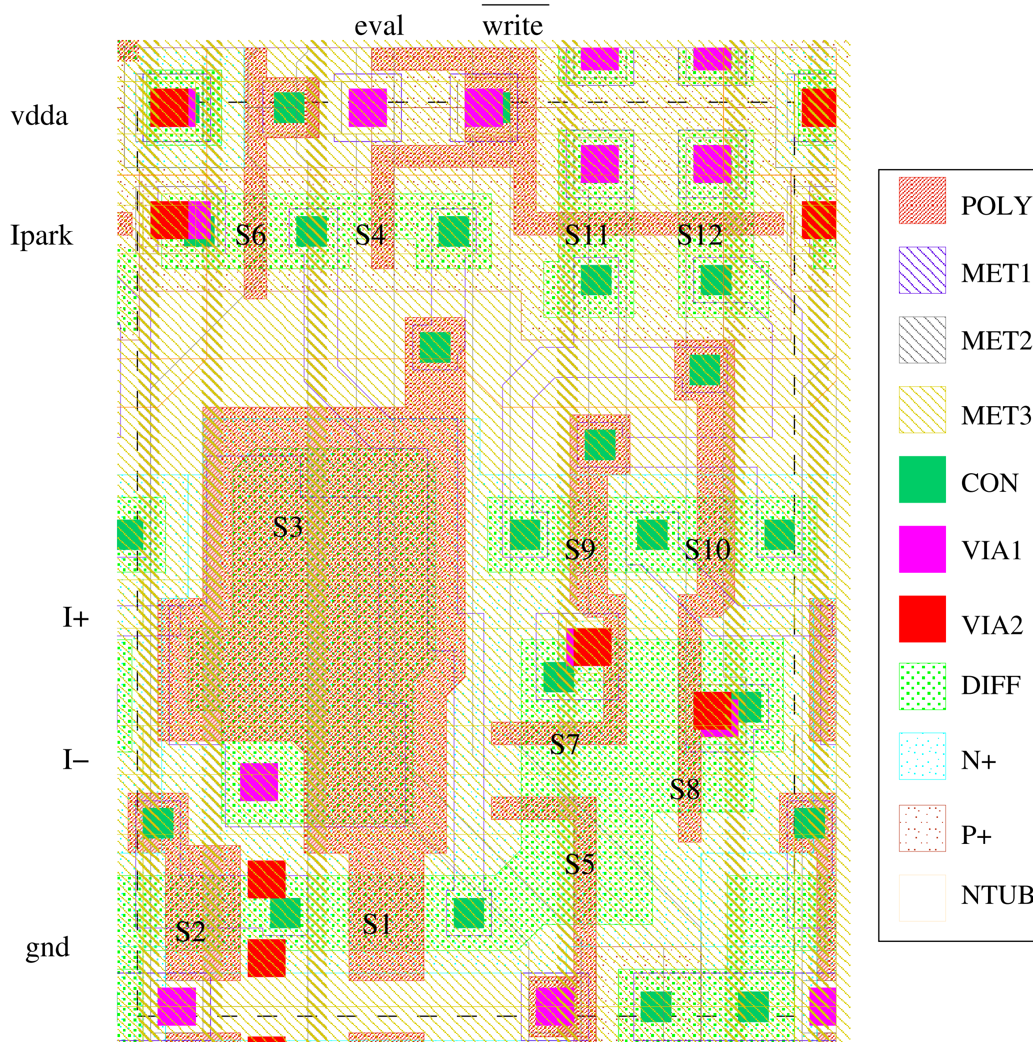


Figure 3.12: Layout plot of the synapse circuit which shows the seamlessly integration to the surrounding synapses. The synapse is dominated by the capacitive weight storage S3. The effective area of the synapse is indicated by the dotted line, i.e., dimensions of 8.7 by 12 μm^2.

Besides the accuracy there are other design goals such as the efficiency of the implementation, i.e., a minimum number of transistors at small size. This is especially reasonable because the synapse circuit is the most repeated circuit of a network block. All switches have minimum length. The ones located in the current paths I_+ or I_-, i.e. S7, S8, and S5, are wider (2.35 μm) in order to reduce the resistance and consequently the voltage drop across them. Switches S4, S6, S11, and S12 are not critical in terms of sizing. Nonetheless, a more sophisticated clocking scheme for S5, S6 may have minimized the mentioned shortcut of I_+/I_- and I_{park}. On the other hand, it would have come at the cost of an additional signal. Since a relaxed network cycle timing can compensate the effect, the design is kept simple. As can be seen from Fig. 3.12, the weight capacitor is the

dominating part of the synapse. In order to compensate for the increased area consumption caused by the use of both, PMOS and NMOS [14], every second synapse row is mirrored as to share the n-well (NTUB). Since the postsynaptic lines as well as the power runs horizontally across the synapse and column-wise addressing by the input neurons is necessary, only one metal (MET1) and poly-silicon layer (POLY) can be used for the cell design.

3.3.3 Neuron

The task of the output neuron is to evaluate in which of the postsynaptic lines, I_+ or I_-, more current is flowing and to present the result as a yes/no answer for the question $I_+ > I_-$. In order to accomplish this task the neuron should provide an adequate accuracy as to evaluate single LSB increments of the synapses. Furthermore, it is important to minimize the time needed for the settling and evaluation because this is the time Δt of a single network cycle which translates into a maximum network frequency.

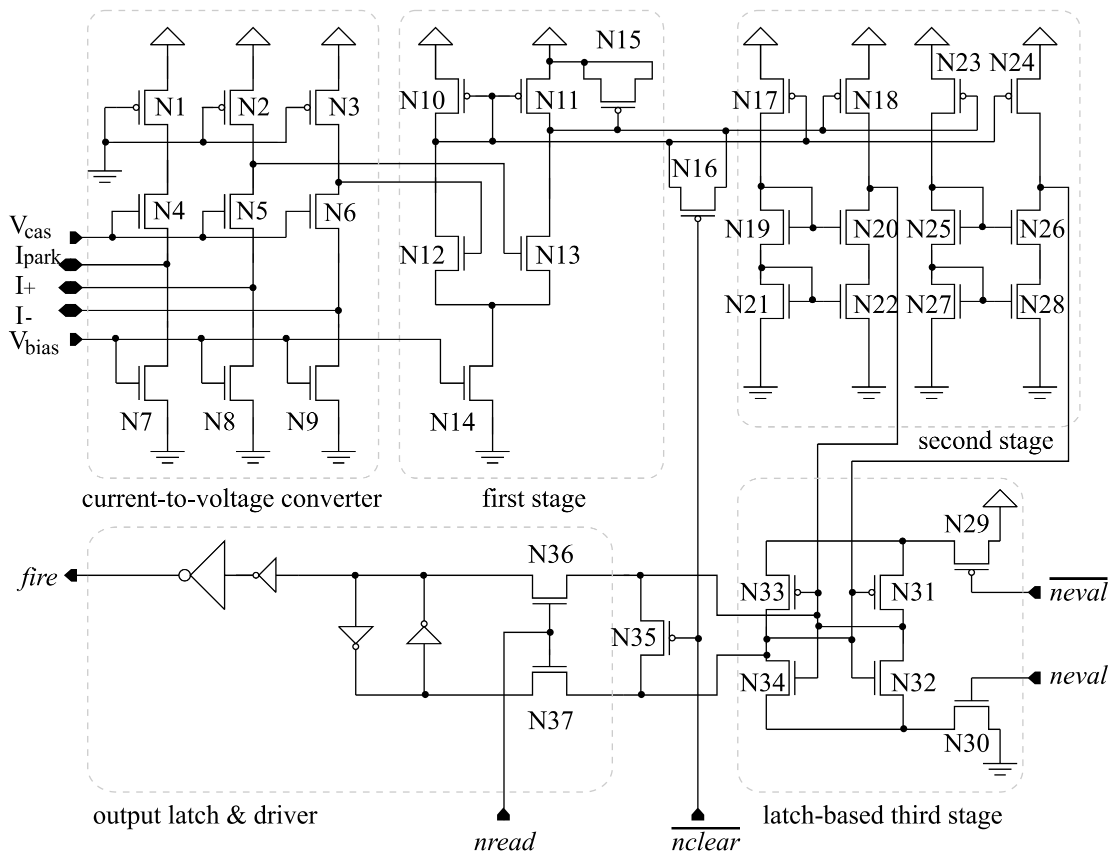


Figure 3.13: Circuit diagram of the output neuron. Figure adapted from [184].

Electronically, the neuron operation can be implemented by a comparator. The accuracy/dynamic range requirement consequently translates into a sensitive input with a high *common mode rejection ratio* in order to resolve a 1 bit increment of a single synapse across a sufficiently large dynamic range. The time requirement puts a restriction on the propagation delay from the neuron input to the output.

The presented neuron circuit (see Fig. 3.13) is designed and simulated to provide a 50 nA resolution across a 300 μA range. This represents a dynamic input range of 76 dB. The evaluation

at this accuracy can be accomplished in only 15 ns, i.e., the propagation delay from the voltage inputs of the first stage to the *fire* output. This performance is reached in all process corners as verified by a Monte-Carlo analysis of 100 transient simulations varying the process parameters. In the course of each transient simulation the neuron had to correctly evaluate the transitions of the input currents (c.f. Eq. 1.8) $I_{\text{pos}} = 300\mu\text{A}$, $I_{\text{neg}} = 299.95\mu\text{A}$ to $\text{I}_{pos} = 50\text{nA}$, $\text{I}_{neg} = 100\text{nA}$ and vice versa. This is achieved by employing a fully differential multi-stage setup. The design is not only constrained by the timing and accuracy requirements, but also by the layout: it is desirable that the neuron circuit can be laid out with the pitch of the synapse array, which is only 12 μm in height. This imposes restrictions on the dimensioning of the transistors if they are to be matched. The actual silicon area consumption is 180 by 12 μm^2.

Weight Summation and Current-to-Voltage Conversion Before the postsynaptic activity can be evaluated by the voltage-mode comparator the currents I_{pos}, I_{neg} need to be converted to a voltage. This is done for the two postsynaptic branches as well as the I_{park} branch to provide inactive synapses, i.e., synapses with for which *ceval* is low, a current source similar to the active case.

The conversion is realized according to Ohm's law: The necessary resistances are realized by N1-N3 operated in the Ohmic region. The gate-source voltage V_{GS} is set to the maximum of 3.3 V in order to assure linearity across the desired 300 μA current range as good as possible. Introducing the cascodes N5 and N6 biased by V_{cas} provides a low input impedance to the current inputs. If the output resistance of the cascodes is much larger than the load resistance (N2 and N3), the input resistance of the cascodes N4-N6 is according to [75] anti-proportional to the transconductance g_m. Therefore, choosing a large W/L ratio (40/0.3) for N4-N6 provides the low-impedance input. Consequently, the voltage swing at the source connectors of N4-N6 caused by changes in the currents on the postsynaptic wires is minimized.

N7-N9, finally, are current sinks for quiescent currents in the branches; they are controlled by V_{bias}. This bias current keeps the cascodes operational in case no synaptic input current is present. This happens if either all synapses are programmed to zero or an all-zero input is presented to the network. Setting V_{bias} to 0.8V, the quiescent current through each of the branches is about 5 μA. With this setting, the neuron achieves a common-mode range of 0 to 300 μA in the signal branches. A very important aspect of this bias current is its contribution to the settling time t_{settling} of the signals in the branches I_+ and I_- in the beginning of each network cycle as described in Sec. 3.3.1: Since the activation of a synapse induces short-time currents (c.f. Sec. 3.3.2) that may be larger than an LSB current of a few tens of nA and due to the non-negligible capacitance of the branches I_+ and I_-, the time needed to have the correct voltage difference established between the cascode sources would be unacceptably long if it was only caused by a single synapse sinking a few or even only 1 LSB current. Yet, even with a quiescent current of 5 μA a settling time of 5 ns may yield wrong results for sign changes caused by a single-LSB current difference and a common-mode current of near 0 A.

It is the very property of implementing the synapse and neuron circuits in analog VLSI that the performance according to one design variable can be improved on the cost of another. The schmoo plot of Fig. 3.14 explores the interdependency of the analog design variables: settling time, bias current, and LSB current. For each parameter set (settling time, bias current) a transient analysis of the neuron circuit connected to two synapses is performed and the failure (black square) or success (white square) according to the following measure is plotted: Simulated are successive network cycles in which the neuron output has to switch. One synapse sinks a current of 1 LSB and has positive sign, the other sinks a 1 LSB current and has negative sign. The synapses are activated in an alternating fashion, which should cause the neuron to fire in one cycle and to

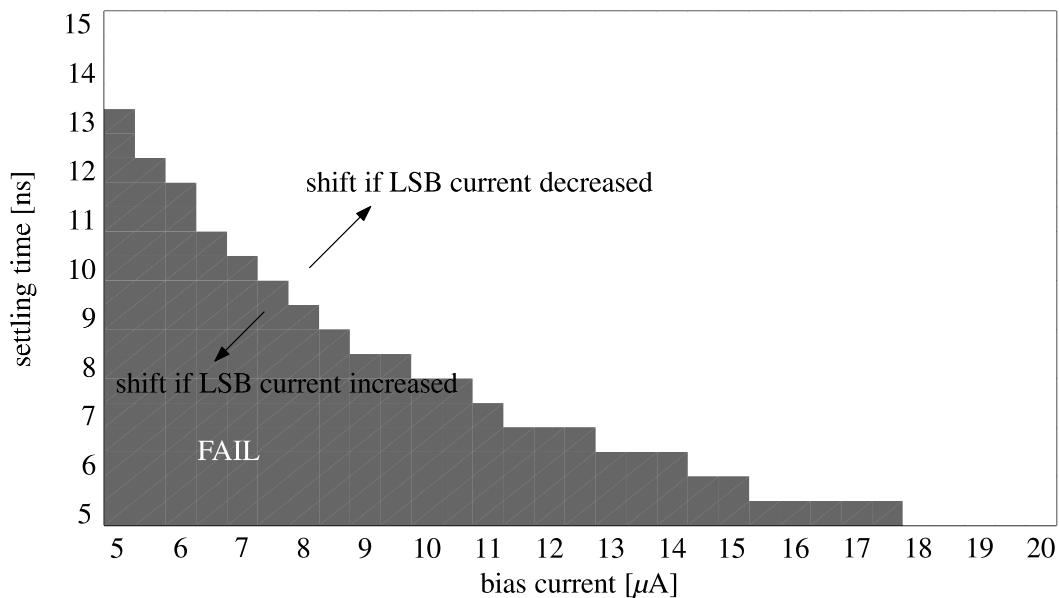


Figure 3.14: Schmoo plot of the neuron's performance in correctly identifying a 1 LSB current difference at a common-mode current of 0 A in dependence on the design variables settling time (time span between activation of weights and the release of the *nclear* signal) and bias current. The plot is derived from independent transient analyses for each parameter set. The LSB current is set to 50 nA.

stay silent in the other. The simulation performed uses a typical-mean model and emulates the remaining 126 synapses by appropriate capacitances. The output neuron is operated with a 15 ns long clocking scheme, details of which are described in Sec. 3.3.1; the LSB current is chosen to be 50 nA. The settling time t_{settling}, is varied from 5 to 15 ns in 0.5 ns increments. Taking into account the evaluation time t_{compeval} used by the voltage comparator (15 ns), this corresponds to network frequencies from $f_{\mathrm{net}} = 50$ MHz to $f_{\mathrm{net}} = 33$ MHz. The bias current is varied from 5 μA to 20 μA in 0.5 μA steps; it is induced by replacing N7-N9 by ideal current sinks.

Fig. 3.14 illustrates that the neuron performance for correctly evaluating a single-LSB difference will decrease if both, settling time and bias current, are too small. The capacitances of the signal lines and the LSB current impose this limitation. Increasing the LSB current of the synapses slightly enlarges the parameter space for successful runs, decreasing it diminishes the parameter space respectively. As a practical consequence, the neuron should not be used to evaluate very few very small weights unless one is willing to reduce the network frequency. Rather, for a high accuracy it is desirable that the weights are chosen as to provide a minimum average (common-mode) current of several μA on both input branches according to Fig. 3.14. This can either be done by adjusting the weight distribution or by using dedicated synapses as bias synapses. Effectively, the lower bound of the neuron's common-mode range is increased.

In order to save a pin for a external reference voltage, V_{bias} was chosen not only to control the bias current of the converter stage but also of the input state of the comparator. This design decision does not allow to freely set the bias current for adjusting the settling time and should be changed in a future design.

Voltage-Mode Comparator The input stage of the voltage-mode comparator is a differential amplifier built from a source-coupled differential pair with a current-mirror load (N10 to N14).

This design was chosen since it allows a high differential voltage gain across a common mode range of about 2.2 V to 3.3 V and provides a good common-mode rejection ratio [75]. The current-mirror load built from N10 and N11 does not only increase the gain but also is self-biasing [67, 116]. The common-mode range is achieved by operating the differential NMOS pair N12 and N13 not only in saturation but also in the Ohmic region. The voltage gain may be reduced in the Ohmic region (i.e., for small synaptic currents), but the increase in common-mode range leads to an overall higher resolution.

In order for the large possible differential voltage gain to be realized, the large output resistance of the active load N10 and N11 requires a large input resistance in the next stage. This requirement is met by connecting the differential outputs only to gates of the second stage (N17, N18 and N23, N24). In between the two stages, N16 is introduced to level the differential output on the $\overline{nclear}$ signal. It is issued prior or simultaneous to the evaluation of the postsynaptic signals in each network cycle. Its purpose is to minimize the time needed by the first stage to establish a correctly signed potential difference on the input gates of the following stage. Especially, if a large current difference on the postsynaptic wires is followed by a small difference with opposite sign, the large output resistance of the first stage would cause a considerable delay before the gates N17, N18 and N23, N24 show the correct potential difference. N15 is connected as a capacitor and sized to compensate the gate capacitances of N10 and N11 connected to the other differential wire. This prevents the introduction of an offset due to charge injection if N16 switches off.

A possible offset induced by the differential pair N12 and N13 is minimized by choosing a non-minimal W/L ratio (10/0.5) and carefully matching the transistors in the layout. N14 provides the bias current and is controlled by V_{bias}.

The second stage comprised of N17 to N28 realizes two push-pull output stages with differential input. It is different from a classical inverter stage in that it makes use of both differential inputs: one of the differential inputs is used to control the PMOS N18 (N24 respectively) in the output branch directly. The other connects the gate of the equally sized PMOS N17 (N23); here, the source-drain current is mirrored with a cascoded current-mirror to the output branch built from N18, N20, N22 (N24, N26, N28).

Task of the last stage, finally, is to amplify the voltage difference to the supply rails and store this binary information until the next network cycle. A dynamic latch—as widely employed for sense amplifiers in dynamic RAM—is used for these purposes (N31-N34). One known problem of dynamic latches is the susceptibility to process parameter variation which causes an input offset [42]. For this reason, the latch does not get activated until the preceeding stages have amplified the differential voltage to at least 150 mV, which is well beyond the expected worst case offset [184]. N29 and N30 activate the latch upon the *neval* and $\overline{neval}$ signals; N35 resets it upon the $\overline{nclear}$ signal which simultaneously is used to clear the first stage. A possible timing that is as well used in the simulations presented here was described in 3.3.1.

When the *nread* signal is issued, M36 and M37 connect the output latch comprised of I1 and I2. Here, the comparator decision remains stored until the next network cycle. The output buffer built from I3 and I4 is dimensioned as to drive the *fire* signal to the bus interface and the internal feedback.

As mentioned earlier, the presented neuron circuit provides 76 dB dynamic range in 15 ns independently of the process variations simulated by a 100-run Monte-Carlo analysis. Nevertheless, the circuit is susceptible to device mismatch, especially, that between the differential pair N12 and N13. These mismatches primarily cause the comparator to have an offset, but mismatches in the converter-stage may lead to non-linear errors, effectively reducing the accuracy.

With the help of a Monte-Carlo analysis the expected neuron offsets can be predicted. Yet, since it is computationally quite expensive to predict the neuron offsets exactly, an indirect method

PSfrag replacements

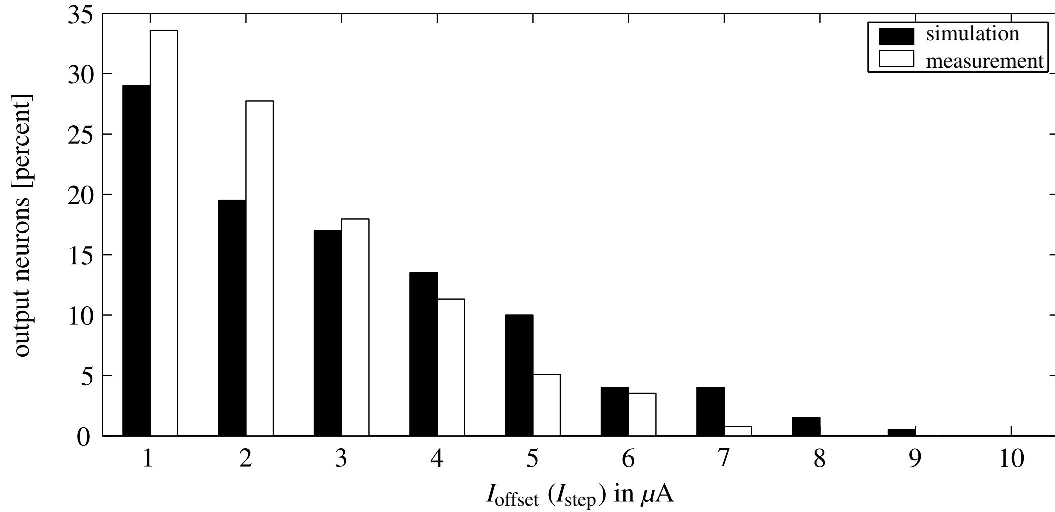


Figure 3.15: Distribution of the absolute neuron offsets predicted by a Monte-Carlo analysis (black bars) and a die-measurement of a HAGEN prototype (white bars). The actual neuron offset is essentially a zero-centered Gaussian distribution as illustrated in the die characterization in Fig. B.3 of the appendix. Yet, for simulation reasons only the absolute offset is displayed; furthermore, the data for the respective offsets is derived from independent simulations, the bars therefore may not add up to 100 % (see main text). While the Monte-Carlo analysis slightly overestimates the actual offset spread, it can be seen that it is an adequate tool to predict the neuron offset variations.

is employed: In a Monte-Carlo analysis modeling the device mismatch, the neuron circuit is evaluated whether it correctly identifies the transition from $I_{\text{pos}} = 300\mu\text{A}$, $I_{\text{neg}} = 300\mu\text{A} - I_{\text{step}}$ to $I_{\text{pos}} = 50n\text{A}$, $I_{\text{neg}} = 50\text{nA} + I_{\text{step}}$ and vice versa. For a given I_{step} the correct runs yield the expected percentage of neurons with an absolute offset smaller than I_{step}. By increasing I_{step} and subtracting the percentage of previously correct results for smaller I_{step}, one gets a distribution of the expected absolute neuron offsets[5].

Fig. 3.15 shows the outcome of these Monte-Carlo analyses with 200 runs evaluated for each bin (black bars). The white bars in comparison show the actual measured absolute neuron offsets of the HAGEN die used later for the experiments in Ch. 5. The measurements to infer the neuron offsets from the network response were developed in [86] and are shortly described in Sec. 4.4.2. From the full measurement data (given in Fig. B.3 of the appendix) it can be seen that the measured[6] neuron offset distribution essentially is a zero-centered Gaussian with a sigma of about 2.3 μA.

Compared to the measurements, the distribution predicted by the Monte-Carlo analysis slightly overestimates the expected spread in neuron offsets but nevertheless yields a good tool to predict the variations. Since the offsets remain in the few percent region of the overall common mode range and they can be completely compensated for by bias synapses.

[5]Since the simulations for the individual I_{step} are independent, the bins of the resulting distribution may not add up to 100 %.

[6]Measurements of other dice [86] result in similar neuron offset distributions.

3.3.4 Weight Storage

The *wren* signal switches between the recall and programming phase. The programming phase is activated upon *wren* being high. In this phase, the input stage of the output neuron gets deactivated by setting V_{cas} and V_{bias} to zero. Secondly, the deactivated current sources of the output neuron in case of I_{park} get replaced by current sources of the analog weight storage unit; in case of the signal branches I_+ and I_-, the branches are either connected to V_{dd} or ground. A simplified overview of the programming phase is illustrated in Fig. 3.16.

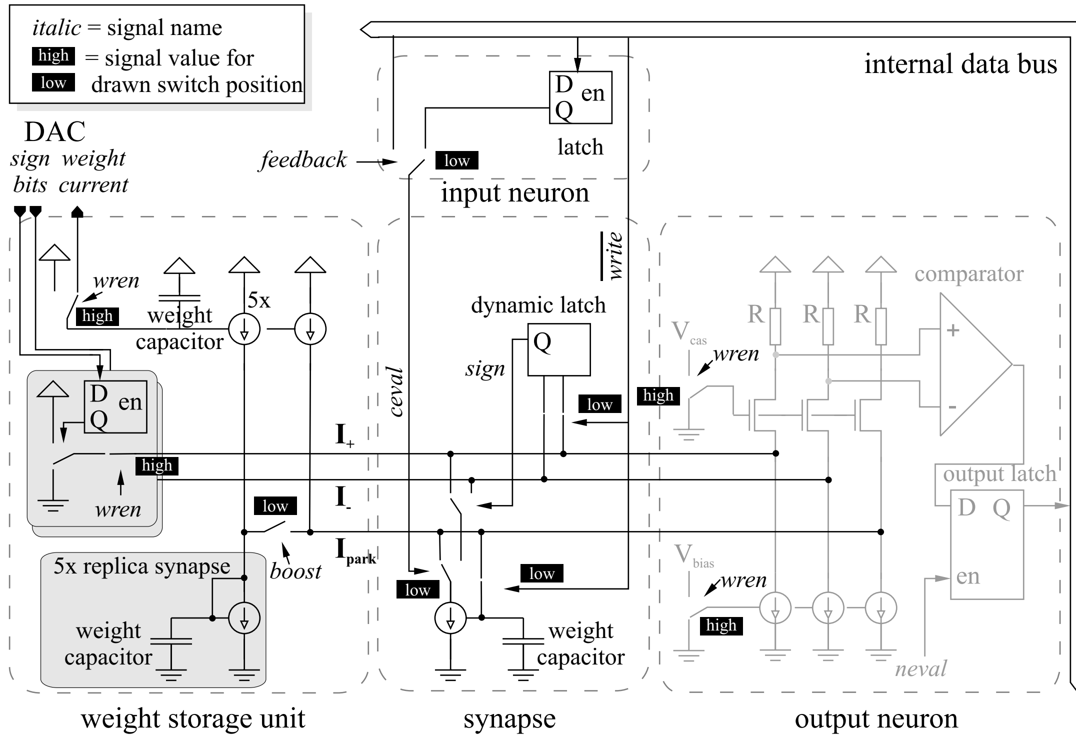


Figure 3.16: Schematic diagram of a network block simplified to one input, one output and one synapse.

The weights are written column-wise to the synapse array. Accordingly, all current memories have to be written by the means of DACs before the weights can get transferred to the array. Once programmed, the column of synapses to which the weights are transferred is selected by the input neuron decoding the column address (decoder not shown); this is done by lowering the $\overline{write}$ signal for a period of time, $t_{colprog}$. The duration of this period is determined by the time needed to program the current memories in the synapses. The weight current is written using the I_{park} branch and due its capacitive load of about 700 fF small weight currents will need longer to establish the respective potential on I_{park} which adjusts the current sink in the synapse to the correct current. The worst case programming time is to be expected for a very small weight current following a very large weight current; then the potential of I_{park} undergoes the full 1.5 V voltage swing of the synapse current sink (c.f. Fig. 3.11).

Eventually, considerations on the conversion speed of the employed current DACs, the amount of DACs used and the number of current memories to fill make a maximum programming time per column of $t_{colprog}$ = 320 ns desirable. This is the time the eight DACs of a twin block need to convert a full column of 64 weight values (see Sec. 3.4). In order to ensure this timing requirement in all cases it is necessary to accelerate the programming. This is achieved by a second current

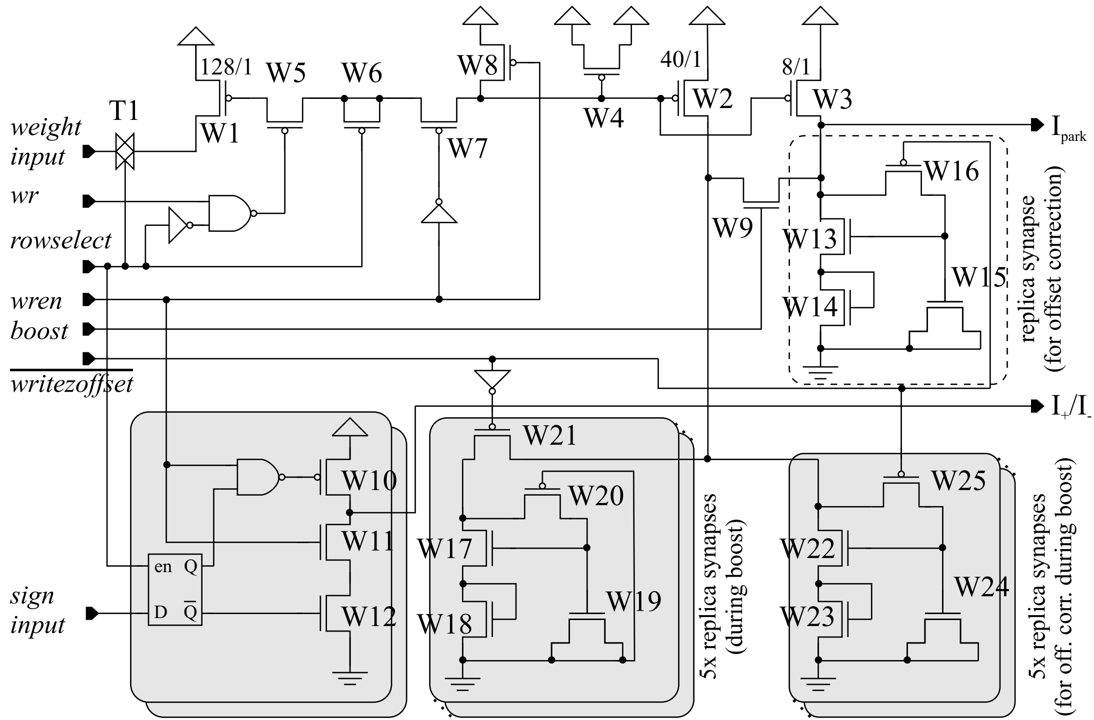


Figure 3.17: Circuit diagram of the weight storage unit. Figure adapted from [184].

source sourcing a fivefold larger current than the desired weight current. Upon the *boost* signal not only the current source but also five replica synapses get activated that each sink the desired weight current. In effect, a six times larger current is sunk, yet, the potential of I_{park} due to the replica setup is close to the single synapse configuration. The remaining difference and the charge injection by disconnecting the synapse replicas finally are adjusted by the actual synapse's current sink (a good value for *boost* being active is according to simulation half the programming period; nevertheless, the exact duration can be programmed). Yet, there remains one systematic effect to the programmed weight: the release of $\overline{write}$ causes the switch S4 of the synapse (c.f. Fig. 3.10) to inject surplus charges on the weight capacitor S3 which causes the weights to be systematically larger. For small weights this effect should be less than an LSB, for large weights it can account for weight values actually larger by a couple of LSBs. This effect principally can be compensated by appropriately programming the DAC. Yet, the device mismatch in S4 between the synapses is a source for fixed-pattern deviation in the synapses. With the help of a measurement routine developed in [86] and shortly described in Sec. 4.4.2, the individual synapse offsets can be measured. The results show that the indirectly measured fixed-pattern offset for small weights is comparable to the temporal variation and has a standard deviation of about 2.5 LSB. Results for an individual HAGEN die are given in Fig. B.3 of the appendix.

Additionally to the weight, the sign of the synapse has to be written. Upon the $\overline{write}$ signal the dynamic latch of the synapse gets connected to the I_+ and I_- branches by two switches S11 and S12 (c.f. Fig. 3.10). By setting I_+ to V_{dd} and I_- to ground the synapse will sink its current from the I_+ branch in recall phase and vice versa. This is because charges from the I_+ branch will flow on the gate of S10. Once conducting, it discharges the gate of its cross-coupled counterpart S11. The charge injection caused by S11 and S12 at the release of $\overline{write}$ reinforces the state of the latch. Setting both branches to ground completely deactivates the synapse.

Fig. 3.17 illustrates the circuit diagram of the weight storage unit in detail. Only the row address decoder is omitted which activates the *rowselect* signal. The central part is the current mirror of W1 and W3. It reduces the current written to the synapse to 1/16th of the DAC current. The transmission gate T1 and the switch W5 allow to disconnect the current mirror from the DAC on the release of the *rowselect*. Since the capacitor-connected W4 keeps the appropriate gate potential, W4 and W3 build an analog current memory. The programming phase ends upon the release of the *wren* signal and opens W7 and closes W8; connecting W4 and W3 to V_{dd} effectively deactivates the current source W3. The current generated by the DAC (c.f 3.4) is by the amount of a bias current larger than the desired weight value. This is accounted for by the replica synapse comprised of W13-W15 connected in parallel to the synapse in the array which always subtracts its programmed value from the current sourced by W3. The current generated by the DAC on the zero pattern is exactly the bias current; it is stored to W13-W15 by setting $\overline{\textit{writezoffset}}$ to zero and causing W16 to open. The replica synapse for the offset correction essentially represent an additional synapse column to be programmed.

The described boost mechanism for accelerated weight programming is realized by a second current mirror comprised of W1 and W2. The drains of W2 and W3 are connected on the *boost* signal by W9. At the same potential stored on W4, W2 sources a fivefold larger current. W17 to W20 represent one of the five replica synapses needed in parallel to have the target synapse sink the correct current again. Since the DAC offset affects this boost current as well, five additional synapse-type current sinks (the one shown is comprised of W22-W25) for the offset subtraction are implemented. The current mirrors and the 11 replica synapses use about two thirds of the 230 by 12 μm^2 area covered by the weight storage unit. In order to minimize mismatch due to gradients (in process parameters or temperature) a common-centroid geometry is employed [220, 217].

Similar to the buffering of the amplitude of the weight in the analog current memory, the sign of the weight is stored in two latches. As described, two sign bits are used to program the sign latch in the synapse. This is done by driving I_+ and I_- by tri-state inverters (W10-W12).

3.4 Digital-to-Analog Converters

The leakage of the synapses' capacitive weight storage (c.f. Sec. 3.3.2) make regular weight refreshes necessary. But during programming phase the twin block is unavailable for data processing. It is therefore desirable to keep the time needed for programming small. One aspect is the number of DACs available in parallel as discussed earlier (c.f. Sec. 3.2); another is the actual implementation of a single DAC used and its conversion time.

Several architectures for DACs are conceivable and implementable in CMOS; see [215] for an overview. For the desired 10 bit resolution and conversion times of a few tens of nanoseconds, a current-steering DAC with segmented architecture is an appropriate choice [226]. Having the converter segmented into a binary encoding of the lower bits and a thermometer encoding of the more significant bits allows good monotonicity at a reasonable area consumption.

The actually implemented DAC (see [184] for circuit details) is a current-steering flash converter with the upper four bits implemented in thermometer code. It provides 10-bit resolution and a conversion time of 40 ns. Its dimensions are 235 by 95 μm^2, i.e., eight of these DACs use about 6% of the silicon area of the complete twin block. The DAC employs a double-buffered interface to allow a continuous operation. The input registers illustrated in Fig. 3.18 can be filled at the full interface speed of 300 MHz.

Due to the current-steering approach the DAC has a constant power consumption independently of the input code. This power consumption—and along with it the magnitude of its max-

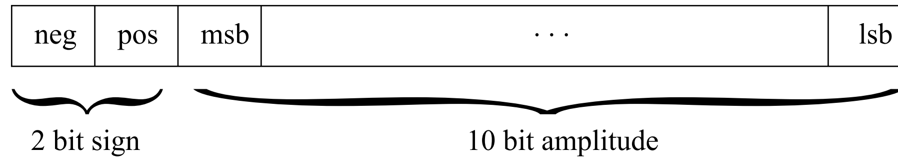


Figure 3.18: Data structure of the 12 bit DAC input register. The upper two bits are used for the sign, the remaining 10 for the weight amplitude. The four most significant bits of the amplitude are converted to a thermometer code.

imum output current—is controlled by an external reference current I_{ref}; the maximum output current is $I_{\mathrm{out}}^{\mathrm{dac}} = 8I_{\mathrm{ref}}$. The 10-bit amplitude value in the input register, finally, determines how much of this $I_{\mathrm{out}}^{\mathrm{dac}}$ current is drawn from the DAC itself or from the source transistor W1 in the analog weight storage unit (c.f. Sec. 3.3.4). The current sourced by the latter is reduced by the current mirror in the analog weight storage unit to 1/16th, which finally resembles the synapse weight.

Thus, the numerical weight value $\omega \in [-1, 1]$ as introduced in Eq. 1.1 in the HAGEN prototype has to be approximated by $|\omega| \approx \hat{\omega} \in [0, 1023]$ which yields the actual synapse current:

$$\omega_{\mathrm{curr}}^{\mathrm{syn}} = \frac{\hat{\omega}}{1023} \frac{I_{\mathrm{ref}}}{2}. \tag{3.3}$$

According to the sign of ω the current is drawn from I_+ or I_-.

By design, the DAC has a positive offset current, i.e., even for the zero input code a non-zero current is drawn from W1. This allows to bias the current mirror in the analog weight storage unit but simultaneously introduces an offset in the synapse weight. This DAC offset is always positive and—once measured, e.g., by the method introduced in [86] and shortly described in Sec. 4.4.2—can be subtracted by the dedicated synapses in the analog weight storage unit (c.f. Sec. 3.3.4).

3.5 Interface

It was described earlier in Sec. 3.2 that the encapsulation of two network blocks with eight DACs to a twin block allow an essentially digital[7] interface. The half chip entity of the HAGEN prototype provides the I/O and power pads to connect with the outside as well as a standard cell logic that generates the appropriate control signals for the twin block, the *bus interface*. The two halves of the HAGEN prototype are identical and can be operated independently. To meet the performance demands of the proposed concept for an efficient and scalable mixed-signal neural network ASIC, it is desirable that the interface:

- allows a high data rate per link;
- allows a low power implementation;
- is electrically robust against noise and crosstalk;
- minimizes the electromagnetic distortions coupled into the analog network blocks.

These requirements can be met by low-swing differential signaling. The advantages of differential data transmission are manifold but are eventually related to noise [76]:

[7] In addition only three reference voltages and one reference current are required.

1. A differential link is *less susceptible* to external noise than a single-ended one. This is because an electrical distortion that couples onto both wires will subsequentially be rejected by the receiver evaluating only the difference of the wire potentials.

2. A differential link *generates less* noise since its net magnetic field almost is canceled due to the opposite single fields induced by the current flowing in opposite directions.

Being less susceptible to noise allow the realization of good signal-to-noise ratios with smaller signal swings. This in turn generates even less noise, reduces the required power consumption, and simultaneously allows shorter transition times. A specification suited for CMOS implementation is the LVDS standard [10] based on current-steering; it is for example used by the HyperTransport technology [39]. Adopting this standard for the HAGEN prototype furthermore allows the straight forward interaction with commercially available programmable logic devices, such as the Virtex-E [233] or Virtex-II pro [232] by Xilinx, Inc. This is described in detail in Ch. 4.

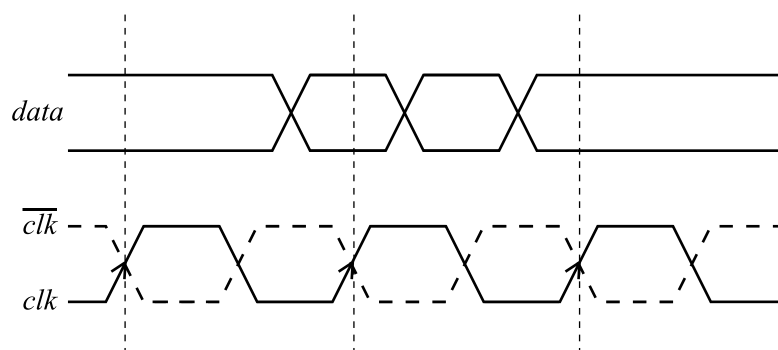


Figure 3.19: Illustration of a double data rate interface.

Two additional requirements have not yet been explicitly mentioned: First, for the HAGEN prototype implementation it was desirable to keep the absolute number of I/O pads to a minimum. One way to achieve this is double data rate transmission, i.e., with each edge of the clock the data can change. Fig. 3.19 is a sketch to illustrate this; yet it does not show the real timing of the HAGEN prototype. Another is the bi-directional use of the links.
Secondly, the interface should be scalable, i.e., signals that can be used by multiple half chips or complete HAGEN prototypes should not be replicated. This is easily realized since LVDS allows a bus architecture with several receivers (bus-LVDS). The differing latencies can be best accounted for by a source synchronous timing.

3.5.1 Physical Layer

The electrical interface of a half chip consists of 13 LVDS links; eight are bi-directional, called *data<0:7>*; four are uni-directional with the half chip being the receiver, called *ca<0:2>* and *clkin*; and one is uni-directional with the half chip being the sender, called *clkout*. Communication *towards* the half chip is synchronous to the clock *clkin* and will be regarded as *writing*. Communication *from* the half chip is synchronous to the clock *clkout* and will be regarded as *reading*.

The presented interface implementation with double data rate is designed and simulated to run at frequencies of up to 300 MHz. Accordingly, with each edge of *clkin* an 11 bit word can be written; this totals to a maximum net write rate of 6.6 Gbit/s. Data can be read from a half chip in 8 bit words with a maximum net read rate of 4.8 Gbit/s.

Since the bus interface implements no self-alignment of the clock and yet to allow the interface to be operated at a wide range of frequencies (up to 300 MHz), the bus interface realizes only a minimal delay of the *data<0:7>* signals. It is therefore necessary for a robust writing to ensure equal trace lengths on the printed circuit board. Data originating from the half chip, i.e., *clkout* and *data*, have no initial phase shift.

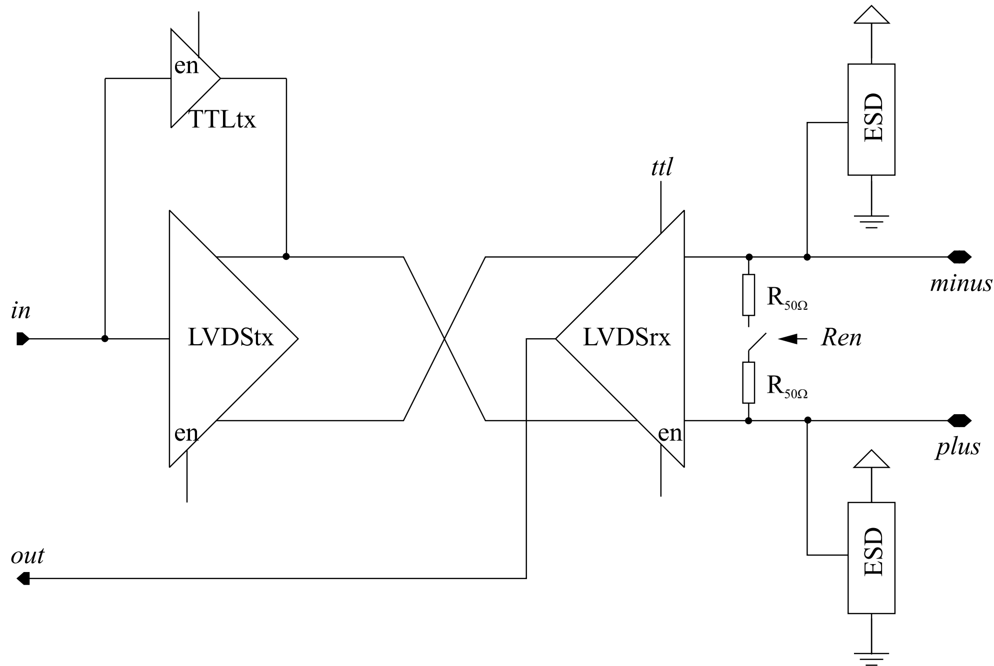


Figure 3.20: Overview schematic of the bi-directional LVDS cell with single-ended fallback.

A full-custom LVDS I/O pad has been developed in order to accommodate the interface needs. Fig. 3.20 illustrates the components integrated in the bi-directional LVDS pad. In addition to the LVDS transmitter and receiver, a switchable on-chip termination resistance is provided. Finally, the I/O pad can be operated as a single-ended transceiver. For this purpose, a standard tri-state output buffer is added which can drive the plus branch with 2 mA drive strength. The minus branch is used to bias the LVDS receiver as to explicitly set the threshold level for the plus branch.

The circuit diagrams of the LVDS transceiver and receiver are based on two separate CERN LVDS designs which are part of a radiation-tolerant standard cell library [113] in a 0.25 μm CMOS 6SF technology. Those have been ported to the AMS CSI 0.35 μm technology and integrated into a single cell. Features for a fast direction-switching, the switchable termination resistance, the power down of individual parts, and the single-ended fallback mode have been added. The presented implementation is designed, simulated, and tested to operate with frequencies of up to 300 MHz in LVDS mode and up to 100 MHz in single-ended mode respectively. In order to prevent the ASIC from destruction by *electrostatic discharge* (ESD), ESD structures as implemented in AMS I/O pads have been utilized. All functionality—including the switchable 100 Ohms p-diffusion resistor—have been fitted into the area of two standard pad cells, i.e. 200 μm by 340 μm, as shown in Fig. 3.21.

LVDS Transmitter The central part of the implemented LVDS transmitter shown in Fig. 3.22 are the transistors Tx1 to Tx4. According to the differential input signal $a/\bar{a}$ either Tx1/Tx4 conduct or Tx3/Tx2. If a is high, the current sourced by Tx5 flows in the *plus* branch and is allowed to

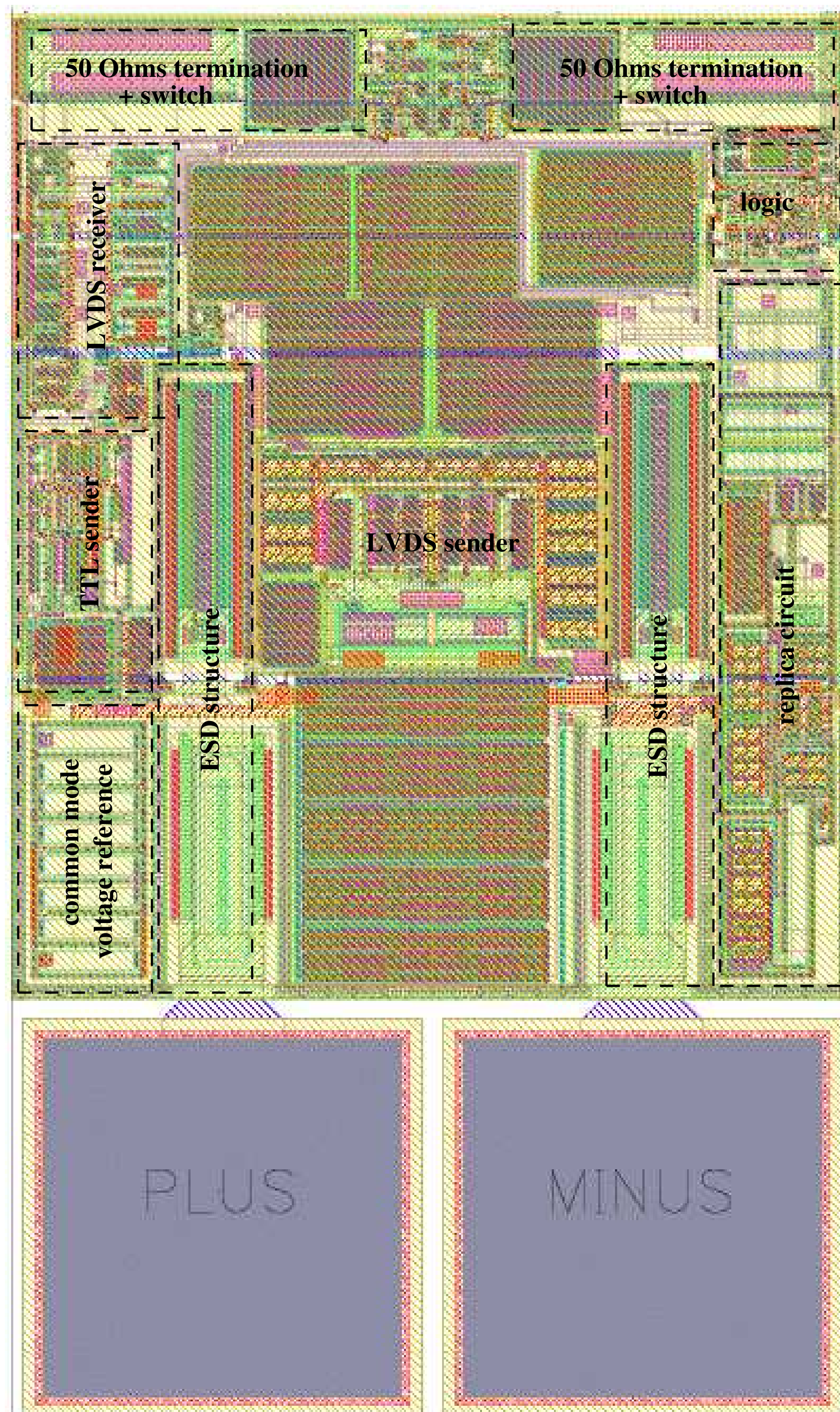


Figure 3.21: Layout overview of the bi-directional LVDS pad.

enter by the *minus* branch and is ultimately dumped by Tx6. Respectively, if $\overline{a}$ is high, the current direction changes. As regulated by the LVDS standard [10], the current sourced by Tx5 has to be between 2.5 mA and 4 mA. Across a 100 Ohms termination resistance therefore a voltage drop of 250 mV to 400 mV will be seen. Additionally, the output offset voltage, or common-mode voltage, is defined to be between 1.125 V and 1.275 V.

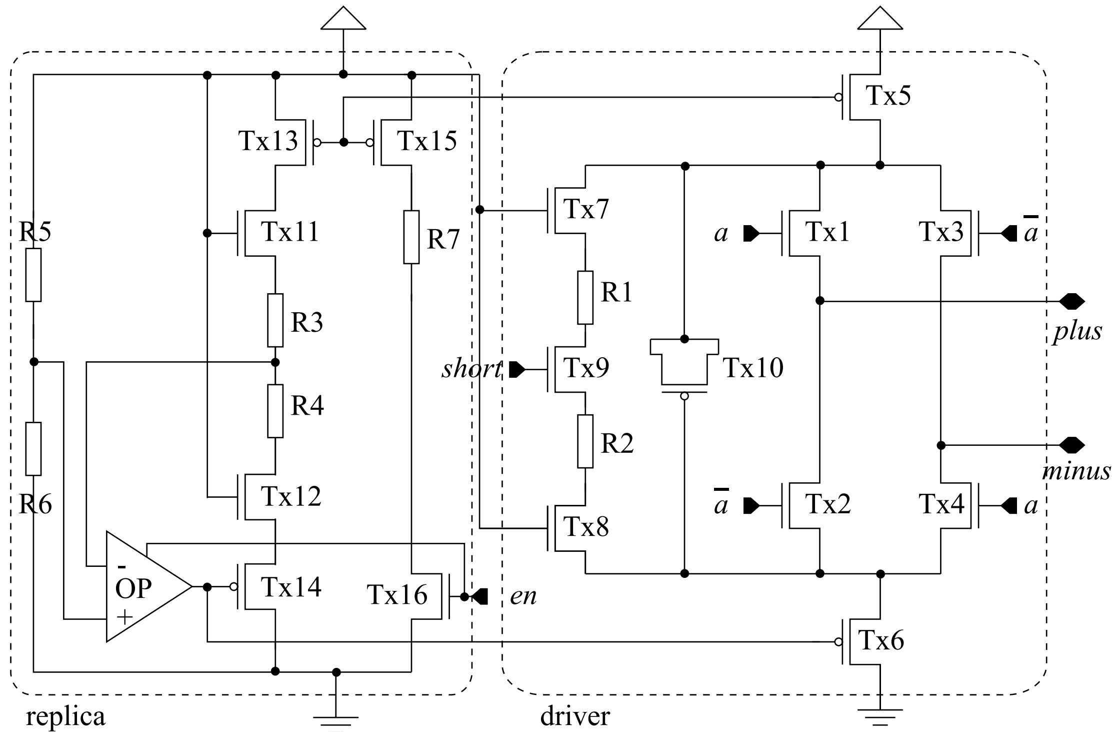


Figure 3.22: Circuit diagram of the LVDS driver circuits.

In the presented implementation, the gate potential of Tx5 controls the sourced current. It is generated by a 1 to 16 current mirror comprised of Tx15 and Tx5. The reference current is set by R7 which is an n-diffusion resistor of about 14 kOhms. The expected driver strength therefore is about 3.8 mA. Tx16 allows to deactivate the reference current and power down the driver upon *en* being low. For Tx5 to be a good current source a W/L of 1600/0.5 was chosen.

The common-mode requirement is met by operating Tx6 as a source follower. The W/L of Tx6 is chosen to be 2400/0.4 to achieve a small output resistance. The necessary gate potential is generated by an operational amplifier (OP) that compares the common-mode with a reference voltage set by R5/R6. The common-mode voltage is tapped in a replica circuit built from Tx11-Tx14 and R3/R4. The idea of the replica is to mimic a complete and closed driver/receiver loop with constant signal: Tx13 is the current source, Tx11/Tx12 are the switch transistors of the driver, Tx14 is the source follower. R3/R4 finally, are the termination resistance of the receiver. For power and area considerations the current is reduced by a factor 4 compared to the real driver, i.e., the current mirror of Tx15/Tx13 generates only a four-fold larger reference current. Accordingly, Tx11-Tx14 have a four-fold smaller width and R3/R4 have a resistance of 400 Ohms.

In order to allow fast direction changes it is necessary to keep Tx5 and Tx6 at their operating points while not sending (Tx1-Tx4 are deactivated). Therefore, another replica of the driver/receiver loop has been added (Tx7-Tx8 and R1/R2). The signal *short* controls Tx9 and activates the current bypass.

Tx10 finally is a capacitor connected PMOS of a couple of pF. In conjunction with the gate-source/gate-drain capacitances of Tx6 and Tx5 this helps to keep the potential at the drain of Tx5, respectively the source potential of Tx6, constant.

LVDS Receiver Once the current driven by the LVDS driver is forced through a termination resistance, the task of the receiver is to identify correctly the sign of the potential difference across the resistor. Essentially, this can be realized by a comparator. It is desirable for the comparator to have a large dynamic input range because this ensures the robustness against common-mode shifts. The here adopted CERN design is an implementation of a very-wide common-mode differential amplifier (VCDA) as reported in [17]. The circuit diagram is illustrated in Fig. 3.23.

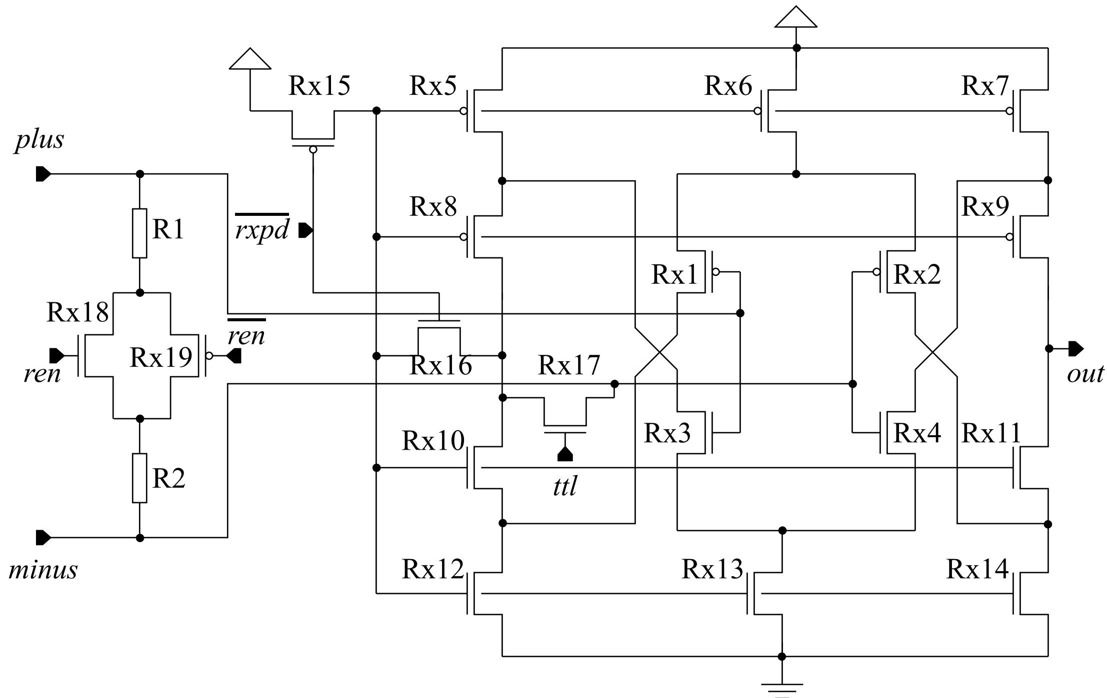


Figure 3.23: Circuit diagram of the LVDS receiver circuit and termination.

The presented design is a fully complementary, self-biasing differential amplifier with negative feedback. It basically consists of two differential amplifiers: one built around the PMOS pair Rx1/Rx2 from Rx5-Rx9, and the other built around the NMOS pair Rx3/Rx4 and Rx10-Rx14. Instead of a load, the outputs of the amplifiers are connected to each other; this allows the amplifier to operate in a push/pull fashion. The combination of two amplifiers allows to cover the complete input range from the negative to the positive supply rail.

Connecting the bias voltage of Rx5-Rx14 to the drains of Rx8 and Rx10 implements a negative feedback loop that stabilizes the bias voltage. Rx15/Rx16 allow to force the bias voltage to V_{dd} upon $\overline{rxpd}$, effectively opening Rx5-Rx7 and therefore powering down the receiver. Rx17 finally allows to externally bias the receiver which can be used to adjust the receiver threshold in single-ended transmission mode. The transistor sizing is chosen such to keep the rms current of the receiver to about 1 mA. The W/L for the PMOS pair Rx1/Rx2 is twice the ratio of the NMOS pair Rx3/Rx4 which is 10/0.4 to compensate the smaller transconductivity of the PMOS.

An switchable 100 Ohms on-chip termination is realized by the two p-diffusion resistors R1/R2 and the R_{on} resistance of Rx18/Rx19. Upon *Ren* the transmission gate comprised of Rx18

and Rx19 conducts and enables the termination. The W/L = 300/0.3 is such as to allow a switching within one clock period.

Eye-Diagram Besides the successful operation of the HAGEN prototype interface with 200 MHz (see Sec. 4.1.5), dedicated test structures have been submitted along with the HAGEN prototype in order to analyze individual LVDS transceiver cells. In these test structures two of the LVDS pads are interconnected internally (signal *out* being connected to *in*) with one pad being configured as a receiver and the other as a transmitter. This allows to externally feed a differential signal and have it sent out again as a differential LVDS signal.

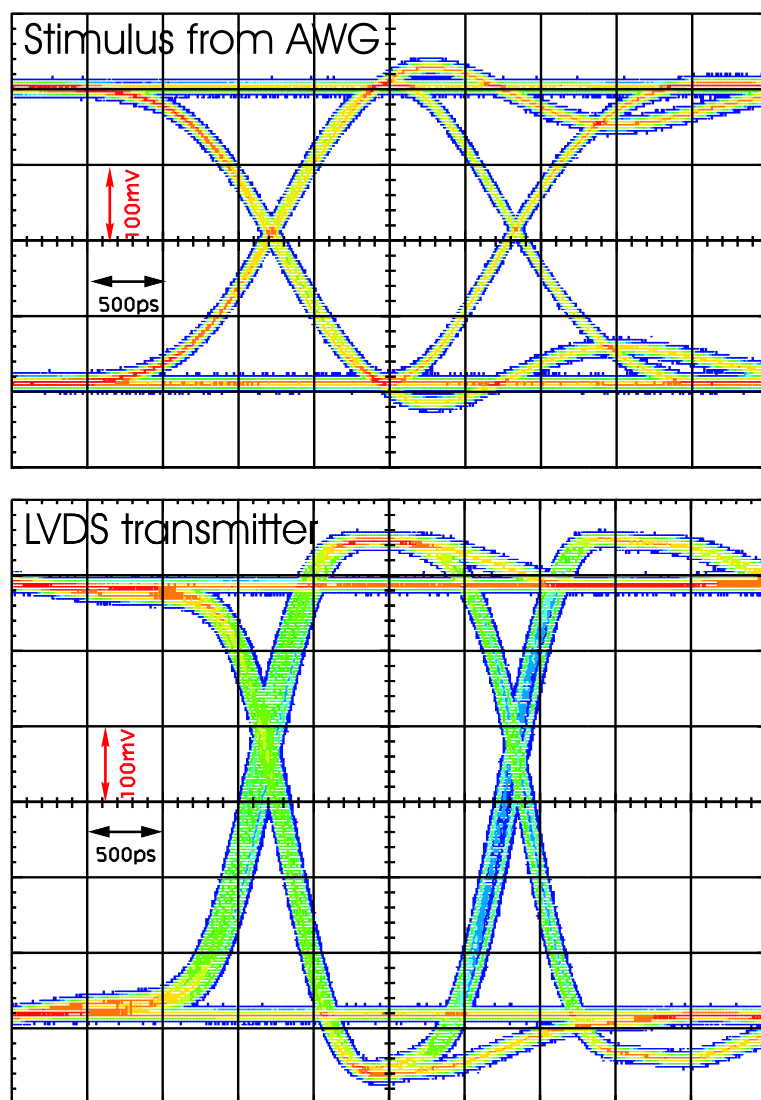


Figure 3.24: Oscilloscope shots of a 300 MHz input generated by an AWG (top) which is used to obtain the eye-diagram of the LVDS transmitter (bottom). Both plots are derived measuring across a 100 Ohms termination with a differential probe using the digital phosphor mode of a Tektronix TDS784D. A red color encodes a more frequently measured value; blue a less frequently measured.

Fig. 3.24 shows oscilloscope shots of two eye-diagrams: The top shows superimposed signals generated by an *arbitrary waveform generator* (AWG) HP 8130A [84] as measured across a 100 Ohms termination resistance by a differential probe (Tektronix P7330 [211] on a Tektronix TDS784D oscilloscope [210] using the digital phosphor mode). The pattern programmed into the AWG yields all possible bit transitions at a frequency of about 300 MHz. An appropriately programmed trigger signal allows to superimpose the consecutive patterns and to assess the jitter and capacitive effects of the LVDS link. The differential output signal of the AWG is connected to the LVDS pad configured as a receiver. The signal sent out by the respective LVDS transmitter is shown in the bottom. The signal is again measured across a 100 Ohms termination resistance.

The measured LVDS signal sent out by the LVDS pad shows a clearly distinct eye, i.e., the jitter of the first and the second transition is small enough to yield a window of about 1.5 ns. This setup tests the receiver as well as the transmitter. Due to the limitation of the AWG no higher frequencies could be tested, yet the small jitter suggests that the LVDS pad can be operated at even higher frequencies. Furthermore, it can be seen that the LVDS transmitter fulfills amplitude specifications [10] with its differential voltage of about 300 mV.

It has to be noted that the signal generated by the AWG with its 200 mV voltage drop across the 100 Ohms resistance is in fact slightly smaller than required by the standard (250 mV, c.f. [10]); this shows that the receiver is capable of coping with decreased amplitudes (tested down to 160 mV at a common mode of 1.2 V). Within the capabilities of the AWG furthermore common mode effects have been analyzed: A 200 mV differential output voltage has been successfully recognized by the LVDS receiver for a common mode from 0.4 to 1.6 V.

3.5.2 Logical Layer

In the preceding section it was introduced that with each clock edge an 11 bit word can be written to or an 8 bit word can be read from the half chip. This section deals with the meaning of these symbols.

Several HAGEN half chips can be connected to the same controller. In the current interface implementation, the encoding of the command symbols is designed for an independent operation of the half chips with common *ca* and clock links but separate *data* links. The communication is controlled by a dedicated master, i.e., the master will initiate reads/writes and take care of the correct half-duplex use of the bi-directional links. In the actual prototype setup, which is described Ch. 4, a programmable logic device takes the role of the master.

The interface is operated at the clock frequency given by *clkin*. Since data is arriving at double the rate, it is aligned to double words at the single speed rate. The upper word is transferred first with the rising clock edge, the lower word follows with the falling edge. Accordingly, data transferred via the *data<7:0>* links results in a 16 bit data symbol, in the following called *DATA<15:0>*; data transferred via *ca<2:0>* forms a 6 bit command-address symbol, *CA<5:0>*. In the HAGEN prototype design the convention is followed that larger signal line numbers correspond with the more significant bits.

Commands, addresses, and data are not fully separated in terms of bits or links. Rather, an encoding is chosen that allows an independent operation of half chips sharing the same *ca* links. Primarily, commands are encoded in the upper three and addresses in the lower three bits of *CA*. Nevertheless, *DATA* can carry data but as well addresses and commands. The latter two are stored in two 16 bit memories. A complete command description and the definition of the bits in these memories is given in Sec. B.3 of the appendix.

Synapse Programming As described earlier, a half chip has eight DACs that can in parallel convert digital weight values and store them into the analog weight storage units attached to each network block. The eight 12 bit weight values are transported sequentially with the *DATA<15:0>* symbols; the address of the DAC is transferred in the lower three bits of *CA*. Once eights values have been transferred, all eight DACs start their conversion in parallel. Due to the double-buffered input registers of the DACs, the respective time can be used to transfer the next eight weight values. Once all 64 analog weight storage units are programmed, the weight values for the next column of the other network block are written. The time required to do this is used by the analog weight storage units to transfer the weights to the synapse array. Fig. 4.14 in Ch. 4 shows the respective state diagram.

Input Data Data to the input neurons is written in chunks of 16 bit. The three address bits in the lower half of the *CA* symbol 'write neuron' therefore are sufficient to address all 128 inputs of one block. A state bit in one of the 16 bit registers determines which of the two blocks of a half chip is written.

Network Cycle As described in Sec. 3.3.1, the signals *ceval*, $\overline{\mathit{nclear}}$, *neval* / $\overline{\mathit{neval}}$ and *nread* are required for a half chip to perform a network cycle. Since these signals originate at the master and are transferred over the interface, the fastest rate at which these signals can be issued is the double clock rate. This will be achieved if with each clock edge all four signals can be transferred. As previously mentioned, the high word, *DATA<15:8>*, arrives at the rising edge, the low word, *DATA<7:0>* at the falling edge. The lower four bits of both these words are used for the signal transfer, i.e., bits 0 to 3 and bits 8 to 11 of *DATA<15:0>*.

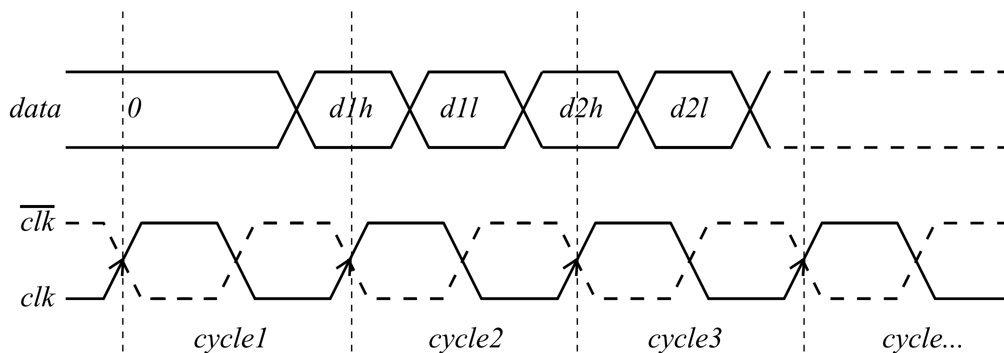


Figure 3.25: Illustration of the recovery of the clock pattern control signals.

In order to allow glitch free signals, the four signals are not sent directly, rather differences in the signal time courses are transferred. Fig. 3.25 illustrates the timing and names the data words. The actual signals are recovered by XOR-ing. The scheme is illustrated for *cycle 3*: At the rising clock edge the four bits of last clock cycle's low word, *d1l*, are XOR-ed with the bits of last clock cycle's high word, *d1h*. This can be done because the high word is pipelined and the low word not yet written. At the falling clock edge the four bits of the low word, *d2l*, are XOR-ed with the respective four bits of last clock cycle's high word, *d1h*.

Readout In order to read from the half chip, the eight data links have to change direction, i.e., the half chips becomes the sender and the master the receiver. Upon the 'read neuron' command, the half chip interface deactivates its termination (if configured for on-chip termination) and activates

its LVDS transmitters. The 3 bit address encoded in the lower word of the *CA<5:0>* symbol is sufficient to read all 128 output neurons of a twin block in 16 bit chunks. The MSB address encodes the side (on high the right side) and the remaining two continuous blocks of 16 output neurons.

The implementation of the bi-directional communication is illustrated in Sec. 4.1.2.

3.6 Power Considerations

Concerning the power consumption there are two aspects to be discussed. First, there is the power consumption inherent to the core computation of the shown current-sum, mixed-signal implementation of a neural network. It is comprised of the power consumption of the synapses, their activation by the inputs, and the evaluation by the output neuron. Due to the necessary refresh cycles, as well the power consumption of a weight update has to be considered. This is discussed in Sec. 3.6.1. The second aspect is the absolute power consumption of the HAGEN prototype implementation, which is discussed in Sec. 3.6.2. Even though the utilized LVDS transceivers are designed for low power consumption, the external as well as the internal interface are not optimized for a minimal power consumption, rather, practical aspects dominated the implementation. Once a specific low power application is aimed for, tailoring the interface to the specific needs should yield a better performance.

3.6.1 Power Consumption of a Network Block

Following [184], the power relevant for the actual computation of a network block is primarily consumed by the synapses, the inputs, and the output neurons. Further contributions come from the DACs which are continuously powered. During programming phase there is as well a contribution from the weight storage units. The current-steering approach allows to split the power consumption into a programming-dependent part caused by the synapses' weights, a signal-dependent part caused by the switching of the neurons, and a static component. For a given network block geometry (determined by the number of inputs, N_i, and the number of output neurons, N_o) the power consumption in recall phase is given by:

$$P_\mathrm{block}(N_\mathrm{i},N_\mathrm{o},\underline{\omega},f_\mathrm{net}) = P_\mathrm{weights}(N_\mathrm{i},N_\mathrm{o},\underline{\omega}) + P_\mathrm{signal}(N_\mathrm{i},N_\mathrm{o},f_\mathrm{net}) + P_\mathrm{static}(N_\mathrm{o}). \tag{3.4}$$

The contribution by the synapses is proportional to their absolute weight values and for a given weight distribution $\underline{\omega}$ constant. It is proportional to the total number of synapses, $N = N_\mathrm{i}\cdot N_\mathrm{o}$, and with $I_\mathrm{syn}^\mathrm{max}$ being the maximum synapse current and ω_j being the normalized weight programmed to the synapse j, the power consumption can be described by:

$$P_\mathrm{weights}(N_\mathrm{i},N_\mathrm{o},\underline{\omega}) = V_\mathrm{dd} I_\mathrm{syn}^\mathrm{max} \sum_{j=1}^{N_\mathrm{i}\cdot N_\mathrm{o}} |\omega_j|, \quad \omega_j \in [-1,1]. \tag{3.5}$$

Unlike the synapses itself, the driving of the evaluate switches in the synapses consumes a signal-dependent power. As it was described in Sec. 3.3.2, the synapses are activated by the *ceval* signal. In a worst case scenario, where all inputs and outputs change each cycle, the capacity of the respective line which is comprised of the capacitive contributions of the single synapses, $C_\mathrm{syn}^\mathrm{ceval}$, has to be charged to V_dd every second network cycle. Similarly, the dominating signal-dependent power consumption by the output neuron is caused by driving the *fire* signal as to allow to feed back the neuron's output:

$$P_\mathrm{signal}^\mathrm{worstcase}(N_\mathrm{i},N_\mathrm{o},f_\mathrm{net}) = N_\mathrm{i}\cdot N_\mathrm{o}\frac{1}{2}\frac{f_\mathrm{net}}{2}C_\mathrm{syn}^\mathrm{ceval}V_\mathrm{dd}^2 + N_\mathrm{o}\frac{1}{2}\frac{f_\mathrm{net}}{2}C_\mathrm{neur}^\mathrm{fire}V_\mathrm{dd}^2. \tag{3.6}$$

While $C_{\text{syn}}^{\text{ceval}}$ is given by the synapse layout and therefore generic for any network block geometry, $C_{\text{neuron}}^{\text{fire}}$ varies depending on the topology of the feedback. Since furthermore smaller voltage swings may be easily used for the *fire* signal the contribution of the output neurons may be neglected, especially, since it only scales with N_{o}. The signal-dependent power consumption therefore essentially is given by:

$$P_{\text{signal}}^{\text{worstcase}}(N_{\text{i}}, N_{\text{o}}, f_{\text{net}}) \approx \frac{1}{2} N_{\text{i}} \cdot N_{\text{o}} \frac{f_{\text{net}}}{2} C_{\text{syn}}^{\text{ceval}} V_{\text{dd}}^2. \tag{3.7}$$

The static power consumption has one contribution from the on-chip DACs which are operated with a constant current and for performance reasons are not deactivated during the recall phase. Sec. 3.4 explained that each DAC constantly sinks $8I_{\text{ref}}$ or in terms of the maximum synapse current $16I_{\text{syn}}^{\text{max}}$. On the other hand, in the current twin block setup a DAC is responsible for 8 synapse rows on each side. Its contribution per output neuron thus is only 1/16th. A second static contribution comes from the biasing of the I_+, I_-, I_{park} branches in the output neuron, and the bias current of the output neuron itself.

$$P_{\text{static}}(N_{\text{o}}) = N_{\text{o}} V_{\text{dd}} I_{\text{syn}}^{\text{max}} + N_{\text{o}} V_{\text{dd}} I_{\text{neur}}^{\text{bias}}. \tag{3.8}$$

In the setup presented in Sec. 3.3.3, $I_{\text{neur}}^{\text{bias}}$ is about the same order of magnitude as $I_{\text{syn}}^{\text{max}}$, thus, the overall static contribution to the power consumption is comparable to about two fully activated synapses. If therefore $N_{\text{i}} \gg 1$, this contribution is negligible.

The major contributions to the power consumption of a network block in recall phase thus are given by Eq. 3.5 and Eq. 3.7. With an assumed capacitance C_{eval} = 10 fF per synapse the switching power in the worst case is 1.4 μW at a maximum f_{net} = 50 MHz. This is comparable to the power consumption of the same synapse programmed to about 0.4 μA, i.e. 1% of its dynamic range. This shows that the total power consumption of the network block is predominantly given by the synapses' weight currents [184]:

$$P_{\text{block}}(N_{\text{i}}, N_{\text{o}}, \underline{\omega}, f_{\text{net}}) \approx P_{\text{weights}}(N_{\text{i}}, N_{\text{o}}, \underline{\omega}) = V_{\text{dd}} I_{\text{syn}}^{\text{max}} \sum_{j=1}^{N_{\text{i}} \cdot N_{\text{o}}} |\omega_j|, \quad \omega_j \in [-1, 1]. \tag{3.9}$$

During programming phase the contribution of the weight storage units has to be considered: According to Sec. 3.3.4 a weight is programmed by forcing the synapse to sink the respective current. Since for this I_{park} has to be charged to the appropriate gate potential, effectively this charging has to be accounted for. In a worst case weight configuration, where it has to be charged to V_{dd} every second synapse, this is given by:

$$P_{\text{prog}}^{\text{worstcase}}(N_{\text{i}}, N_{\text{o}}, f_{\text{prog}}) = N_{\text{i}} \cdot N_{\text{o}} \frac{1}{2} \frac{f_{\text{prog}}}{2} C_{\text{syn}}^{\text{Ipark}} V_{\text{dd}}^2. \tag{3.10}$$

Here, f_{prog} is the frequency with which the array is programmed and $C_{\text{syn}}^{\text{Ipark}}$ the capacitance of the I_{park} branch per synapse. With an assumed $C_{\text{syn}}^{\text{Ipark}}$ = 10 fF and a minimum programming time of 320 ns per column the worst case programming power consumption yields about 0.08μW and is no relevant contribution.

3.6.2 The HAGEN Prototype

In the preceeding section it was derived that the power consumption of a network block is dominated by the actually programmed weights and basically constant for a given weight configuration.

Since the HAGEN prototype is comprised of two half chips each consisting of two network blocks, the bus interface, and the external interface, the net power consumption of the prototype therefore has an offset contribution which is necessary to operate it but arbitrary since it has not especially been designed for power efficiency:

$$P_{\mathrm{HAGEN}} = \sum_{i=1}^{4} P^{i}_{\mathrm{weights}}(128, 64, \underline{\omega}) + P_{\mathrm{businterface}} + P_{\mathrm{IOpads}}. \tag{3.11}$$

Electrically, the three contributions are drawn from different power sources. P^{i}_{weights} is provided by a supply dedicated for the analog circuitry, V_{dda}; it is shared between the two half chips. In order to prevent the ASIC from destruction by overheating as it could occur due to the weight drift towards larger weights (c.f. Sec. 3.3.4), the supply wires within each block are sized as to prevent this from happening: Essentially, the branches I_+, I_-, and I_{park} for each output neuron are laid out such that their finite resistances prohibit the synapses of an output neuron from being simultaneously set to maximum. The power consumption thus should always stay below the limit given by the maximum current allowed by the two power pads used, i.e., smaller than about 0.7 W.

Separate from the analog supply and separate for each half there is V_{ddi} from which primarily the internal bus interface is powered. Similarly, V_{ddo} supplies the output electronics such as the LVDS line drivers. The different power nets are interconnected by diodes for ESD protection. The bonding diagram Fig. B.2 in appendix B illustrates the various power pads. While the power consumption of the external interface can be obtained from simulation to be about 0.4 W, the power consumption of the digital circuits is expected to be of similar oder of magnitude.

3.7 Beyond the HAGEN Prototype

The HAGEN prototype was designed to explore the feasibility of an efficient and scalable hardware-implemented neural network. Within the neural network experiment it has been possible to evaluate its technical properties [86] as well as to use it to explore a variety of training approaches and possible applications, e.g., [87, 57, 194, 195]. The diversity of the applications underline the flexibility arising from the network block structure and the modularity of the embedding framework. The latter will be described in detail in the next chapter. The aim of this section is to explore the implications for possible future ASIC implementations succeeding the HAGEN prototype.

A successor implementation is desirable if certain applications are targeted at. For example, the object recognition in a visual scene presented as the application in [57] not only has a demand for more network blocks but also for more inputs per network block; the latter cannot be accommodated by the prototype implementation. Similar demands arise for other application, especially if several multi-valued inputs are to be adapted by the variable network resources presented in Sec. 1.2.2 and results in Sec. D of the appendix. If a mobile application is targeted at, the integration of appropriate weight storage and training strategies has to be considered. A small example may illustrate the suitability of the HAGEN architecture for a mobile system: A coarse comparison with an Intel Pentium IV (using a 0.09 μm process technology and being operated at 3.6 GHz) shows that if the full parallelism of the ANN ASIC can be used, even a single network block of the prototype outperforms the state-of-the-art microprocessor[8] by more than an order of magnitude:

- HAGEN network block: 8192 synapses $\cdot$ 50 MHz = $4.1 \cdot 10^{11}$ CPS
- Pentium IV: 4 float additions $\cdot$ 3.6 GHz = $1.44 \cdot 10^{10}$ CPS

[8] For this it is assumed that the Pentium IV can perform an optimal of four floating-point instructions per clock cycle with its streamed SIMD unit (SSE). Necessary load/store operations etc. are neglected.

This comparison, however, assumes that the microprocessor is used to simulate the operational principle of the HAGEN prototype and accepts that the 32 bit precision floats are not the most efficient way to represent the 11 bit weights. Yet, considering furthermore the power consumption of the HAGEN ASIC and the Pentium IV (and neglecting the power consumption of the surrounding systems), it can be seen that this computing performance is achieved at a power consumption which is about two orders of magnitude smaller for the HAGEN network block.

The goal of this section is to discuss how the presented concept can be tailored to specific demands. First, minor modifications are mentioned that should be incorporated by any successor implementation; it is followed by an exploration of the design space for a version scaled in the number of network blocks or in the dimensioning of the network blocks. Finally, an outlook is given which considers an implementation in a newer process technology.

3.7.1 Minor Modifications

In order to allow a freely programmable bias current in the current-to-voltage converter stage of the output neuron, reference voltage V_{bias} should be decoupled from the input stage of the comparator as already mentioned in Sec. 3.3.3.

Other desirable changes arise from the bi-directional interface. In the prototype setup the direction switching proved to be quite difficult to implement in a real scenario (c.f. Sec. 4.1.2). The main drawback is that the master has to prevent both sides from simultaneously driving the LVDS links, yet, it has to minimize the time the links are floating. This timing is not only dependent on the frequency the interface is operated at but also on the signal propagation time over the transmission lines on the PCB. As a result, the operation frequency cannot easily be changed but rather needs fine tuning.

The situation of the *clkout* link is different but not less complicated: Although it is a uni-directional link, the driver is only active upon read commands; in the meantime the transmission line is floating. Since it may take several tens of nanoseconds to have the signal levels conform with the LVDS standard (depending on the drift of course), this would seriously slow down the operation of the ASIC. The solution in the prototype setup of Ch. 4 uses a driver at the master's side to intermittently drive the link. The activation of this 'keeper' is again dependent on the interface frequency and the signal propagation time.

While these problems can be overcome, it seems desirable to closely examine the required signals of a future implementation and—most importantly—provide uni-directional links in both directions and a continuous *clkout*[9]. This not only simplifies the operation but also speeds up the transfer. Furthermore, it allows to comply with the physical specification of the HyperTransport technology [39] and thus eases interconnection to off-the-shelf components.

A second interface difficulty arises from the source synchronous design. To reach the desired interface rates it is necessary to closely control the propagation delays of all links on the PCB level. Additionally, a deterministic output timing is required on the master's side[10]. Even though board-level simulations allow to predict propagation delays (c.f Sec. 4.1.5), the system design would be simplified if a programmable delay was integrated into the LVDS pads.

3.7.2 Scaling

Replicating the Twin Block Structure The employed AMS CSI process technology allows reticle size of up to 20 by 20 mm^2. By simply replicating the twin block structure across an ASIC

[9]This may require an additional link to indicate valid data.

[10]This may require to strictly constrain the place & route step in the programmable logic design.

of that dimensions easily synapse counts in the mega-synapse range are possible, e.g., 50 twin blocks require only 165 mm^2. Of course, several questions arise:

- **power consumption:** The considerations of Sec. 3.6 for one network block can be generalized. Since with the reuse of the twin blocks the power density is not increased, there is no limitation to how many blocks can be operated. Especially, since only two standard power pads are required per twin block, the number of power pads is of no concern. On the other hand, the power consumption can be easily controlled by programming zero weights which effectively reduces the power consumption of the corresponding block to zero. Therefore, multiplying the number of twin blocks does not make the concept inefficient in terms of power consumption.

- **yield:** While the yield for the HAGEN prototype is near 100 %, there are faults to be expected for larger die sizes. This can result in one or more defect twin blocks on a large ASIC. But provided a flexible routing scheme for the network data, it is easily possible to deactivate single twin blocks without affecting the overall functionality. Yield therefore is of no concern for a large ASIC.

- **routing:** In order to minimize the required logic, the HAGEN prototype implementation uses fixed connections for the inter- and intra-block feedback. This allows to feed back the outputs immediately to the inputs and allows to use them within the next network cycle, but it simultaneously limits the flexibility for routing in between network blocks. Most importantly, this type of direct wiring cannot be realized globally for many network blocks. Therefore, a reasonable amount of silicon area of a large ASIC has to be dedicated to a special routing network which allows to transport input and output data (and weight data) isochronously and flexibly. The first requirement is necessary for a consistent operation of several network blocks (c.f. Sec. 1.2.2), the latter for the realization of topologies which may arise from application or non-operational twin blocks.

  A fully programmable routing network, which allows the distribution to of neuron data while providing appropriate delays and buffering, is currently developed in software and programmable logic to synchronously operate several HAGEN prototypes on a parallel platform (c.f. Sec. 4.1.5). Ideas from that may be reused to implement routing blocks in digital logic that are spread across the ASIC (e.g., one per twin block) and are only interconnected with their nearest-neighbors. Especially, this reduced routing requirement make it feasible for an implementation on the ASIC.

Network Block Dimensioning Changing the number of inputs or outputs of the network blocks requires more design effort than the mere replication of twin blocks. Nonetheless, it is possible if some aspects are considered:

Extending the number of output neurons in a network block is easily achieved: There are only a few implication for the design: First of all, it will be necessary to provide programming capabilities. This can be done by adding more DACs to the twin block. A positive side effect of this is that the programming time for one column stays constant and no timing changes are necessary. On the other hand, adding more DACs and output neurons has consequences on the amount of data that needs to be transferred. More precisely, the number of entities to be addressed grows which will make a new addressing scheme necessary. An effect that ultimately limits the number of output neurons is the driving of the *ceval* signal. This is done columnwise and with each connected synapse the capacitance grows. Accordingly, the driver strength has to be adapted.

Extending the number of inputs requires more considerations. Set aside the addressing issues, the capacitive loads of the I_+, I_- and I_{park} branch are the primary concern. Adding more inputs increases them and will force other design variables of the analog implementation to vary (c.f. Sec. 3.3.3). Yet, there is no principle caveat. At a similar accuracy the evaluation time needs to be extended. If a similar maximum frequency is to be reached, the LSB current needs to be increased effectively reducing the dynamic range of the neuron and thus the resolution. As explained earlier, these adjustments can be performed during operation. Lastly, if the output neurons are used unchanged, the dynamic range of the output neurons limits the postsynaptic activity that can be evaluated correctly. Many simultaneous inputs therefore can only be used if small weights are programmed.

Data Encoding Whether more twinblocks are used or the network blocks themselves are enlarged, the result is an increased amount of data that needs to be transferred over the interface. Especially, the weight data is of concern because it scales not only with the number of blocks but also with the number of inputs *times* the number of outputs. If a network block, on the other hand, is used to implement multiple layers of a network, a certain amount of synapses will have to be set to zero (c.f. Sec. 1.2.2). Similarly, if sparse coding is used a large amount of synapses will be zero as well. Unfortunately, in the current implementation also these zeros have to be transferred across the interface. The standard approach to compress redundant data is to use a lossless data compression algorithm, e.g., Huffmann coding [98] or LZW [238]. But if only zero weights are to be suppressed, a much simpler approach can be adopted: A small logic can be employed to observe consecutive write commands to the input registers of different DACs. Once a DAC is left out, its input register automatically is set to zero.

3.7.3 Outlook

A new process technology with smaller feature sizes, such as UMC 0.18 μm Mixed-Mode/RFCMOS technology [213], is of immediate advantage only for the digital parts of the ASIC which can be smaller and faster. Yet, for the prototype HAGEN there is no gain since the current interface already uses only about 3% of the silicon area occupied by a twin block. On the other hand, if the proposed routing extensions are to be implemented, the area and speed advantages can be used. Especially, the routing resources provided by contemporary processes, e.g., six metal layers in case of the UMC technology, would allow a highly interconnected routing network capable of transporting network data between distanced network block within one network cycle.

Nevertheless, in order to use a new technology for routing and a better interface, the remaining parts of the ANN ASIC have to be ported as well. Since the twin block as well as the LVDS interface pads are full-custom circuits, the effort for adapting the HAGEN prototype circuits is substantial. Indeed, trying to port the circuit schematics without major changes can only succeed, if an analog option in the process allows to maintain the supply voltage of 3.3 V and provides transistors with acceptable leakage currents. The UMC technology, for example, has a nominal voltage of 1.8 V but has an option for 3.3 V transistors with better analog properties. Since this basically keeps the transistor sizing unchanged (at least for the critical transistors), there will be no immediate area gain.

Some minor modifications to the circuits on the other hand may yield some interesting improvements: If for the weight storage in the synapses (S3, see Sec. 3.3.2) instead of a gate capacitance a metal-in-metal capacitance (MIM-CAP) is used, the synapses may be reduced in size by roughly a factor of 2. The achievable reduction in area is primarily limited by the sheet capacitance of the MIM-CAP, which is about 1 fF/μm^2; thus for a 60 fF capacitance about 60 μm^2 are

required. Of course, only a layout and parasitic simulation allows to estimate whether the remaining circuitry an be fitted into this smaller area; essentially, S3 and the diode S2 are spared. The diode can be relinquished because the MIM-CAP does not need is potential difference to be larger than a specific threshold voltage; yet the decreased output resistance has to be closely examined.

In conclusion, the proposed circuitry allows to be easily scaled towards the mega-synapse realm on the currently used substrate, the 0.35μm AMS technology. This can be done either way: by using multiple small twin-blocks and routing logic or by fewer twinblocks of larger dimension. The advantage is that the elementary analog circuits are well tested which minimizes the necessary preparations and precautions for producing a large ASIC. The porting of the design to a new process technology does not seem justified since an area advantage of at most a factor of 2 is expected (for the UMC 0.18μm technology). Finally, additional iterations may be necessary to guarantee the analog performance. On the other hand, these conclusions are based on practical considerations such as the design effort. Since the underlying architectural concepts of the HAGEN prototype proved quite successful, the effort for adopting those principles with future substrates will be worthwhile.

Chapter 4

Experimental Framework

Embedding the ANN ASIC in a flexible system comprised of modular electronics, programmable logic, and a microprocessor allows the combination of the advantages of hardware with the versatility of a pure software realization. This chapter will illustrate the modularity of the framework on the substrate as well as on the functional level. This separation is the basis for readily integrating different ANN ASICs or to migrate the experimental framework to a highly integrated PCB. Furthermore, extensions to the software are presented that can be used to adapt the neural network experiment to specific needs arising from the question under investigation. Special emphasis is put on the realization of liquid computing with the HAGEN prototype.

4.1 Substrates

4.1.1 ANN ASICs

So far two prototype ANN ASICs have been developed as described in Ch. 3. Even though the physical layer and the link layer of the interface of the EvoOpt prototype are quite different from the ones employed in the HAGEN prototype (c.f. Sec. 3.5)—and may be different for future implementations—the higher levels of the interaction with the ASICs are very similar which is due to the network block organization. Especially, the operation sequence is similar and independent of the actual analog implementation: First, a two-dimensional[1] array of weights is configured. Then a fixed-size vector of input data is applied, and in one or more network cycles the product of the input vector and the weight matrix is calculated. Regarding the logical operation of the framework this step is independent from the actual electronic realization; of course, technical aspects do vary in between different ANN ASICs, such as the accuracy, speed, or actual dimensions. The outcome, finally, is read as a fixed-size vector of output data.

The modular structure of the experimental framework allows to exchange parts of it while reusing others. If, for example, the ANN ASIC is exchanged, primarily the parts directly interacting with the hardware of the ASIC are to be adapted while most parts of the framework can be reused. Parts to be adapted are: The PCB providing the immediate electrical environment, the chip

[1]The actual representation in memory is linear.

carrier PCB. Furthermore, the entity of the programmable logic design directly interacting with the ASIC obviously has to be adapted to the used ANN ASIC. Similarly, a software representation of the specific capabilities (e.g. the number of inputs) will be necessary[2].

On the other hand, all parts of the programmable logic design dealing with a more abstract representation of the ASIC, i.e., the network block structure, remain unchanged. Especially, the management of input, output and weight data does not have to be adapted. Similarly, the interface to the host PC, and thus the interaction with the software, is mainly reusable. Supporting entities, such as AD/DA conversion capabilities, power supplies, memories and their interfacing in programmable logic are completely unaffected by changes in the ASIC. Most importantly, most parts of the software, especially, training strategies that represent a major part of the scientific work can seamlessly be used. For these reasons, it is expected, that future ANN ASICs of similar type can be almost immediately be integrated into the experimental framework.

4.1.2 Dedicated Peripheral Electronics

Since the PCI-based mixed-signal FPGA adapter described in the next section has some generic features, such as different power supplies, AD/DA conversion capabilities, and a configurable digital interface, the electronics that are specific for a certain ANN ASIC are mainly reduced to mechanical adapters and passive electronics. Their primary task is to provide a carrier for the ASIC and a hot-pluggable connector to the other parts of the framework hosted in a general purpose PC. This is realized by a split approach: One PCB is the actual chip carrier; it hosts the ASIC and is hot-pluggable. The other PCB is an extension to the mixed-signal FPGA adapter and basically translates its generic interface into the specific hot-plug interface of the chip carrier PCB.

To illustrate the modularity, these components are described for the EvoOpt prototype as well as for the HAGEN prototype implementation.

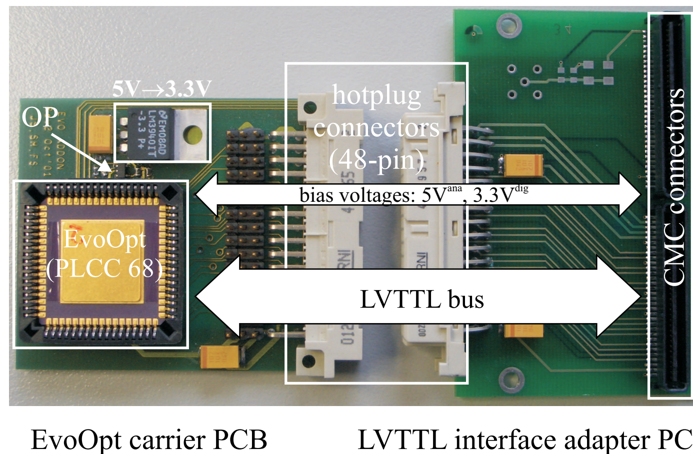


Figure 4.1: Photograph of the EvoOpt carrier PCB and the LVTTL interface adapter PCB.

EvoOpt Prototype The single network block ANN ASIC based on the charge-sharing implementation, the EvoOpt ASIC (see Sec. 3.1 and [186]), is hosted in a 68-pin PLCC package. Built in the same 0.35 μm technology as the HAGEN prototype, it requires a supply voltage of 3.3 V, separate for the analog core and the digital interface. The physical layer of its interface is realized

[2]Due to the employed object-oriented approach this is straight forward as can be seen in Sec. 4.3.

by 16 bi-directional, and 20 uni-directional single-ended LVTTL links. Besides 4 constant bias voltages, a fast varying voltage input is needed for the analog programming of the weights (c.f. Sec. 3.1).

Fig. 4.1 illustrates an augmented photograph of the chip carrier PCB and the interface PCB. The passive interface PCB provides a rewiring of the generic common mezzanine card (CMC) connectors [143] of the mixed-signal FPGA adapter (c.f. Sec. 4.1.3) to a 48-pin hot-plug connector to the carrier PCB. It gets mounted on the mixed-signal FPGA adapter and remains in the PC. A pin reference is given in [18].

The carrier PCB contains only a few active components: First, the analog supply voltage is generated by a voltage regulator from the 5 V provided by the interface. Second, the fast varying voltage input is buffered. While the analog value across the interface is driven by a current against a parallel termination, it is converted to a voltage by a non-inverting operational amplifier.

HAGEN Prototype The HAGEN prototype of Ch. 3 is hosted in a 144-pin PGA package. Besides the three static bias voltages, the two static reference currents, and a few configuration pins it only has digital IO. As described in Sec. 3.5, the HAGEN interface uses differential links according to the LVDS standard that are operated synchronously at frequencies of up to 300 MHz. There are 16 bi-directional, and 5 uni-directional LVDS links. Furthermore, there are 28 uni-directional single-ended LVTTL links for the direct-in and direct-out (c.f. Sec. 3.2). While the single-ended links are not critical due to maximum frequencies of 50 MHz, the LVDS links operated at up to 300 MHz have sub-nanosecond rise times. Multiple mechanical connectors used to ensure the modularity of the framework require to control the signal integrity. Especially, the impedance of the differential traces and across connectors of LVDS links need to be controlled.

In order to accommodate impedance-controlled structures, a 4-layer PCB process has been used as to comply with guidelines published by National Semiconductor [76]. A 4-layer PCB allows to shield the differential signal traces by equipotential planes from single-ended signals or from other signal layers. Fig. 4.2 illustrates a trace configuration that is called microstrip line. It shows the upper two (lower two respectively) layers of a PCB; ε_r is the dielectric constant of the insulator material (usually, FR4), above the microstrip line is air which is assumed to have a different dielectric constant of 1.

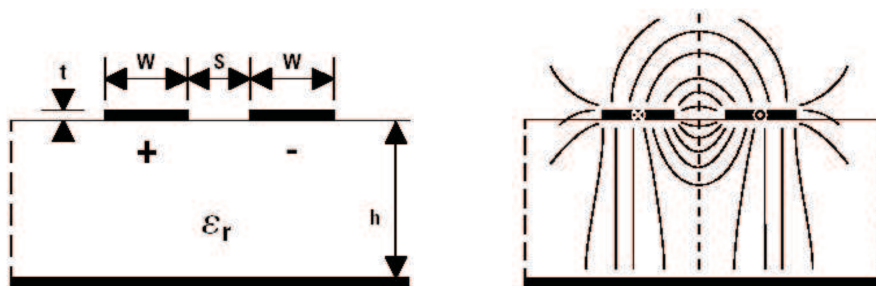


Figure 4.2: Left: Schematic cross section of a microstrip trace configuration. **Right:** Ideal electric field configuration for a microstrip trace. Both figures are taken from [76].

The differential impedance of this configuration is derived from the unit length capacity and impedance of the single traces to an equipotential plane and the coupling in between them under the assumption that the signals in the two traces are equal in magnitude but opposite in polarity. The right side of Fig. 4.2 shows the ideal electric field for the microstrip configuration. Due to the strong dependency of the impedance on the geometry, e.g., edge-effects, neighboring structures,

and layer structure, the impedance is inferred numerically rather than analytically for a given setup. This is done by dissecting a circuit layout into separate cross sections, for which a field solver calculates the charge distribution (e.g. [165]). From there the RLGC matrices per unit length of the four-terminal network model [85] can be computed. Another way for calculating the impedance are approximate equations published by National Semiconductor [136, 76]. These equations are derived from empirical data. According to [30] the results for the differential impedance are consistent with those of the field solver approach within an error of 10 %. Since this is of similar magnitude as the variations that are to be expected due to manufacturing, this is acceptable.

Accordingly, the impedance of a single microstrip trace is given by (using the dimension measures of Fig. 4.2):

$$Z_0 = \frac{60}{\sqrt{0.475\varepsilon_r + 0.67}} \ln\left(\frac{4h}{0.67(0.8W + t)}\right)\Omega, \tag{4.1}$$

and the differential impedance by:

$$Z_{\mathrm{diff}} \approx 2Z_0\left((1 - 0.48e^{-0.96\frac{S}{h}}\right). \tag{4.2}$$

The closer the two traces are, i.e., the smaller the distance S gets, the better the magnetic fields are canceled which reduces the radiated noise. Additionally, close traces make it more likely that external noise affects both traces and occurs as a common-mode shift which is rejected by the receiver. Therefore, all LVDS links on the HAGEN carrier PCB and the LVDS interface PCB have been designed according to a minimum S allowed by the PCB manufacturer and to the desired 100 Ω differential impedance[3].

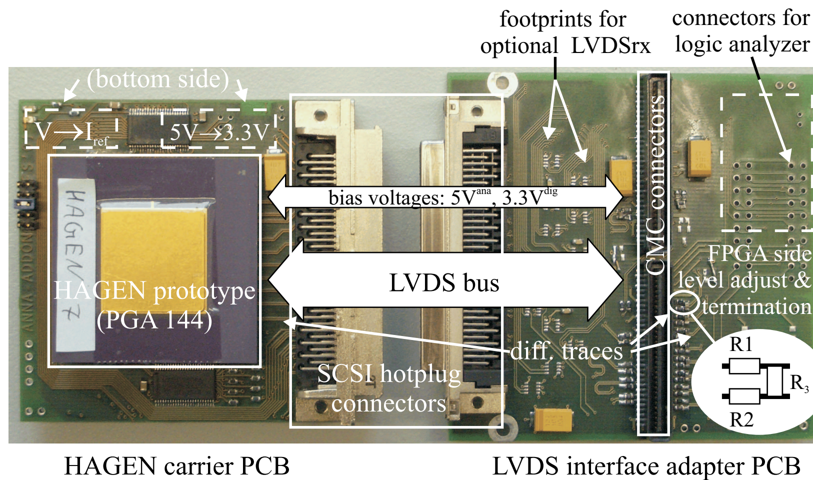
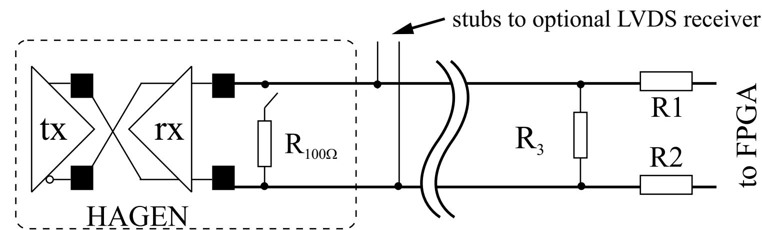


Figure 4.3: Photograph of the HAGEN carrier PCB and the LVDS interface adapter PCB.

Fig. 4.3 shows an augmented photograph of the HAGEN carrier PCB and the LVDS interface PCB. The interface PCB routes the 27 differential pairs from the CMC connectors of the mixed-signal FPGA adapter to a 68-bit hot-plug connector. This connector is mechanically compatible with the SCSI Parallel Interface 2 connector type P [162]. Electrically, the dedicated pairing of pins has been followed, but the actual pin configuration is proprietary. A pin reference is given in Tab. C.1 of the appendix.

[3]Since all remaining parameters such as dielectric constant ε_r, insulator width h, and trace height t are determined by the manufacturer, Eq. 4.1 and Eq. 4.2 can be solved for the only remaining free parameter: W, the width of each trace.

The traces of the interface board are laid out such that all data links have similar lengths from the FPGA to the SCSI-connector. The links for *clkin* and *clkout* have minimum length in order to make the clocks faster in relation to the data[4]. The employed Virtex-E FPGA [233] on the mixed-signal FPGA adapter allows to use each one of the two paired wires individually in a single-ended configuration, or as a differential input or differential output, and even bi-directional. This is configured by the programming of the FPGA and appropriate resistances which have to be mounted on the interface PCB. Those are necessary since the FPGA only provides pseudo-differential signals. Fig. 4.4 shows the possible configurations. In order to make the differential interface analyzable by non-differential logic analyzers, furthermore standard LVDS receivers can be mounted to listen at the bus and to convert the differential signals to single-ended ones.



Bi-directional Configuration: $R_3 = 100\ \Omega$, $R_1 = R_2 = 120\ \Omega$

Uni-directional Configuration:

HAGEN tx, FPGA rx: $R_3 = 100\ \Omega$, $R_1 = R_2 = 0\ \Omega$

HAGEN rx, FPGA tx: $R_3 = 100\ \Omega$, $R_1 = R_2 = 120\ \Omega$

Figure 4.4: Schematic illustration of single LVDS link between the HAGEN prototype and the FPGA adapter. According to [233] the pseudo-differential outputs of the Virtex E need the shown resistor configuration ($R_1 = R_2 = 165\Omega$, $R_3 = 140\Omega$) to generate LVDS logic levels. Since in a bi-directional setup R_3 is the termination resistance for the sender in the HAGEN prototype, the resistances have been rescaled to $R_3 = 100\Omega$, $R_1 = R_2 = 120\Omega$ which slightly increases the differential output voltage.

Fig. 4.5 shows an analysis of the bus between the HAGEN prototype and the FPGA. The top part of the figure shows a partial screen shot of a HP 16712A timing and state analyzer module in a HP16700A logic analyzer [2] connected to the optional LVDS receivers. Shown are the clock signals, the two 8 bit *data* links to both HAGEN halves (D1, D2), and the 3 bit *ca* links (ADDR) (see Sec. 3.5.2). Due to the restricted sampling rate of 1 sample every 4 ns per channel, the time resolution is quite coarse. The lower part of the figure shows a simultaneously recorded oscilloscope shot of the extra listed 7th bit of D1 recorded across the resistance R_3 with a differential probe (P7330 on a Tektronix TDS784D [211, 210]). The marker G1 indicates the same timing in both plots. The shown analog signal of a data link shows direction changes of the bi-directional link. About 6 ns after G1, the FPGA deactivates its sender and the differential voltage across the termination resistance fades to zero[5]. About 15 ns after the deactivation, the HAGEN prototype

[4]In practice it is necessary to further slow down each of the data links by a 3.3 pF capacitor between paired traces, c.f. Sec. 3.5.

[5]The time between the deactivation and activation of the respective senders should not be too long since otherwise the common mode may drift. For the bi-directional data links this is of no concern. Yet, the uni-directional clock *clkout* is floating for longer periods. Therefore, an additional LVDS sender of the FPGA is connected to the *clkout* link in bus configuration and maintains the logic level in periods no reads are performed.

activates its transmitter and sends data. The different logic levels between the FPGA sender and the HAGEN LVDS transmitter result from different driving strengths.

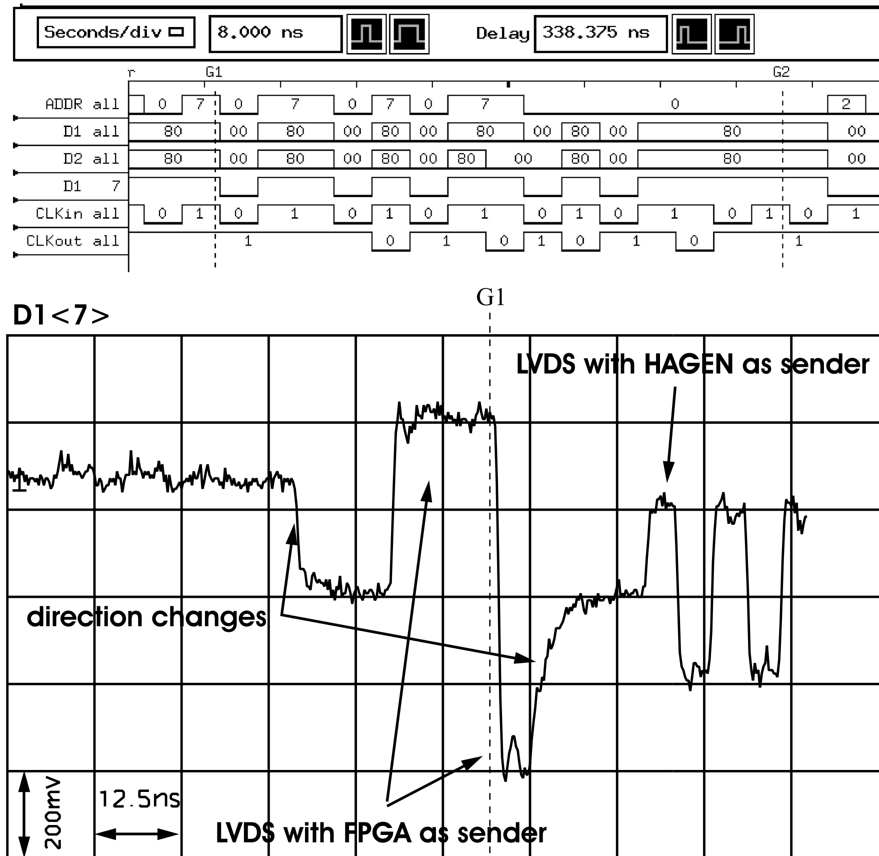


Figure 4.5: Screen shot of an HP 16712A timing and state analyzer (top) recording the bus between the HAGEN prototype and the FPGA. It observes the bus with the help of additional LVDS receivers tapped to each differential link. The link D1<7> is plotted extra and a simultaneous oscilloscope recording is shown in the bottom. The signal is recorded with a differential probe across the resistor R_3 on the LVDS interface PCB. The marker G1 indicates the same time in both plots.

Six of the available differential links on the interface PCB are configured for single-ended operation. In order to do this the resistance R_3 must not be mounted on the interface PCB. These 12 wires can be multiplexed by two buffers on the HAGEN carrier PCB and are used for direct-I/O.

Besides the appropriate LVDS traces, the carrier PCB provides a few active components: These are a voltage regulator to generate the 3.3 V analog supply voltage and a discrete circuit to generate the further required two reference currents from a single[6] reference voltage. Fig. 4.6 shows a discrete NMOS transistor which is configured in a constant current source configuration. The negative feedback to the inverting input of the operational amplifier adjusts the gate-source voltage such that the voltage drop across R_3, U_3, equals the voltage drop across R_2, U_2. Determined

[6]The two reference currents do not need to be individually controlled; merely, each half of the HAGEN prototype requires its own reference current, and sharing a twofold current would strongly rely on the identical resistance of the current inputs.

by the voltage divider comprised of R_1 and R_2, U_2 is derived from U_{iref} according to:

$$U_2 = \frac{R_2}{R_1 + R_2} U_{\text{iref}}. \tag{4.3}$$

Since $U_2 = U_3$, the resulting current I_{ref} is determined by R_3 according to Ohm's law and can be calculated. In the current implementation, the ratio R_1/R_2 is chosen to be 7/3 with R_1, R_2 in the kΩ range which causes U_2 to be $0.3U_{\text{iref}}$. With $R_3 = 10\text{k}\Omega$ and a voltage range of the DAC from 0 V to 3.3 V therefore I_{ref} can be adjusted between 0 A and about 100 μA according to:

$$I_{\text{ref}} = 3 \cdot 10^{-5} \frac{1}{\Omega} U_{\text{iref}}. \tag{4.4}$$

As it was described in Sec. 3.4, the on-chip DACs of the HAGEN prototype have a maximum output current swing of eight times I_{ref}. Due to the 16-fold reduction in the weight storage unit, the maximum synapse current results to:

$$I_{\text{syn}}^{\text{max}} = \frac{1}{2} I_{\text{ref}} = 1.5 \cdot 10^{-5} \frac{1}{\Omega} U_{\text{iref}}. \tag{4.5}$$

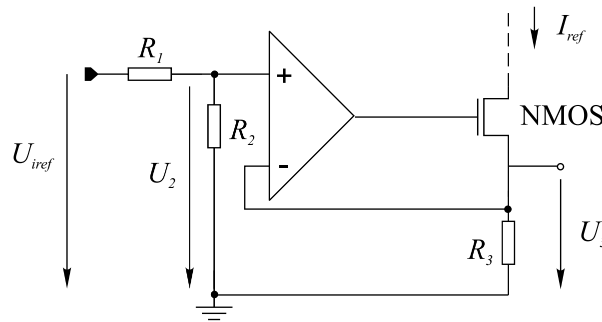


Figure 4.6: Controlled constant current source on the HAGEN carrier PCB for generating I_{ref}.

4.1.3 PCI-based Mixed-Signal FPGA Adapter

A central component to the experimental framework is the programmable logic. It allows to interface and operate the actual ANN ASIC and to implement certain tasks in dedicated logic circuits. Furthermore, it can provide the interface to a general purpose computer which runs software and thus manifests the flexibility of the framework. Fig. 4.7 shows a simplified block diagram of a custom-built mixed-signal FPGA adapter that is employed as the substrate for this functionality. It has been developed as a generic test system for mixed-signal ASICs [18] which interconnects to a standard IBM compatible computer via a PCI connector. Its technical specifications are listed in Tab. 4.1. The initial startup, the experimental test, and the general availability of these adapters as a research tool in the Electronic Vision(s) group fell into the responsibility of this thesis. 21 of these adapters have been built and are in use as a common basis in different types of experiments that involve ASICs: The here presented neural networks, evolvable hardware [117], and optical sensors with logarithmic response [28]. Fig. 4.8 shows a photograph of the adapter with the components highlighted.

The test system is built around a commercially available FPGA. On the one hand, this allows to develop all logic in a hardware description language[7] and use the manufacturer's tools to

[7]The hardware description language used is VHDL [46].

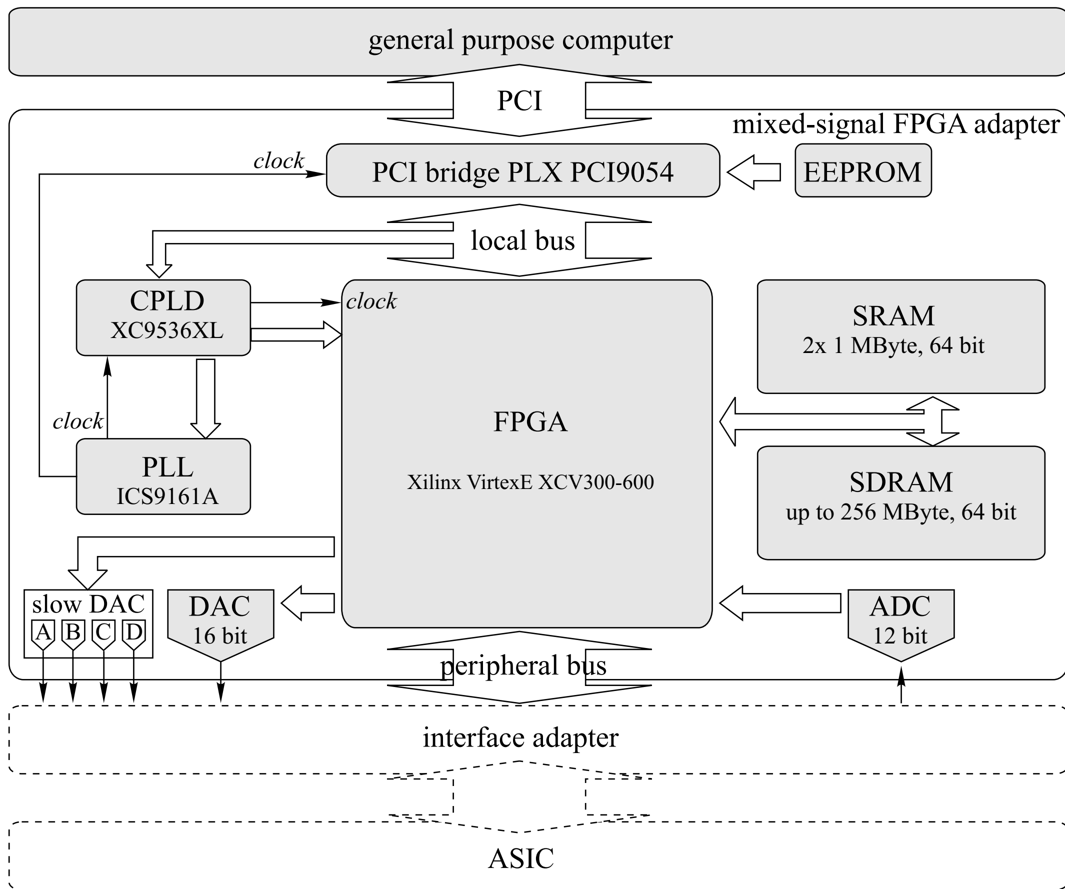


Figure 4.7: Block diagram of the discrete electronics on the PCI-based mixed-signal FPGA adapter.

synthesize a design that can be loaded into the FPGA. This way the logic in large parts remains independent of the actually employed FPGA. On the other hand, an FPGA is well-suited to *interface* the custom-built ASICs since its I/O pins can be configured for various signalling standards, e.g., LVTTL or LVDS. It furthermore allows to *operate* the ASIC's interface with a guaranteed latency and at speeds that cannot be realized by a standard peripheral bus such as the very common 32-bit PCI bus. The latter has a maximum bandwidth of 132 Mbyte/s (at 33 MHz and 32 bit) which is only 10 % of the nominal bandwidth of the HAGEN prototype ASIC[8]. For the FPGA to be able to meet the interface requirements it needs an associated local memory which can be accessed at the required rate. In conjunction with a clever partitioning of the functionality in the experimental framework (c.f. Sec. 4.2) this allows to operate the ANN ASIC faster than possible by the PCI bus alone, yet, it does only rely on commonly available components. The current implementation allows to operate the HAGEN prototype at frequencies of up to 100 MHz which yields a bandwidth of about 500 Mbyte/s. This is about 4 times the theoretical maximum PCI speed[9] and about one third of the theoretical interface speed. It is primarily limited by clock/data timing which is

[8]Recent extensions and additions to the PCI interface such as PCI-X and AGP8X or completely new interfaces such as PCI-Express are launched to overcome the old limitations and provide bandwidths of a few Gbyte/s. While those can provide the required data transfer rates between a PC and the ASIC, their availability in off-the-shelf desktop hardware is still very limited.

[9]In practice, transfer speeds of about 80 to 90 Mbyte/s are reached.

PC interface	32-bit 33MHz PCI 2.1 card
power supply	on board generation of 1.8V, 2.5V, analog +5V/-5V
FPGA	Xilinx Virtex-E XCV300E-600E
memory	2MByte SRAM, 64-bit SO-SDRAM of up to 256 MByte
modular user interface	28 bidirectional LVDS links & 2 unidirectional LVDS links & 79 single-ended links or 139 single-ended links
digital to analog converter	one 16-bit, 35 MHz two 2-channel 12-bit
analog to digital converter	one 12-bit, 40MHz

Table 4.1: Technical specifications of the mixed-signal FPGA adapter.

non-optimal due to the chosen FPGA and the generic design of the adapter. Sec. 4.1.5 illustrates how this limitation can be overcome.

FPGA and PCI Interface The adapter is designed to host a Xilinx Virtex-E FPGA [233] in a 432-pin BGA package. The available types XCV300E, XCV400E, and XCV600E that differ in the number of provided logic blocks can be accommodated. They are available in different speed grades. To provide the desired flexibility, the FPGA is not configured at startup, rather, the configuration data has to be provided by the software on the general purpose computer via the PCI interface. Therefore, a stand-alone PCI bridge PLX PCI 9054 [163] has been chosen to interface the FPGA to the PCI bus. While the bridge can autonomously serve the PCI interface and provide the requests and data to its local bus, it cannot configure the FPGA. This role is taken by a non-volatile programmable logic device: Xilinx CPLD XC9536XL [235]. Upon specific requests, the CPLD configures the PLL for the clocking of the local bus with up to 50 MHz and provides clocks and configuration data to the FPGA. After its configuration the FPGA serves the local bus and operates the remaining components, such as memory, AD/DA conversion ICs, and the interface to the ANN ASIC. This functionality has to be programmed in a hardware description language (c.f. Sec. 4.2).

Local Memory There are two types of local memory to the FPGA: A static RAM (SRAM) comprised of 2 chips with 1 Mbyte each and a removable synchronous dynamic RAM (SDRAM) of up to 256 Mbyte. Both are attached to the same 64 bit wide data bus which makes their usage mutually exclusive; yet, due to dedicated select signals this can be changed during operation. The primary task is to have weights and input data readily available to operate the ANN ASIC and to provide memory for the read back network results. The memory controller in the FPGA (c.f. Sec. 4.2) allows software running on the general purpose computer to transparently read and write to this local memory by DMA burst accesses. Which memory finally gets used is dependent on the application: The SRAM has a guaranteed latency but is small. The SDRAM is much larger but the latencies vary.

Peripheral Bus The purpose of the FPGA adapter is to interconnect mixed-signal ASICs to a host PC. Within the neural network experiment so far two ASICs are to be interfaced; this was described in Sec. 4.1.2. The necessary modularity is achieved by employing an FPGA that provides enough I/O resources not only to serve the local bus and the local memory but also to serve

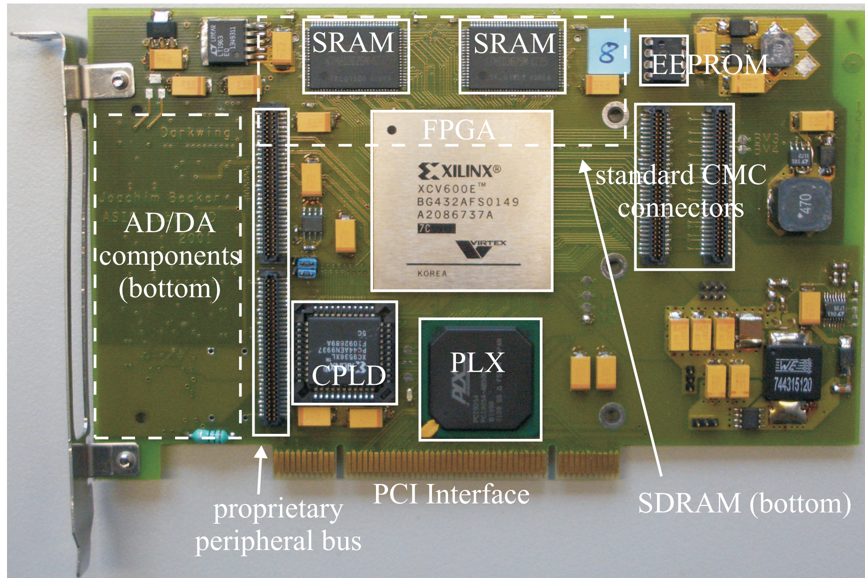


Figure 4.8: Top side photograph of the PCI-based mixed-signal FPGA adapter.

a peripheral bus and other components needed for a mixed-signal test, e.g. AD/DA components. On the one hand, 79 bi-directional LVTTL links are provided according to the CMC extension standard [143] with appropriate power supplied. This allows to interconnect to commercial CMC extensions. On the other hand, for the specific needs of mixed-signal ASICs there is an additional custom interface using the same type of mechanical connectors (CMC) but a proprietary configuration: This interface additionally provides 60 bi-directional links and input and output to the AD/DA components. Depending on the external wiring (c.f. Sec.4.1.2) pairs of these links can be used as differential LVDS links (a total of 30). In order to allow this dual use the 60 links are laid out in 30 differential traces with controlled impedance. Additional to the microstrip traces of the LVDS interface adapter, there are stripline traces used, i.e., traces run on an intermediate layer shielded by planes above and below; the appropriate formulas can be found in [76].

DA/AD Conversion Capabilities Besides the digital signals of the peripheral bus, a mixed-signal ASIC test system requires analog signals. Since the FPGA is purely digital, components are added that allow the FPGA to interface analog signals. On the one hand, there are DACs that convert a binary encoded value into a proportional analog value. In the current implementation there are two types of DACs:

- In order to generate slowly varying voltages there are two dual-channel 12-bit voltage DACs (MAX5104) with a settling time of about 12 μs. One channel each is buffered by an additional operational amplifier in voltage follower configuration for driving low-impedance loads. In the neural network experiment these signals are used to generate static bias voltages.

- For some applications it is necessary to provide analog values much faster. E.g., the weight programming in the predecessor ANN ASIC relies on an external analog voltage. Therefore, one 16-bit current DAC (AD768) is integrated on the FPGA adapter. It has a conversion time of 25 ns and an output range of -20 mA to 20 mA. The current is converted to a voltage by an operational amplifier in an inverting setup with shunt feedback and offset compensation.

With the feedback resistor the unipolar output voltage range is adjusted to 0-4 V. In the HAGEN setup this DAC is not used.

On the other hand, if analog input signals need to be connected to the FPGA, analog-to-digital conversion has to be performed. For this reason, a 12 bit ADC (ADS800) with a sampling rate of 40 MHz is mounted on the adapter. Due to the purely digital output of ANN ASICs with McCulloch-Pitts neurons, the AD conversion is not needed for the operation of the ANN ASICs. Since in the HAGEN setup the ADC as well as the fast DAC are unused, they are available for applications such as line equalization in a data transmission.

Analog Power Supply The primary power supply for the FPGA adapter is provided by the PCI interface; there, 3.3 V, 5 V, and ±12 V are available. These power supplies are shared between all components of the PC which makes them electrically noisy. Therefore, besides power supplies for special voltages needed by the digital components, extra power supplies with adequate blocking capacitors are provided for the analog components. An analog 5 V power supply feeds the DACs/ADC and can as well be used for the tested ASIC. From this the 3.3 V analog supply voltage for both ANN ASICs is derived by dedicated voltage regulators.

4.1.4 General Purpose Computer

Implementing parts of the experimental framework in software allows to easily extend and modify the training strategies and areas of application. This is important since several approaches are under research which is described in Sec. 4.4. In the current framework implementation the microprocessor executing the software is the CPU of an IBM compatible general purpose computer. This type of computer provides several PCI extension slots which allows to operate one or more of the PCI-based FPGA adapter cards simultaneously.

The choice for a specific CPU, e.g., Intel Pentium IV or AMD Athlon XP, is not of primary importance since contemporary C++ compilers [63, 101] allow to automatically optimize code for the different platforms. Yet, once a platform is targeted at, it still may be desirable to use even more optimized code. It is therefore avoided to use compiler specific extensions to the C++ standard. In the case of the Intel Pentium IV, for example, the Intel compiler [101] can be used which should be able to make more use of the specific extensions provided by the CPU.

Eventually, it is depending on the application how much the CPU of the general purpose computer actually has to do: The evolutionary coprocessor implemented in programmable logic (see Sec. 4.2.3 and [189]) is dedicated to release the general purpose CPU from most of its tasks. Still there are applications, such as the liquid computing setup described in Sec. 4.4.3, which do a lot of number crunching and therefore revert to hand-optimized platform-specific libraries[10].

Since the computational requirements vary and therefore not primarily dictate a preferred CPU architecture, another factor becomes important: the cost of operation and maintenance. The PCI-based FPGA adapters are used throughout the Electronic Vision(s) group, as are the general purpose PCs hosting the adapter. Especially, by using similar and well supported hardware (i.e., chipset, network adapters etc.) a new general purpose computer can be added to the farm of *hardware operation computers*[11] simply by cloning the hard drive image. In order to allow a consistent setup and a remote administration the freely available operating system Linux[12] is used, currently based on a 2.4.x kernel. Kick-start CDs and remote maintenance scripts have been developed during this thesis which allow to manage the farm of currently 11 hardware operation computers plus

[10]A library is a collection of functions that implement functionality which may be re-used by other programs.

[11]In reminiscence of the primarily used evolutionary algorithms (c.f. Sec. 4.4.1) these computers are called *evolvers*.

[12]Throughout this thesis SuSE distributions from version 7.2 to 9.0 have been used.

a couple of hardware test computers. The current set of hardware operation computers is based on the Intel 845PE chip set with Intel Pentium IV CPUs between 2.4 and 2.6 GHz.

Although Linux currently is the primary development and operation platform, the experimental framework is not restricted to it. For one, this is achieved by using the WinDriver 6.x product [109] by Jungo, Inc. for allowing the hardware access[13] in user mode[14] which is available for all major platforms. For the other, code precautions and platform-independent libraries described in Sec. 4.3 allow not only the exchange of compilers, e.g., the use of the Intel compiler [101] instead of the GCC [63], but also the switching of operating systems. With only minor modifications in the source code[15] therefore it should be possible to migrate the hardware operation computers to Windows NT/XP. Another application of this platform-independence is the migration to the advanced implementation of the neural network experiment described in the next section.

4.1.5 Advancing the Framework

The scaling to larger networks with the network block approach can occur on the chip level by integrating more network blocks in a new ASIC or by simply using multiple chips of the existing HAGEN prototype. In principle, this can be realized by the components described so far but using more than one mixed-signal FPGA adapter in the same general purpose computer. This would allow right away to operate several ANN ASICs simultaneously. But in order to have them form a large network, network data needs to be transported isochronously from one ANN ASIC to the other as described in Sec. 1.2.2. In an unmodified setup this activity would have to be transported across the PCI bus twice[16] which considerably slows down the network operation. Furthermore, the latency of a PCI transfer is not guaranteed.

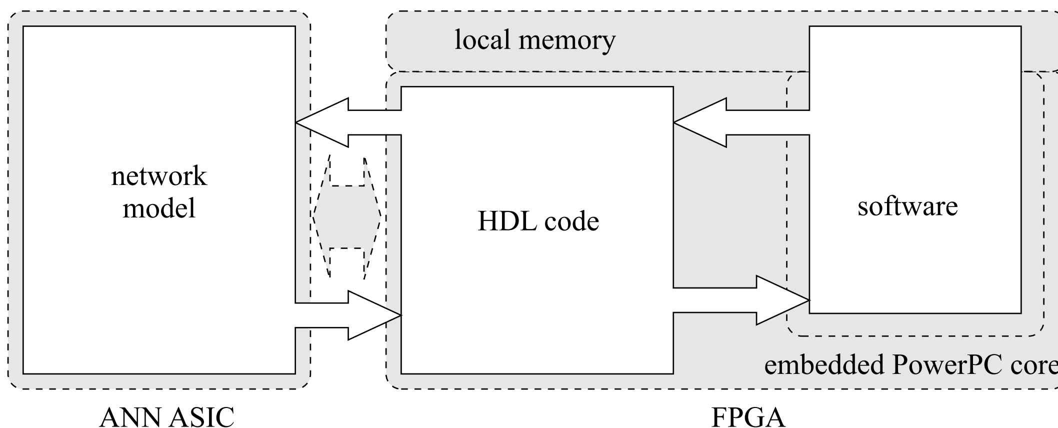


Figure 4.9: Components of the neural network experiment in an advanced realization. Compared to the implementation of Fig. 1.8 the software runs on the FPGA substrate and therefore the PCI bottleneck is avoided.

This bottleneck in the scalability of the current framework implementation can be overcome with a different approach: To be able to reuse most of the work spent on the functionality, a printed

[13]It provides an interface to the used PCI bridge PLX PCI 9054 and manages, for example, the DMA access.

[14]Multi-user operating systems implement different levels of privileges for executed processes. Especially, the hardware access is restricted to the kernel. Therefore, a dedicated device driver, or kernel module, is required to make it accessible by processes with standard privileges, i.e., processes in user mode.

[15]With 'minor modifications' bug fixes are meant which will be necessary if the code actually is tested on Windows.

[16]Or once, if the PCI bus master capabilities of the PLX PCI 9054 would be used to access the data of another adapter's local memory directly.

circuit board was built [77] that integrates all parts of the current framework, i.e., the ANN ASIC, a programmable logic, and a microprocessor to run the software but on a different substrate. This is illustrated in Fig. 4.9. Basically, the current implementation of the neural network experiment that requires a complete PC is shrunk to a PCB of the dimensions 13 by 8 cm^2 (see Fig. 4.10). This was possible by using a Xilinx Virtex-II pro [232] that has an embedded PowerPC core. On the one hand, most of the programmable logic design of Sec. 4.2 can be mapped to the logic blocks of this FPGA. On the other hand, the task of the general purpose computer is shifted to the embedded PowerPC core. Due to the precautions for platform independence in the software development (c.f. Sec. 4.3) it is possible to compile the existing code for an embedded Linux [205] on the PowerPC.

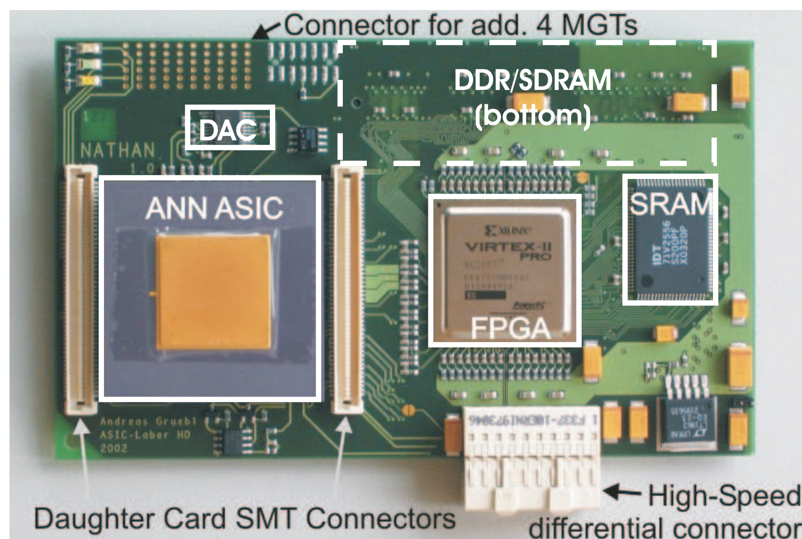


Figure 4.10: Top side photograph of the PCB that essentially implements the functionality of the PCI-based framework with a general purpose computer and the ANN ASIC. The shrunk form factor and dedicated high-speed links (MGT) allow to interconnect several of these so-called *network modules*. The bottom side hosts the DDR-SDRAM, another SRAM, and primarily routing.

The electronic design of this advanced framework implementation strongly relied on the experiences made with the PCI-based framework implementation: Each of these highly integrated, so-called *network modules* suits an FPGA with 672 pins, a double data rate SDRAM (DDR-SDRAM) with a 64 bit data bus, an SRAM memory with a 64 bit wide data bus, a synchronous LVDS interface providing up to 32 differential links plus clocks for the HAGEN prototype or future ASICs, and connectors for up to eight high-speed links that are integrated in the FPGA, so-called multi-gigabit transceivers (MGT); a block diagram is shown in Fig. 4.11. Especially, since the high-speed links with signals in the GHz range have to be accommodated between modules and thus have to go across connectors, signal integrity becomes an issue. While the trace configuration of the PCI-based FPGA adapter have been calculated separately (c.f. Sec. 4.1.2), for the design of the network module board- and system-level simulations during the layout phase have been utilized [77]. With the simulation models provided by the component manufacturers this allowed to optimize the routing and shape of the traces. Furthermore, the employed tools allowed to interactively ensure equal trace lengths for critical busses such as the DDR-SDRAM memory bus and the peripheral LVDS bus for the ASICs. In order to realize the necessary shielding and provide enough routing resources an eight-layer PCB technology with special MicroVias has been used [231].

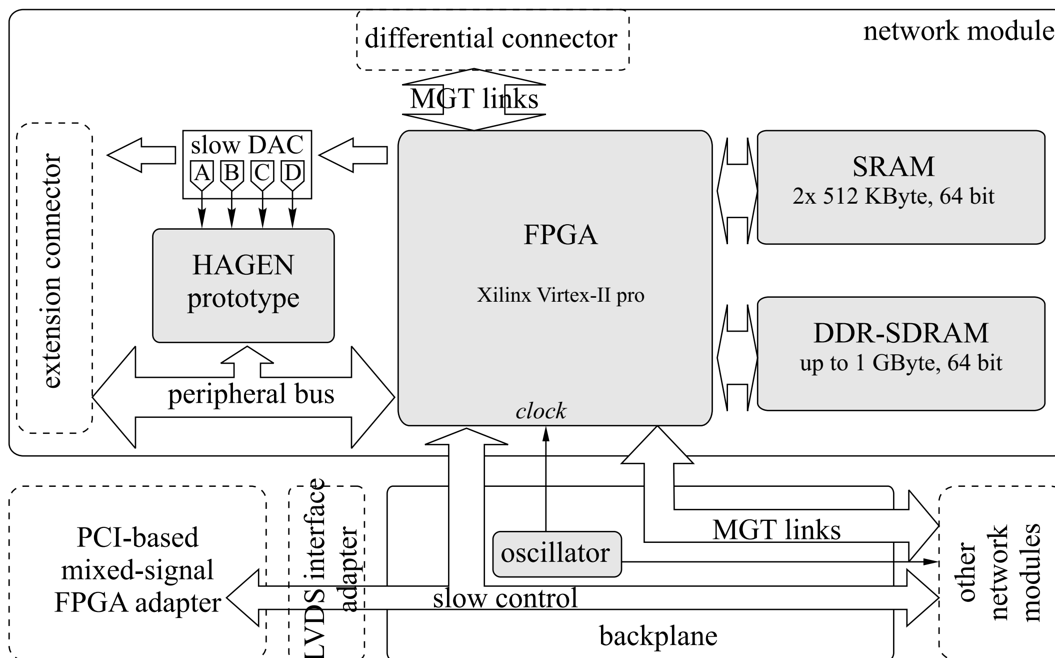


Figure 4.11: Schematic diagram of the network module PCB. Up to 16 of these modules can be plugged into a backplane and form a high-speed network. A slow-control token ring interconnects all modules with a controlling host PC. For this connection the PCI-based FPGA adapter with the LVDS interface board is used.

The advantage compared to the current framework implementation is two-fold: First, the new FPGA provides advances in the operation of LVDS links. In conjunction with its capability to shift clocks in fractions of the system clock, this allows to operate the HAGEN interface at higher frequencies and thus yields a speed-up compared to the PCI-adapter based framework implementation. A recent test operated the HAGEN prototype interface successfully at 200 MHz. Second, the utilized FPGA provides up to eight serial high-speed links that are capable of transferring up to 3.125 Gbit/s each. Due to the form factor of the network module, up to 16 of them can be mounted on a custom-built backplane that fits a 4 height unit rack mount (see Fig. 4.12). This backplane connects the network modules by the serial high-speed links and implements a torus-topology between them [77].

The design of this advanced framework is governed by the possible reuse of existing components and the extendibility for future applications:

- The backplane with its network modules is interfaced to a controlling PC with the help of the existing PCI-based FPGA adapter. The LVDS interface usually used for operating the HAGEN prototype is used to establish a slow-control network between the PC and the network modules.
- The migration path for the functional level of the experimental framework started by realizing the functionality of the PCI-based test system on each of the modules, i.e., the software (Sec. 4.3) runs on the controlling PC and accesses the ANN ASIC over the slow-control. In a second step, the software is migrated [86] to be executed on the embedded PowerPC running Linux [205]. In a third step, the individual network modules will cooperate via the high-speed network. A technical publication and an outlook to a possible application can be found in [57].

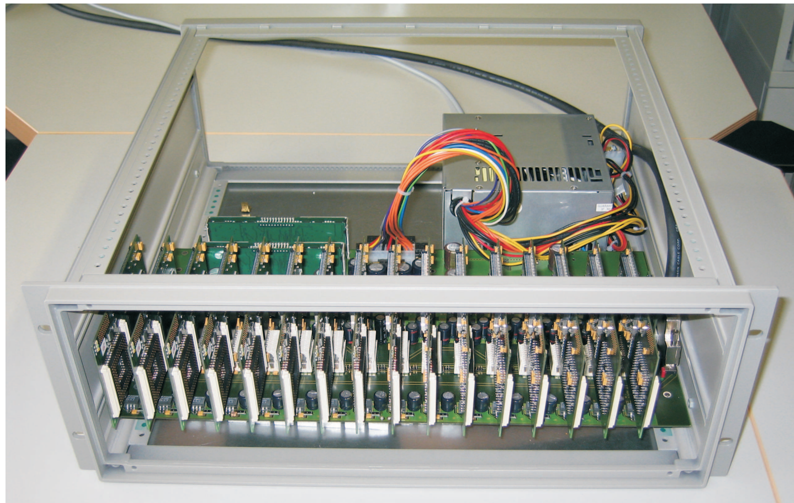


Figure 4.12: Photograph of a fully-populated backplane with 16 network modules in a rack mount.

- Once a larger ANN ASIC is developed, it is possible to interface it with the extension connector for future ASICs (see Fig. 4.11) to the existing parallel platform and reuse the infrastructure.
- Even the currently developed spiking neural network ASIC [185] will be accommodated by this distributed framework. This will be part of an Integrated Project funded by the EU (FACETS) [137].

This advanced framework is a conjoint work with J. Fieres, A. Grübl, S. Hohmann, S. Philipp, Dr. J. Schemmel, T. Schmitz, and A. Sinsel. Its operation and further development is beyond the scope of this thesis and will be published elsewhere. This project is supported in part by the European Union under the grant no. IST-2001-34712 (Sensemaker).

4.2 Programmable Logic Design

4.2.1 Overview

Programmable logic in the neural network experiment is responsible to interface the dedicated circuits of the ANN ASIC with the software running on a microprocessor. Furthermore, it can be used to realize tasks that profit from a dedicated implementation but that do not have an entirely finalized design. This even allows to start with a software implementation for certain tasks and migrate them to the programmable logic later on.

An easily reprogrammable substrate, i.e., the FPGA described in Sec. 4.1.3, is used as the target to implement the logic which is developed in the high-level developing language VHDL [46]. In a first step, the code is simulated to check its functionality on the signal level; this is done with ModelSim [141]. Once this step yields the desired results, the code is translated into a netlist of generic library elements[17]. Since this compilation to a netlist is independent from the target substrate, tools of different vendors may be used, e.g. FPGA Compiler II by Synopsys

[17]Depending on the targeted device, e.g., a specific FPGA or a digital ASIC in a given technology, the library elements may vary. This is a reason why code changes may be necessary if it is to be migrated from the FPGA to the ASIC.

[209] or Xilinx ISE [234]. The last step, has to realize this netlist on the target substrate, i.e., it has to map the library elements to available structures and to configure the necessary routing. For this task, a complete knowledge of the substrate's capabilities is necessary. Therefore, in case of the presented framework implementation with a Xilinx Virtex-E device the Xilinx ISE is used. After the synthesis, all data required to configure the FPGA is contained in a binary data file, the *bitstream*. As described in Sec. 4.1.3 this can be transferred across the PCI bus to the CPLD that configures the FPGA with the bitstream; a reconfiguration of the FPGA requires about 200 ms.

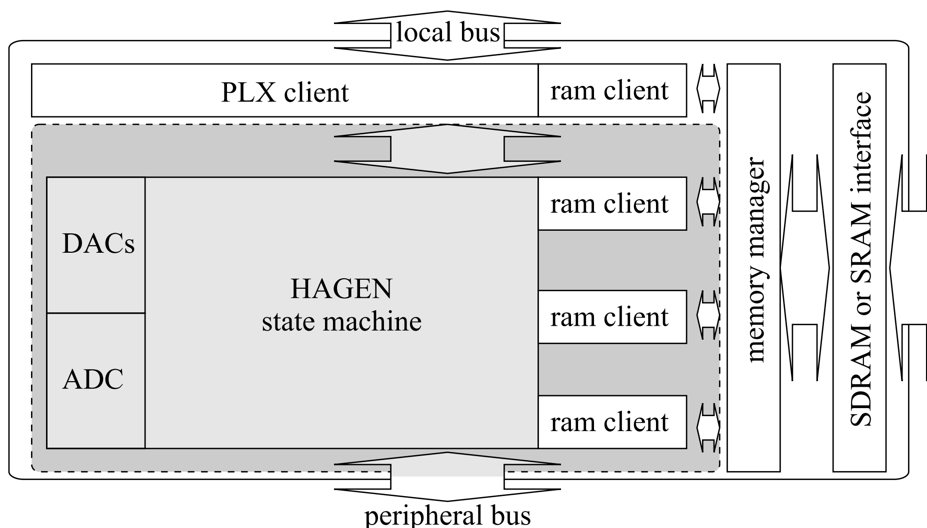


Figure 4.13: Simplified block diagram of the entities in the programmable logic when used to operate the HAGEN prototype. The dark gray area is available to extensions; the light gray shaded components are specific to the HAGEN ANN ASIC.

Fig. 4.13 shows a simplified block diagram of the programmable logic design as it is used for operating the HAGEN prototype. The white boxes represent entities that are generic to the test system and are likely to be reused if other ASICs are to be interfaced: The interface to the PLX PCI 9054 bridge, a memory manager that allows to handle and prioritize concurrent memory accesses, and an entity that manages the actual communication with the memory (SRAM or SDRAM). The dark gray area represents the available resources to extensions of the programmable logic. The light gray boxes show entities specific to the HAGEN prototype. It is to be noted that the size of a drawn box does not represent its amount of resource used. Rather, in the Virtex-E XCV600E device less than 10% of the flip flops are used by the PLX client, memory manager, and SDRAM client; another 15% are used for the HAGEN state machine[18].

In the following the entity interacting with the HAGEN prototype, the HAGEN state machine, is introduced. Furthermore, extensions are described that illustrate the flexibility of the experimental framework. A concise description of the programmable logic will be found in a related PhD thesis of T. Schmitz. The programmable logic design is a conjoint work of S. Philipp, M. Reuß, Dr. J. Schemmel, and T. Schmitz and presented to illustrate the operation of the HAGEN prototype.

[18]Depending on the remaining design, these figures may vary due to different place&route success of the synthesis software.

4.2.2 HAGEN State Machine

In order to control the HAGEN prototype interface with the correct timing, a state machine is used. Each of these states defines all required signals to issue two consecutive commands to the HAGEN. If those signals are to change, a transition between states or a loop of the same state will have to occur. These transitions are synchronous to the FPGA clock that is as well used for the operation of the local bus, i.e. the mentioned 50 MHz. With each clock, a 64 bit word is read from the memory. This is enough to prepare two consecutive commands to both HAGEN halves ($2 \cdot 2 \cdot 16$). By a special entity those commands are finally issued to HAGEN at double the frequency (i.e. 100 MHz) and double data rate. In the current implementation, both halves receive the same commands but different data, i.e., either both halves are being programmed or are in recall phase.

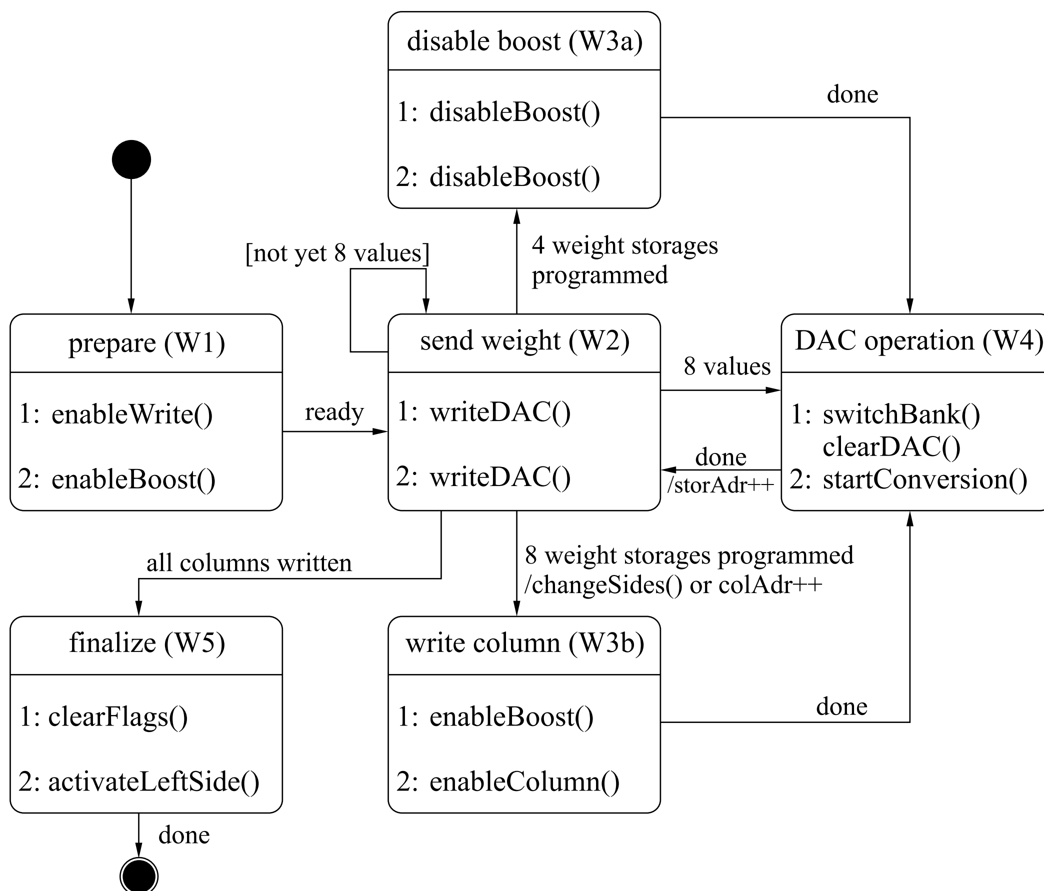


Figure 4.14: Simplified state diagram of the HAGEN weight programming. Each state issues two consecutive commands to HAGEN which are denoted with 1 and 2.

Fig. 4.14 shows a simplified state diagram of the weight programming. In each state two HAGEN commands are prepared and consecutively sent; they are denoted by 1 and 2. The shown commands are simplified clear text versions of the real commands which were presented in Sec. 3.5.2. Special transitions for the zero offset compensation of the DACs as well as the memory management are left out.

The state diagram illustrates how the weights of a complete network block are written. Each of the eight DACs is programmed (W2) before the conversion is initiated (W4). If all eight weight storage units of one side are filled, the weight column is transferred to the synapse array (W3b),

and the conversion for the other side starts. In the middle of the column programming the boost option is deactivated (W3a).

While it is clear from the principles of Sec. 1.2.2 and the commands of Sec. 3.5.2 that the HAGEN prototype can be set up to compute arbitrarily[19] deep network topologies, it is the state machine that controls in detail how the input/output data is handled, how many cycles there are performed, and when the network may get reset. Two major modes of operation are realized:

- multi-layer networks with static inputs,
- multi-layer networks with time-varying inputs.

Due to the on-chip feedback described in Sec. 3.2 the realization of these two modes differs in the way input and output data has to be sent to the HAGEN prototype: For a multi-layer network with static inputs it is sufficient to program the inputs once, perform the requested number of network cycles to calculate the hidden layers, and finally read back the output of the last layer. Time-varying inputs, on the other hand, require appropriate input before each network cycle. If furthermore a network response is wanted for each time step, after each cycle the output has to be read as well. This input/output overhead slows down the operation of the network compared to the static input case. For single-layer networks the performance is identical. If the input processing at the maximum network frequency is wanted, the direct I/O feature—which circumvents the I/O overhead—needs to be used.

Fig. 4.15 shows the part of the HAGEN state machine responsible for the operation in recall phase. The grayed boxes are used to visualize loops that can be used to vary the course of the network operation. States D1/D2 transfer the input data, in states D4-D6 the network response is read back. Depending on the value of *loops* an equivalent number of network cycles is performed in between writing the input and reading the output. This is the operation mode with static inputs. It is usually combined with a network reset (D0) by executing the preamble (c.f. Sec. B.4), i.e., for each input pattern the transition D6 to D0 is used.

If *loops* is set to one and *numpat* is larger than one, *numpat* input patterns will be evaluated with one network cycle each and without resetting the network (D6 to D1). With appropriately configured feedback connections this allows to use the network response in a consecutive cycle. This is the operation mode for time-varying inputs. In this mode as well the transition from D6 to D0 can be used: It resets the network for a new time series.

4.2.3 Extensions

The functionality realizable in programmable logic can go beyond the required entities that control the ANN ASIC itself or the remaining infrastructure such as the memory and the interface to the software. In the following extensions of the programmable logic design are presented shortly that show the flexibility of the experimental framework:

- The data intensive parts of an evolutionary training strategy have been successfully migrated from software to the programmable logic [189]. This is done transparently for the software running on the general purpose computer: The algorithmics in software remain mostly unchanged since they operate on objects. The data objects that previously were completely located in the main memory of the PC are replaced by proxy objects merely sending instructions to the evolutionary coprocessor. Since their actual data and all the operations on them are performed on the local memory of the FPGA adapter, the PCI bottleneck is avoided [86].

[19] As long as the weights are refreshed appropriately.

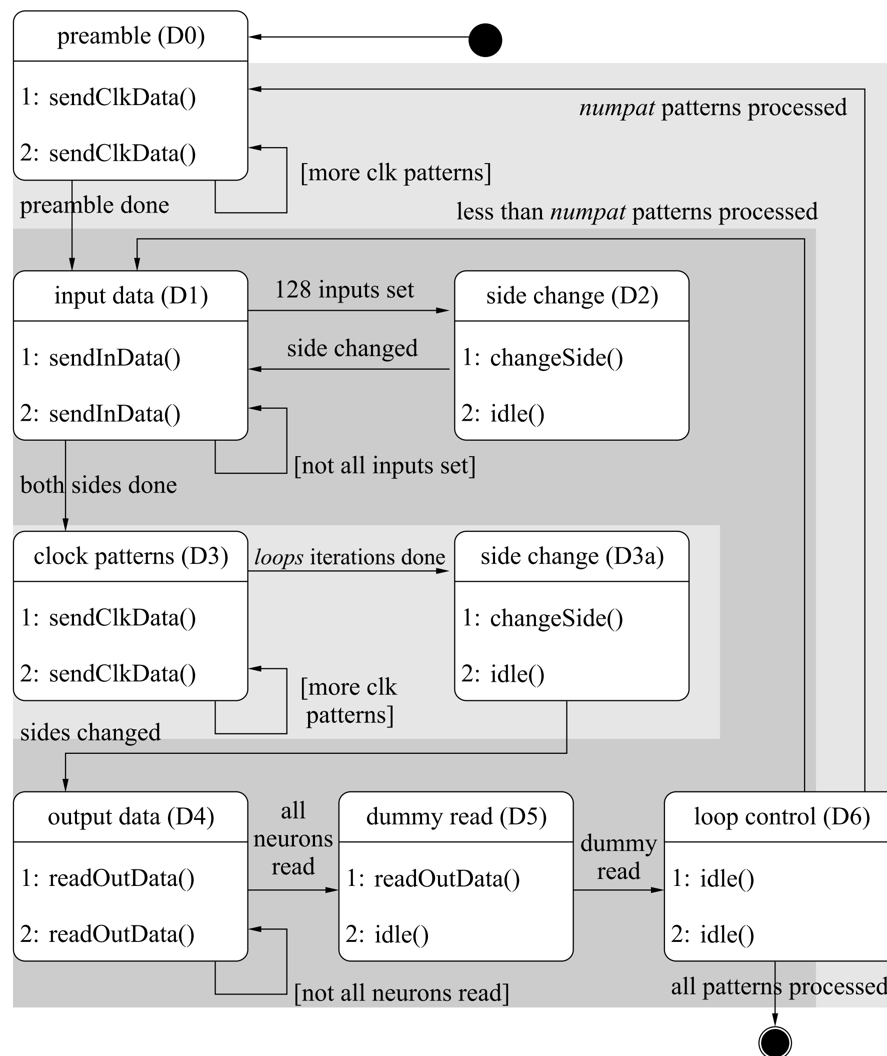


Figure 4.15: Simplified state diagram of the HAGEN network operation. Each state issues two consecutive commands to HAGEN which are denoted with 1 and 2.

- A totally different modification to the neural network experiment was inspired by the research on spiking neural networks: Following [200], the membrane potential of a biological neuron in the so-called high-conductance state is dominated by the excitatory and inhibitory conductances. In an approximation, the summing capabilities of the HAGEN perceptron can be used to accumulate these conductances and evaluate whether the threshold is reached, i.e. the firing condition. In order to emulate the time-varying course of these conductances, multiple inputs with appropriate weights are combined and activated over several network cycles. This mechanism can be extended to realize a refractory period for the neurons triggering it. Furthermore, a *feedback router* is incorporated that implements a switch matrix for the feedback connections and thus softens the constraints of the on-chip fixed feedback connections (c.f. Sec. 3.2). These programmable logic extensions, called *HASTE* (HAGEN Spike Translation Environment), are a current PhD project by M. Reuß of the Electronic Vision(s) group. In effect, the modifications utilize the computational power of the perceptron ANN ASIC but by emulation extend to a different neuron type. In order to constrain

the design space and to have early feedback, the HASTE extensions have been modeled in software [31].

- In order to use the HAGEN ANN for object recognition in a visual scene, a hierarchical network model has been implemented [57]. In each layer of this hierarchy identical features are sought after which makes the application suited for the hardware adaptation. But to present the detected features of one layer as input to the next, the data needs to be reordered. While this task is currently done in software, a reordering in programmable logic should yield a significant performance increase.

4.3 Control Software (HANNEE)

4.3.1 Overview

Interactive monitoring capabilities are a very important part in an ASIC prototype test system. Realizing this on a standard PC allows the highest flexibility since high-level programming languages and libraries for the graphical user interaction are readily available. Within the neural network experiment the software plays another important role: Since the training strategies are of research interest themselves, it is important that they can be easily changed while relying on the otherwise tested and operational components of the framework. Furthermore, it substantially simplifies the adaptation of the framework to new types of applications since the input/output data can be converted, pre-processed, analyzed, etc.

The software part of the neural network experiment is called *HANNEE* (Heidelberg Analog Neural Network Evolution Environment). It is the primary work platform for all co-workers of the Electronic Vision(s) group that do research on hardware neural networks. It is a conjoint work with J. Fieres, S. Hohmann, E. Mueller, T. Schmitz, and Dr. J. Schemmel.

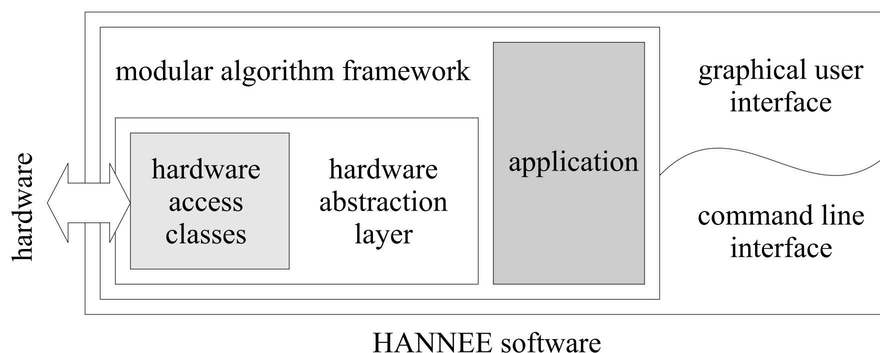


Figure 4.16: Overview of the HANNEE control software. It shows the encapsulation of the hardware access by an abstraction layer which makes the algorithmics independent from the actually used hardware. The gray-shaded boxes indicate where extensions for specific applications occur (dark gray) or modifications are necessary if the hardware is changed (light gray). All remaining parts can be re-used (white). Figure adapted from [86].

Fig. 4.16 shows a schematic overview of the HANNEE software. It is implemented using C++ [208] and object-oriented programming. This programming technique allows to program modular software and makes extensions and modifications especially easy:

- The hardware access is encapsulated in a set of objects, i.e. the hardware abstraction layer. It provides a common interface to the higher parts of the software but internally uses specific

code to access the hardware. This hardware-specific code is added by derived classes; this way all types of ANN ASICs that can be described in the network block approach can be integrated by only minor changes. Low-level aspects are described in Sec. 4.3.2; the high-level interface is described in [86].

- All other classes[20] of HANNEE are derived from a special base class, `HObject`. This base class equips objects of any derived class with the following features: The ability to represent its state in XML, and similarly, the ability to be initialized by an appropriate XML string. This property is used for an automatic generation of a *graphical user interface* (GUI). Additionally, all objects have a scripting capability which allows an autonomous remote execution with and without the GUI. Since these features are instantaneously available in any newly added class, HANNEE is easily extended. A detailed description can be found in [86].

- The adaptation to specific applications raises a problem: The inherent representation of the problem's data usually does not fit the binary format of the network blocks. Additionally, during training it may be desirable to use different subsets than for the generalization etc. A pattern handling mechanism that is used throughout the HANNEE software is described in Sec. 4.3.3. It is an efficient implementation, yet, modular enough to be easily adapted to many applications.

- Besides the features inherited from the HANNEE base class, a dedicated set of classes simplifies the extension of the HANNEE software with new training strategies: the modular algorithm framework. It provides flow control and interfaces to any kind of algorithm. Since it allows to chain algorithms, complex tasks can be tackled by interactively plugging in algorithms. Application specific extensions to this part of HANNEE are described in Sec. 4.4. A description of the framework itself can be found in [86].

The primary development platform currently is Linux with a 2.4.x kernel and a GCC 3.3 compiler [63]. In order to allow the migration to other compilers and platforms the following precautions have been taken:

- Only openly available libraries have been used which can be compiled for any platform supported by the GCC, e.g., the ACE library for multi-threaded applications [188].

- For the graphical user interface, the Qt library by Trolltech [212] has been used. It is openly available as well and can be compiled for Windows and Linux. In case no graphical interface is wanted, e.g., for the embedded application in Sec. 4.1.5, the software can be compiled completely without GUI.

- The QMake utility by Trolltech [212] has been used to create a meta-makefile of the HANNEE software. It allows to generate specific makefiles for all compilers and platforms supported by Qt, especially, the GCC [77] and Intel Compiler [101].

- The access to the hardware is performed using WinDriver 6.x [109]. It is available for all major platforms and encapsulates the hardware device by a kernel module (a device driver for Windows, respectively).

[20]Only classes that encapsulate performance critical, low-level functionality and which do not need any interaction by the user are excluded.

4.3.2 Hardware Abstraction Layer

If various parts of the software were to access the hardware anytime they wanted to, the hardware would most likely end up in an undefined state. The hardware abstraction layer therefore introduces a dedicated manager class, `HNetMan`, that controls the access to the hardware. On the other hand, it may be desirable to have multiple network configurations readily available in different parts of the software. For this reason, the class `HNetData` is introduced that provides a complete virtual image of a network block including the data that is to be processed. Any high-level part of the software[21] therefore can instantiate `HNetData` objects and have them queued for processing by the `HNetMan` responsible for the hardware to be used. Consequently, `HNetData` provides a generic interface for accessing its data which is independent from the actually used ANN ASIC. Nonetheless, it has to provide limitations similar to the ones of the network block on the ASIC—otherwise it could not be mapped. Therefore, for each ANN ASIC a derived class has to be provided.

HNetMan In the current implementation, an instance of the `HNetman` class is responsible for the access to the FPGA adapter. It utilizes specific hardware access classes that have all functionality to operate the interface to the programmable logic and the employed ANN ASIC. These hardware access classes were developed in [183] for a different ASIC but a similar setup of software, programmable logic, and dedicated hardware. The underlying concept is to represent different hardware modules by an appropriate class that is derived from a common base class, `TAccess`. From this base class a complete class hierarchy is derived as to represent hardware only consisting of registers, or a memory region not located in the main memory of the general purpose computer, or a hardware with run control.

At runtime, a tree of instances of these classes is constructed which represents the paths of the communication: In the current implementation, all interaction with the hardware goes across the PCI bus to the PLX chip. Its representing `TAccess` descendant therefore is the root node of this tree and is the only object actually accessing the hardware. It provides read/write functionality[22] to the local bus (c.f. Sec. 4.1.3) on the PCI-based FPGA adapter and therefore means to configure the FPGA (via the CPLD) and communicate with it after the configuration. It then allows to access the complete local memory on the adapter. This memory now can be partitioned into logical[23] memories by instances of other `TAccess` classes that are leafs to the root node (or to other nodes) and manage a certain address range. These child nodes never access the hardware directly, rather, they invoke the appropriate methods of the parent node.

Similarly, there exists a class for representing the HAGEN prototype ASIC, or more precisely, a class representing the commands and registers of the programmable logic that is controlling the HAGEN prototype. This class is not instantiated directly, rather, a derived class that also provides logical memory for the weights and input/output data in the local memory. An instance of this class is a child to the root node and manages the appropriate address and data range.

If the implementation of the neural network experiment changes, only the affected parts of the hardware access classes have to be replaced. E.g., for the use of the HAGEN prototype in the distributed system (c.f. Sec. 4.1.5) the root node of the hardware access tree has to be exchanged by an instance of a new class. The HAGEN class, on the other hand, can be fully reused.

[21] A high-level description of the hardware abstraction layer can be found in [86]

[22] Here, the WinDriver product comes into play: it translates read/write requests of data or blocks of data to the appropriate communication with a specific PCI device—in this case the PLX PCI 9054.

[23] In this context, the term 'logical memory' is used for entities that partition a physical memory and act like independent memories.

HNetData The `HNetData` class primarily provides a data structure to store all configuration data of a network block, i.e., the weights, clock patterns, bias voltages, etc. in the memory of the general purpose computer. Furthermore, it provides fields for the input and output data. The amount of input data and the run control information (e.g., the variables *loops*, *numpat* of the HAGEN state machine, c.f. Sec. 4.2.2) implicitly define a 'program' for the ANN ASIC. A `HNetData` object is then processed in total and after completion the network response can be found in its output data field.

Since an ANN ASIC can be comprised of more than one network block—the HAGEN prototype has four—`HNetData` objects can be chained. This allows to seamlessly integrate future ANN ASICs with more than four network blocks. Similarly, in a system of distributed HAGEN prototypes like the one of Sec. 4.1.5 the HAGEN ANNs cannot only be operated independently, but rather it is conceivable to have a master entity in programmable logic that enslaves the individual modules and allows to have a multi-HAGEN `HNetData` mapped to it. This is part of a current PhD research project by S. Philipp of the Electronic Vision(s) group.

4.3.3 Input/Output Pattern Management and Operation Modes

The `HNetData` class described in the previous section encapsulates a network configuration and workload of input data[24] to be processed by the network. To the external interface this input data is represented by an array of bit fields. Each bit field holds the input data for one network cycle of one network block and therefore has the length of the number of inputs to the network block. The order in the array is the order in which they get presented to the network block.

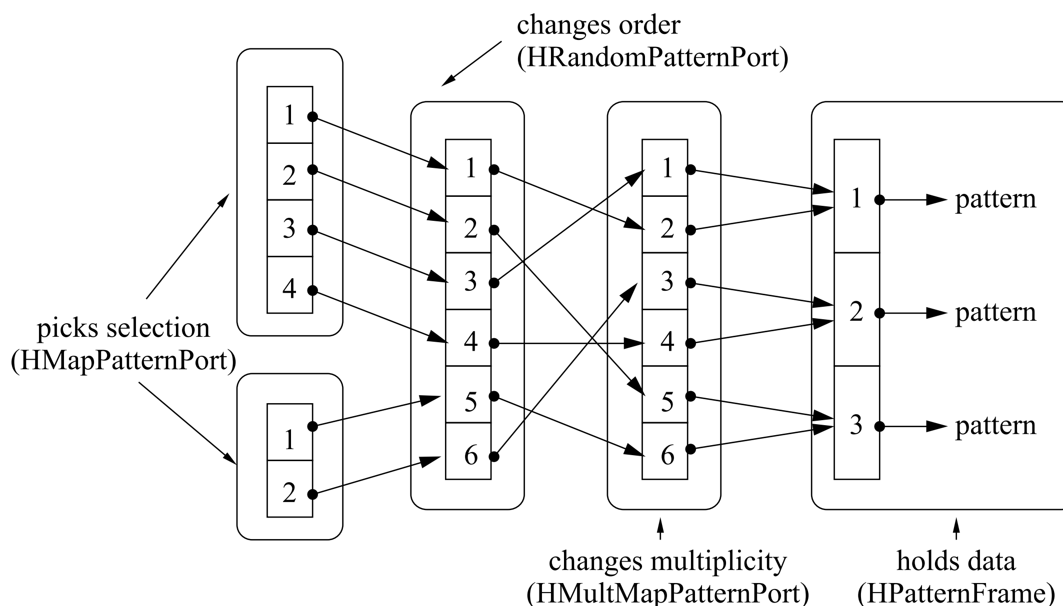


Figure 4.17: Schematic view of a chain of pattern ports.

This representation of the input is sensible to allow an efficient co-operation of hardware and software, but it is not the natural representation of a problem to be solved by the neural network: The 4-bit parity problem, for example, has 16 different input patterns of 4-bit width. Each input pattern has an associated 1-bit output pattern. In order to present this problem to the hardware, first of all, this 4-bit information needs to be properly assigned to the inputs actually used in the

[24] Similarly, there is memory reserved for the output data. It has the same organization as the input data field.

hardware. While this is trivially done by filling in corresponding zeros for the inputs not in use, it illustrates that there are differences in the representations. During training even more variations to the original problem are desirable: Due to the analog nature of the ANN ASICs used, it is desirable to present a certain input pattern not only once but, rather, show it repeatedly. Similarly, it is advisable to present these patterns in a random order to prevent artefacts.

The pattern handling that is introduced to the HANNEE software is based on the axiom that a pattern is only the very nature of the problem and all the other things such as training order, selection, multiplicity etc. are features of how one looks at the patterns. Basically, this represents a ‘chain of views’ to the actual patterns.

A class hierarchy has been implemented that allows to chain views at runtime; these views are called *pattern ports* which is owed to the fact that they indeed can do more than just viewing, e.g., modify the underlying patterns. An exemplary chain of ports is shown in Fig. 4.17. In appendix C a simplified version of the class hierarchy is illustrated in Fig. C.1, and tables C.2, C.3 shortly explain the most important classes; a complete reference can be found in the source code. There are specialized versions that actually provide data, reorder data (e.g. random order), partition data for training and generalization test, or that take care of the multiplicity of patterns. The separation of these features allows it to reuse these ports while changing the data source. The functionality of the data source is provided by the last pattern port in the chain, i.e. the base port. Currently, ports are provided that hold data in a value array structure of the *standard template library* (STL) [108], `HVAPatternFrame`, or ports that generate the data on the fly (descendants of `HGeneratorPort`). Furthermore, there are completely user defined ports conceivable that, for example, retrieve patterns from a database.

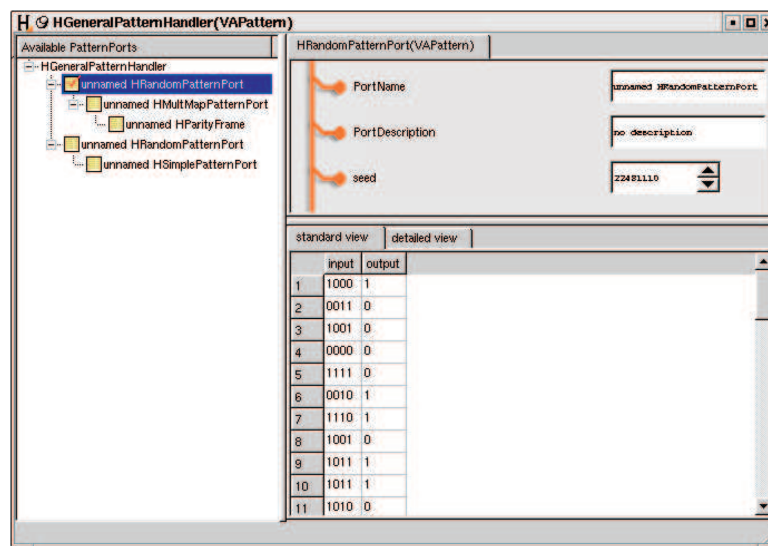


Figure 4.18: Screen shot of the GUI of a pattern handler that manages two chains of pattern ports. The GUI is partly generated automatically, partly specifically designed to allow drag and drop functionality etc.

One or more of these chains of ports can be organized by a *pattern handler*; the most general handler provided is the class `HGeneralPatternHandler`. It stores all pattern ports and chain information in XML format and allows the efficient interconnection of chains of ports. Fig. 4.18 shows a screen shot of a pattern handler with two pattern port chains.

A crucial point has not been mentioned so far: the patterns themselves. Accessing patterns by the proposed pattern ports allows to separate the actual pattern implementation from their management. On the other hand, for performance reasons it is desirable that a pattern port can efficiently iterate across the pattern data without the repetitive need to resolve virtual function calls. The solution to this kind of problem is to use the template feature of C++ which is extensively used in the STL. Somewhat similar to the polymorphism mechanism this allows to write generic classes, but with the essential difference that the specialization of a template class is determined at compile time instead of being dynamically assigned during run time. The template feature therefore allows code that can be as efficient as a hard coded implementation, yet, it remains generic. The pattern port hierarchy therefore is implemented as a template hierarchy and the pattern class is the template parameter[25].

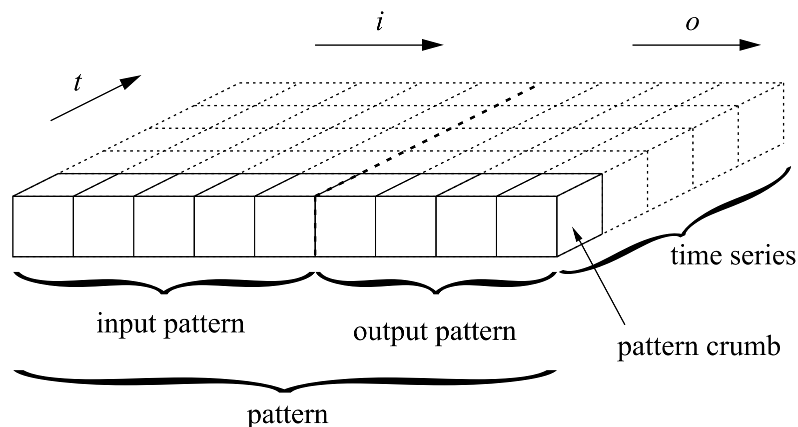


Figure 4.19: Schematic diagram of a pattern.

The task of the pattern class is to organize a *pattern* into an *input pattern* and *output pattern* and to be able to present them in *pattern crumbs* that are of an elementary data type. Thus, the internal representation is up to the pattern class, but it has to provide efficient iterators on the input and output pattern. Fig. 4.19 shows a schematic diagram. A set of patterns that have causal order in time is called a *time series*. Since from the pattern port's point of view these time series of patterns behave similarly to ordinary, independent patterns, the time series capability is as well managed by the pattern class; the pattern class thus is more than a single pattern.

Without loss of generality it is assumed that the patterns of a single pattern port[26] have the same input (i), output (o), and time dimension (t). The proposed implementation of a pattern class (`VAPattern`) therefore uses the value array class of the STL to organize its pattern crumbs. The chosen elementary data type is a signed integer which should suffice to represent the variables of the problem in most cases. Especially, in the case of multi-bit variables it is desirable to have them represented appropriately as long as the pattern is dealt with on a high level. For the pattern import to the `HNetData` structure, on the other hand, it is necessary to translate them to a binary representation of the chosen bit size. A meta structure of the pattern class, the so-called *pattern trait*, provides the necessary information to interpret[27] the pattern crumbs: A single pattern crumb can be expanded to a binary representation of choosable width. Similarly, several pattern crumbs

[25] Even the GUI classes that extend the automatic GUI generation are implemented as templates. Only in a few specific cases a specialization was necessary to accommodate the specific needs of the chosen pattern parameter class.

[26] There is always the possibility to use several pattern ports.

[27] More information can be found in the source code.

can be contracted into a single multi-bit value. This allows to pool input neurons or to interpret the information of several output neurons as a coherent value.

Encapsulating the pattern information in a parameter class with only few interface requirements allows to easily implement different pattern classes and use them right away. For example, a pattern class optimized for storage is conceivable that stores several bits (even if they do not form a multi-bit value) in the same pattern crumb. The translation from the pattern port representation to the `HNetData` format is done by only using the `HPatternPort` interface and the parameter class interface functionality. At this point, the proper run control information to the `HNetData` structure is provided as to configure the ANN ASIC for static inputs or time-varying inputs (c.f. Sec. 4.2.2). The operation mode of the ANN ASIC therefore is determined by the type of patterns applied.

4.4 Applications

The components of the neural network experiment mentioned so far are fundamental for the actual operation of the ANN ASIC and generic in nature. But providing means to configure and operate a neural network is only one part. In addition, appropriate training strategies are required that actually allow to explore the implemented network paradigm and to actually solve application problems.

4.4.1 Hardware-in-the-loop

The preceeding chapters explained in detail how the hardware-implemented ANN performs the evaluation of Eq. 1.1 but simultaneously pointed out the peculiarities of an analog implementation. Set aside the transient noise, the actual network deviates from the ideal description of Eq. 1.1 due to fixed-pattern effects in the weight programming, the synapses, and the output neurons. One way is to try to measure these deviations and compensate them; this will be discussed in Sec. 4.4.2. Another is to use a hardware-in-the-loop setup to have the outcome of a nominal weight configuration be evaluated by the hardware directly, rather than using a formula to predict it. In this kind of setup, the hardware can even be a black box as long as the algorithm can test a configuration of this black box against a set of known patterns; iteratively, it can evaluate various configurations and finally may come up with better solutions.

If there is no continuity in the behavior of the black box to its configuration parameters, the algorithm merely implements a random search. This is, of course, not true for the HAGEN prototype: Its response to variations in the configuration parameters, e.g. the weights, is continuous by construction, only the precise shape of the characteristic curve of certain elements may not be known exactly. A hardware-in-the-loop algorithm therefore can do better than plain trial and error. It can try to vary configurations that are already performing better than others according to the performance measure. Or it can try to combine two configurations in the hope that it improves performance. Indeed, this is not an abstract concept. Quite contrary, natural evolution shows how well this algorithm works. Despite the actual encoding of the configuration, individuals are ‘evaluated’ according to an implicit performance measure of the environment, i.e., their ability to survive and reproduce. This so-called *selection* sorts out the individuals that are allowed to produce offsprings. Due to *mutation* and combination of genetic information, the *crossover*, the next generation inherits already successful features which may be slightly altered or combined to new features. These changes are directionless, i.e., whether they are positive or negative is judged upon by the survival.

Many evolutionary algorithms for training neural networks are conceivable (see, e.g. [236]); the successful development of evolutionary algorithms especially suited for the type of hardware presented here is studied with the help of the neural network experiment in [86]. There, the *HEAF* (HANNEE Evolutionary Algorithm Framework) extension to the HANNEE software is introduced which provides interfaces and a class collection to train the weights of an ANN ASIC. Due to the hardware abstraction, this training is independent of the actually used ANN ASIC. These investigations show competitive results in commonly used classification benchmark problems and underline the applicability of the implemented neural network paradigm.

Later on in this thesis, in Sec. 5.3.3, the HEAF extension is used to train a reference network for comparison with the liquid computing approach. A relatively simple evolutionary algorithm is used that is readily available in the HANNEE framework and already proved its applicability on several occasions [193, 88].

The algorithm works as follows: In a first step, the topology is defined, i.e. it is chosen which input and output neurons of a network block are used, which feedback connections are activated, and how many network cycles are to be performed. In a second step, the synaptic weights are configured on which the algorithm operates on. All weights connecting to a single output neuron form a chromosome with the single weights being the genes; the chromosomes of all considered output neurons form the genome of one individual which represents the weight configuration of the network. The used genome class allows to assign a constraint to each weight: There are weights that can get modified during evolution, ones that can be preset and never get modified, and others that are inactive[28].

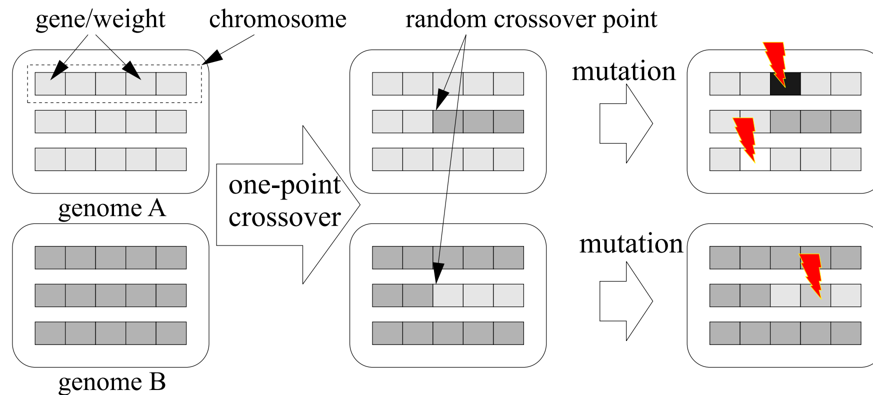


Figure 4.20: Schematic diagram of the chromosome-wise one-point crossover and mutation step after selection.

At the start of the evolution, a population of ϕ individuals is generated with random genes. In order to generate the next generation, individuals are selected by a standard tournament selection[29] of size $\tau = 2$ [22]. Basically, two individuals are randomly picked and evaluated according to the same fitness measure. The one with the better fitness is passed to the next generation. Following this selection, all surviving individuals are randomly grouped to mating pairs that undergo a chromosome-wise one-point crossover with probability χ. In this step weight data of two individuals can be exchanged[30]. In a final step, single weights can get modified by two types of

[28]These are set to zero and for example allow layered topologies, c.f. Sec. 1.2.2. For performance reasons these inactive weights are not encoded in the genome directly, rather, are considered before the genome object is generated.

[29]Alternatively, a rank-based selection scheme is used.

[30]To give this exchange a meaning it is per chromosome, i.e., the weights of the incoming connection for the same output neuron are combined.

mutation operators: With probability μ_u each weight is replaced by a new weight drawn from the valid weight interval with uniform probability. With probability μ_n a Gaussian mutation operator is applied that changes the existing weight value by an amount that is drawn from a Gaussian distribution of width σ_n. Fig. 4.20 schematically illustrates the crossover and the mutation of two genomes after selection. A special elitist option, finally, allows to take *e* copies of the best individuals completely unchanged into the next generation. The actually used values of these parameters will be given in the context.

The fitness of an individual is evaluated according to a simple mechanism: Since the training is supervised, each input pattern has an assigned output pattern with a nominal value. The actually calculated value by the network is compared to that nominal value and for each matching bit the fitness is increased by 1. The maximum fitness will be achieved if for all patterns all bits match.

4.4.2 Precalculated Weights

Usually, there is no difference between the actual weight and its representation by a variable—at least in digital hardware. In the neural network experiment with its mixed-signal ANN ASIC, on the other hand, there is a difference between the nominal value of a weight, i.e., the weight representation in software (the quantized digital representation in the programmable logic, respectively) and the actual weight implemented by the analog synapse. The concepts of Ch. 1 aim at minimizing this discrepancy and the HAGEN prototype implementation allows to compensate the fixed-pattern effects by additive offsets to the synapses.

In [184] and [86] a procedure is introduced with which the DAC offset (Sec. 3.3.4), the output neuron (Sec. 3.3.3) offset, and single synapse offsets (Sec. 3.3.2) can be infered from measurements by only evaluating the fire output of the respective neuron. It is to be noted that the mentioned offsets are not measured in absolute values, but rather in LSB increments of the corresponding DAC. This is because the measurement tries to sweep a dedicated synapse to adjust the input activity of a neuron in a way that it is firing half the time in repetitive measurements, i.e., the trip point of the neuron. Ideally, this happens if a positive weight and a negative weight of same amplitude are activated; in the absence of systematic effects such as fixed-pattern offsets, it is to be expected that the comparator in the output neuron yields random results due to temporal fluctuations. In practice, there will be a difference in the amplitudes necessary to achieve this trip condition from which the actual offsets can be extracted.

Once performed for all output neurons of a HAGEN prototype this data can be used to compensate the offsets of a specific die. Two types of offsets should be differentiated: The DAC/neuron offsets, and the single synapse offsets. The former are part of the design and meant to be compensated for a regular operation. Since the compensation of the individual synapse offsets requires a preprocessing of each newly generated weight configuration, the latter type of offset compensation is not always desirable. The DAC offsets, which by design are the zero-values of the DACs, can be subtracted by an extra column of synapses in the weight storage unit of each network block (c.f. Sec. 3.3.4). The compensation of the output neuron offsets can be realized by a column of synapses in the synapse array that is operated as a bias (i.e., the column is always on) and constantly subtracts a weight equivalent to the offset. The synapse offsets, finally, may be compensated by an additive offset to each synapse weight which can be stored in an additional `HNetData` object.

The fact that the offset compensation does not ensure the same exact weight currents for synapses of different output neurons with nominally identical weight value is not important: Since no analog information is transfered between neurons, it suffices to compensate the offsets within each synapse row. The digital representation in DAC codes (of the DAC that measures and pro-

duces the weights!) then allows to consistently use precalculated weights for neurons that may be programmed by different DACs (c.f. Sec.3.3.4) and therefore can even be located in different network blocks or on different dice.

A successful offset compensation allows the representation of a network by weights that are independent from the substrate. On the one hand, this makes it possible to transfer configurations from one ANN ASIC to the other [86]. On the other hand, it allows the derivation of weight configurations by whatever means and have it meaningfully be evaluated on the ANN ASIC. This is used to explore the dynamics of input-driven network liquids in the *liquid*HAGEN setup on a homogeneous substrate, see Sec. 5.2. Furthermore, the variable network resources introduced in Sec. 1.2.2 are an application; respective measurements are shown in the appendix D.

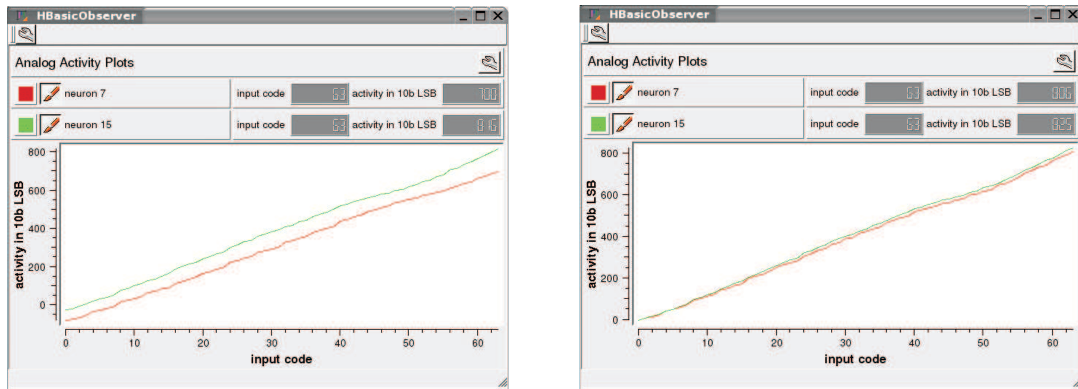


Figure 4.21: This figure shows the measurement output of the `HMeasureAlgo` while measuring a 6 bit multi-value input to two output neurons. The left side shows the measurement without neuron offset compensation; the right side yields the measurement with DAC and neuron offsets compensated.

In order to evaluate the achievable resolution, a measurement algorithm has been developed that integrates itself into the modular algorithm framework of HANNEE. Via the dedicated algorithm interface, the `HMeasureAlgo` algorithm can be chained to any other algorithm and measure the postsynaptic activity for the provided weight configuration and input data. Of course, the algorithm has to infer this activity from the fire output of the neurons (similar to the calibration algorithm) and therefore returns the activity in LSB of the corresponding DAC.

For determining the activity, the measurement procedure used for the calibration (shortly described above) is adapted since it allows an offset-free measurement [184, 86]. In order to speed up the measurement, the sweep-based measurement routine is only used in a small value range around the expected trip point. The trip point itself is roughly localized by an interval nesting approach: A single synapse is first set to the upper and then to the lower limit of an interval[31] in which the trip point is expected[32] and at both limits the neuron's fire output is evaluated (repeatedly). Then the measurement synapse is set to the midpoint of the interval and the fire output is again evaluated. This midpoint now resembles the new upper or lower limit of the interval—which one is determined by the fact that the trip point lays in the interval in whose limits a transition of the observed fire output occurs. This procedure is repeated until a flickering in the neuron output

[31] In the current implementation, the maximum interval is the dynamic range of a single synapse. With little modification it is possible to extend this range by using several synapses set to maximum positive/negative weight and activate a binary weighted number of them.

[32] Since the measurement algorithm was primarily developed for the variable network resource experiments the expected trip point always lays in the dynamic range of one synapse.

during the evaluation of an interval limit is observed. In an adjustable interval around that trip point bottom-up and top-down sweeps are performed which eventually determine it exactly.

Fig. 4.21 illustrates the measurement output for two 6 bit multi-value inputs to output neuron 7 and 15 (they are programmed by different on-chip DACs). According to Sec. 1.2.2, six consecutive synapses are binary weighted and the maximum code causes 80 % of the maximum single synapse activity. Plotted are the measured activities in LSB of the on-chip DAC versus the applied multi-bit value (input code). The desired outcome of the measurements are two identical, monotonically growing straight lines that start at zero. While a quantitative analysis is provided in appendix D, it can readily be seen that monotonicity is achieved. The difference between the two measurements is the neuron offset calibration which is disabled on the left side and enabled on the right side (furthermore, the DAC offset is calibrated as well as the individual synapses). The shift in y-direction on the left hand side plot directly yields the neuron offset.

4.4.3 Liquid Computing

The adoption of liquid computing to the neural network experiment (c.f. Ch. 2) yields a quite different approach compared to the other two described above: A recurrent neural network, i.e. the liquid, is used as a temporal integrator and non-linear filter to the inputs. The network itself is not trained, rather, it has essentially random weights (c.f. Ch. 2). The adaptation to a problem is realized by a specific observer to the network; it is this readout that gets trained. While this readout *can* be done using a (separate) neural network, it does not have to. Especially, for reasons explained earlier, the readout here is implemented by a linear classifier. Thus, the 'training' reduces to a well-defined least-squares linear regression (Sec. 2.2.2). This difference, to e.g. an evolutionary training approach, is important to note since it implies convergence at an ensured number of steps if a solution exists.

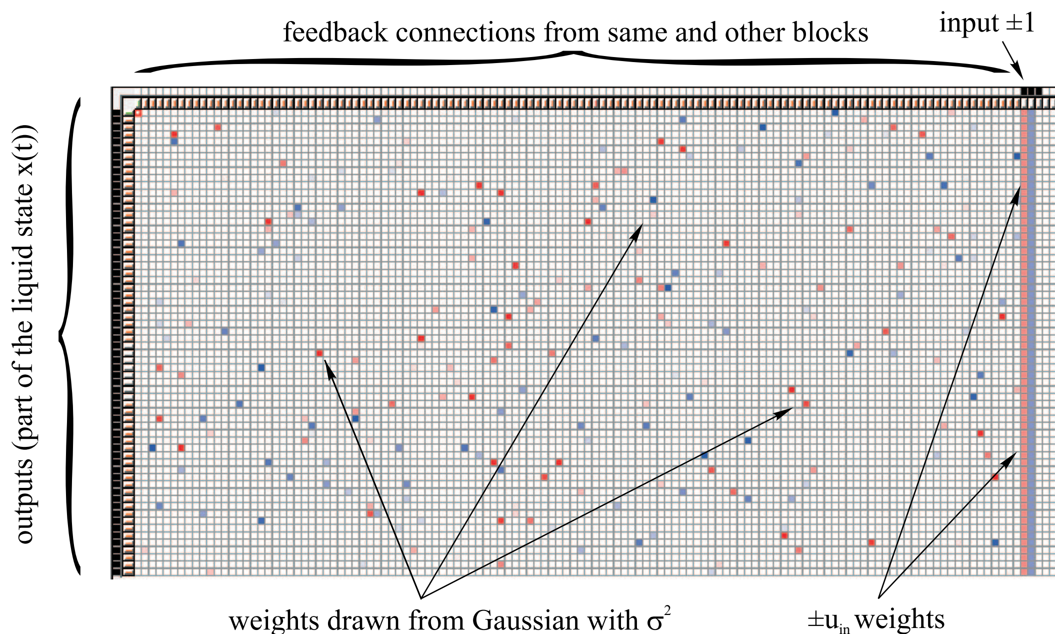


Figure 4.22: Typical weight configuration of a single network block in the *liquid*HAGEN setup for a liquid with $k = 4$, $\sigma^2 = 0.5$. Red squares indicate positive weights; blue squares negative ones. The intensity is encoding the amplitude.

The liquid state machine implementation proposed in Sec. 2.2.2 chooses to configure the HAGEN prototype as the liquid and to implement the readout in software. Accordingly, the HANNEE software has been extended to provide means to generate different kinds of liquids, export the liquid state, and to train the readout. These extensions seamlessly integrate into the modular algorithm framework; this allows to use some of the functionality in even totally different contexts.

Generation of Liquids As described in Sec. 2.1.2, there are various ways to configure a neural network to act as a liquid: Examples are the neural microcircuits according to Maass et al. [131] or the input-driven networks according to Bertschinger et al. [21]. Even within these different strategies, various parameters influence how the dynamics of the liquid are. Fig. 4.22 shows an exemplary weight configuration of a single HAGEN network block being part of an input driven network liquid drawn from $k = 4$, $\sigma^2 = 0.5$ (c.f. Sec. 2.3.1). In order to explore these parameters as well as different types of liquids, software classes have been developed for the generation of microcircuit-type liquids and one for input-driven network based liquids. They are derived from a common base class `HNDGenerator` which unifies their common features. On the one hand, they both generate a complete `HNetData` object (or chain respectively) with all necessary feedback connections and weights according to their generation parameters. On the other hand, the generation parameters, e.g., the width of a specific distribution or the number of connections, may be varied in certain limits in order to explore the effects on the liquid dynamics.

For the latter purpose, a dedicated algorithm has been developed that allows to choose the desired `HNetData` generator and which extracts variables indicated as generation parameters. This `HSweepAlgo` integrates itself into the HANNEE algorithm framework and allows a systematic sweep of the extracted generation parameters and the evaluation of each configuration. In [86] it is described how variables in the `HObject` concept are managed. They are all derived from a common data type base class, `HValue`, which provides the XML capability and GUI generation. Introducing a specialized class, `HMutable`, that takes an existing `HValue` variable as reference indicates a variables to be swept or to be otherwise modifiable. For the elementary numerical data types (inclusive Boolean) derived from `HValue` this class provides a unified interface for the step control including range checking, step size control, initialization etc. A simplified class interface is shown in Fig. 4.23.

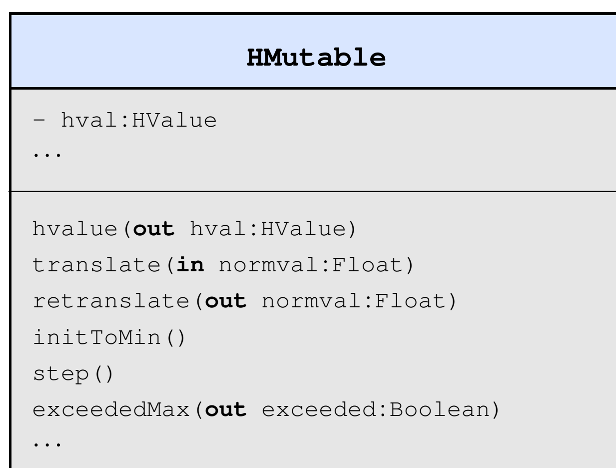


Figure 4.23: Simplified representation of the `HMutable` class. The diagram does not show the complete class interface.

While the above concept is currently applied to systematic studies of liquids, it is well suited to be reused for other applications. For example, an evolutionary algorithm could be used to explore the dynamics of a liquid. This could be easily done since the `HMutable` class maps the discrete

value range of a configuration variable to the normalized interval $[-1,1]$ independently of the data type. An evolutionary algorithm operating on an abstract representation therefore does not need to care how it is realized. But the mechanism of providing certain variables to the outside with an interface to modify them in a controlled fashion is not restricted to the generation of liquids. Quite contrary, other `HNDGenerator` descendants are conceivable that for example perform a measurement.

Providing an extra module for the liquid generation in the liquid state machine setup allows its extension. One example for this are liquid computing experiments performed using this setup in conjunction with the HASTE extension to the HAGEN ASIC [31]. In order to use the flexible feedback routing and the modified neurons provided by HASTE (c.f. Sec. 4.2.3), only the `HNetData` generation needed to be adapted.

The Learning Tool For reasons of flexibility and interchangeability the readout part of the liquid state machine is implemented in software. Yet, there are alternative ways of doing that: One can use the programmability of commonly used algebra tools such as MATLAB [79] by MathWorks or one can extend the HANNEE software with a dedicated custom-built implementation in C++. Both ways have been pursued in this thesis. The MATLAB approach is described first.

The research group of Maass published a MATLAB extension, the *Learning Tool* [155], that implements different readout strategies and provides some analysis capabilities. This extension usually interconnects with a liquid simulator written in C implementing spiking neural networks (CSIM, as well introduced in [155]). Both tools are published under the GNU GPL [62] and are meant to spread the liquid computing idea. They are distributed on a dedicated portal web site [148]. Using MATLAB for the readout is sensible since one can revert to its powerful mathematical library, and it allows to interactively check and visualize the results.

First steps with a liquid implemented on the HAGEN prototype have been done using the Learning Tool; Sec. 5.2.1 shows the experimental results. For this reason, a MATLAB export has been developed which allows the extraction of the liquid state information and the transfer from the HANNEE software to MATLAB and the Learning Tool. This exchange is file-based and therefore not very efficient, but it allows the evaluation of liquids implemented on the HAGEN ANN ASIC in the analysis tool used by other researchers dealing with liquid computing. Furthermore, it provides a good cross check for the dedicated software implementation explained next.

Advanced Readout Implementation Implementing the readout within the HANNEE software of the neural network experiment has the advantage that it can be realized by an optimized code and thus be very efficient. All overhead necessary for data exchange and scripted elements in the MATLAB tool chain can be avoided or shifted to a post processing that is not time-critical. This make the extensive parameter sweeps of Sec. 5.2.2 and the following sections possible. Additionally, the response of the readout can immediately be used within the HANNEE software. This allows the utilization of the response as a fitness value in an evolutionary algorithm and makes it possible to use the liquid setup for interactive applications. For the moment, the advanced readout implementation realizes a linear classifier readout. This is, of course, no general restriction and it is conceivable that it is extended with a readout based on a multi-layer perceptron which then even could be implemented on the ANN ASIC itself. The advanced readout implementation is visualized in Fig. 4.24.

The readout integrates itself as a *fitness function* to the HANNEE modular algorithm framework. The concept of fitness functions is in detail explained in [86]: A fitness function has a representation in the `HObject` hierarchy which is used for parameters and to obtain a function pointer to the actual code that calculates the fitness; this mechanism is chosen to reduce the overhead that

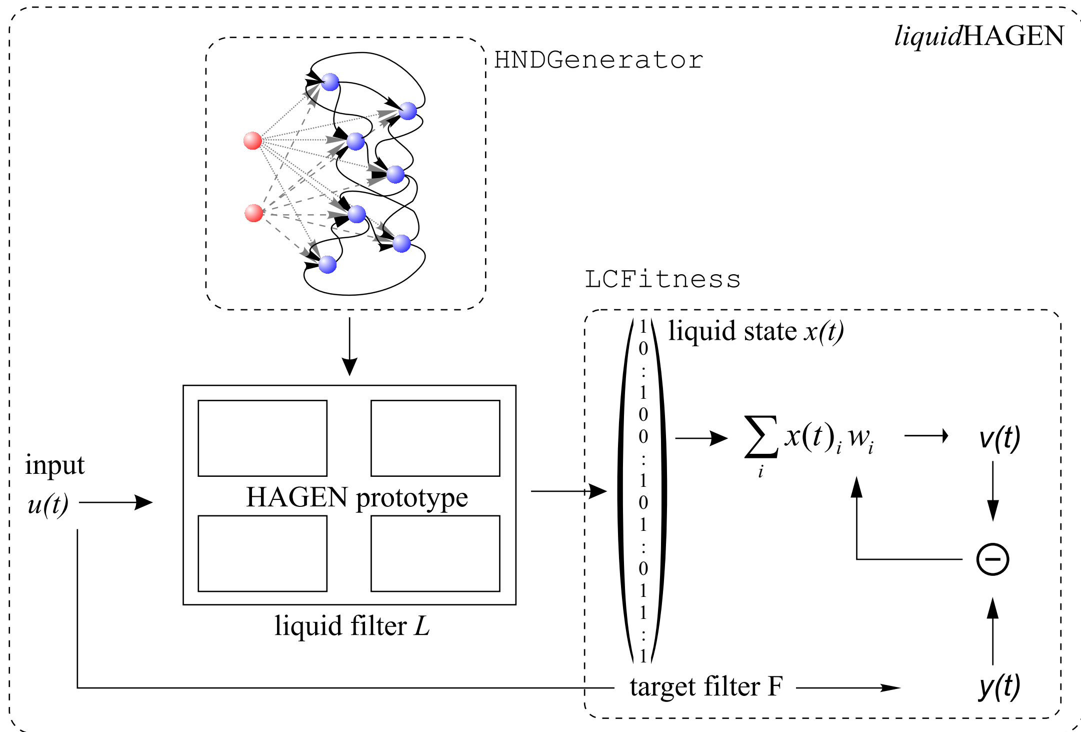


Figure 4.24: Schematic diagram of the software entities that configure the liquid topology (HNDGenerator) and implement the readout (LCFitness).

may occur if the fitness evaluation is called for single patterns iteratively. The fitness function is immediately called after the evaluation of an HNetData object by the hardware, with the same HNetData being the argument and is the dedicated place to do efficient low-level computations. The principal fitness value is either the mutual information, memory capacity or averaged versions of this if several readouts are trained simultaneously. While in many applications the fitness function only returns this principal number, the data container class employed for the liquid state machine setup allows the transport of other data out of this inner loop. This is necessary for the analysis of individual classifiers' performances or other data such as the weight configuration. Great care is taken to avoid time-consuming tasks such as the reallocation of memory in successive runs.

For each run of a liquid, i.e., after the processing of a HNetData object configured by one of the liquid generators, the LCFitness class computes the least-squares solution to the overdetermined set of linear equations described in Sec. 2.2.2. Once the weights for the readouts are determined, a matrix-vector multiplication is performed on a generalization set of input data that has not been used for computing the linear regression. If one was to implement this naively, the resulting performance most likely would be disappointing. Luckily, there are well established implementations of linear algebra routines. A prominent example for efficient subroutines is the publicly available BLAS (Basic Linear Algebra Subroutines) collection. It provides vector-vector operations in BLAS-1[119], vector-matrix operations in BLAS-2 [51], and matrix-matrix operations in BLAS-3[50] optimized for various data types. Additional packages have been developed on top of these BLAS routines that provided more complex algorithms such as various types of matrix factorization algorithms; an example is LAPACK [5]. These libraries are originally written

in FORTRAN and are used in many contemporary mathematical libraries, e.g., the GNU Scientific Library (GSL) [66], the Intel Math Kernel Library (MKL) [102], and even in MATLAB [79].

In terms of portability, the GSL is the best choice if a commonly available implementation of LAPACK and BLAS is wanted; it can be compiled for most platforms. On the other hand, even with special compile flags it may not be the most efficient implementation for a specific platform. Since the general purpose computers used for the liquid experiments have an Intel Pentium IV processor and Intel provides a hand-tuned math library (MKL) that includes an implementation of LAPACK and BLAS, a compromise has been chosen: For the Householder algorithm as described in Sec. 2.2.2 the implementation of the GSL has been adopted. Yet, instead of using the BLAS and LAPCK routines provided by the GSL, optimized routines from the MKL are used. This is possible since the employed matrix and vector data structures of the GSL are compatible with the memory representation required by the BLAS routines. The used functions are shortly explained in Tab. 4.2. If the setup is to be used on non-Intel platforms, the code only has to be modified as to call the GSL routine `gsl_linalg_QR_lssolve()` directly.

`dgeqrf`	QR factorization ('qrf') of a general matrix ('ge') with double precision ('d'); LAPACK routine
`dtrsv`	solves a set of linear equations ('sv') with a triangular matrix ('tr') and double precision ('d'); BLAS-2 routine
`dgemv`	matrix vector product ('mv') of a general matrix ('ge') with double precision ('d'); BLAS-2 routine

Table 4.2: Overview of the used BLAS and LAPACK routines for the Householder Algorithm with the naming scheme of these packages illustrated in brackets.

Even though the above effort allows to realize all required parts of the liquid state machine within the *liquid*HAGEN setup, an interface to MATLAB is still desirable. Especially, for the visualization of parameter sweeps or the subsequent analysis of individual readouts or other performance measures, e.g. the mutual information on subwindows in time (c.f. Sec. 2.3.3), it is still the adequate tool. For this reason a specific MATLAB export class has been developed that has a counterpart on the MATLAB side. This allows the automatic transfer of all relevant information for a liquid experiment to MATLAB where a collection of visualization and analysis functionality has been developed. This tool chain has already been reused [31].

Degradation Setup The robustness against faults in the substrate after training is examined in different setups: To see the degradation over time (Sec. 5.3.2) is suffices to evaluate a sufficiently long time series without refreshing the weight configuration. The performance is evaluated using the mutual information evaluated on subwindows in time as introduced in Sec. 2.3.3.

For evaluating the performance degradation caused by single synapse faults (Sec. 5.3.3), a different setup is necessary. A successfully trained LSM configuration, i.e., the `HNetData` object with the configuration of the liquid plus the weights of the trained linear classifiers, is handed over to a new algorithm that successively introduces faults to the weight configuration of the liquid. After each introduced fault the algorithm evaluates the performance of the LSM on the generalization set without retraining the linear classifiers. This `HLSMCorrosionAlgo` algorithm as well fits itself into the HANNEE algorithm framework and uses the dedicated interface for chaining algorithms. It extends the `HCorrosionAlgo` class—which provides the fault functionality—with specific functionality to export the liquid data for further analysis. For comparative fault studies with networks derived by other algorithms the `HCorrosionAlgo` is used directly.

Chapter 5

Experimental Results

As elaborated in the previous chapters, the neural network experiment is suited to explore the liquid computing approach in hardware. In a first set of measurements this chapter shows the feasibility of computing without stable states on the presented HAGEN prototype and characterizes liquid filters resulting from varying the input driven network topology. The second set of measurements is dedicated to exploring the promise of liquid computing being a hardware-friendly strategy in terms of tolerance to substrate variations and robustness to faults occurring during operation.

5.1 General

The *liquid*HAGEN setup described in Sec. 2.2.2 is the platform for the experiments presented in what follows. Accordingly, the liquid filter is physically realized on the HAGEN prototype by a recurrent network of an input-driven type (c.f. Sec. 2.3.1), while the readout as well as the evaluation is performed in software; for details see Sec. 4.4.3.

Unless otherwise stated, all measurements use the same settings for the hardware, which in the following are: $I_{\mathrm{syn}}^{\mathrm{max}} = 22\ \mu\mathrm{A}$; $V_{\mathrm{cas}} = 2.9$ V, $V_{\mathrm{bias}} = 0.91$ V, $V_{\mathrm{casdac}} = 2.9$ V; the HAGEN interface speed is 88 MHz. The clock pattern given in the appendix B.4 is used.

5.2 Exploring a Hardware Liquid

5.2.1 Initial Experiments

Liquid computing allows the harnessing of the capabilities of recurrent neural networks by only having to train a readout. This readout may be as simple as a linear classifier or a perceptron if the liquid provides the appropriate non-linear projections and short-term memory. Consequently, by restricting oneself to a linear readout one can assess these very properties of the liquid.

While the theoretical framework assumes the idealized computational power of the readouts, an experimental implementation has to rely on the actually realizable implementations. These may differ from their ideal counterparts in their finite size or in the achievable accuracy. An experiment

comprised of several levels of hardware in interaction with software furthermore needs to be validated for its correct implementation. It is thus necessary to confirm that an actually implemented readout does not limit the liquid state machine realization beyond its inherent limitations.

The Learning Tool maintained by the group of Maass [148] and introduced in Sec. 4.4.3 has been primarily used for experiments with liquids comprised of spiking neurons [131]. Yet, it is possible to adapt the framework to input-driven threshold networks with binary inputs and neuron firing states [21]. As described earlier, an export/import tool chain has been developed in order to use the Learning Tool in conjunction with liquids implemented on the HAGEN prototype. This allows the verification of the *liquid*HAGEN setup and a cross-check of the native software implementation of the linear classifier used for the exhaustive parameter sweeps in the next section.

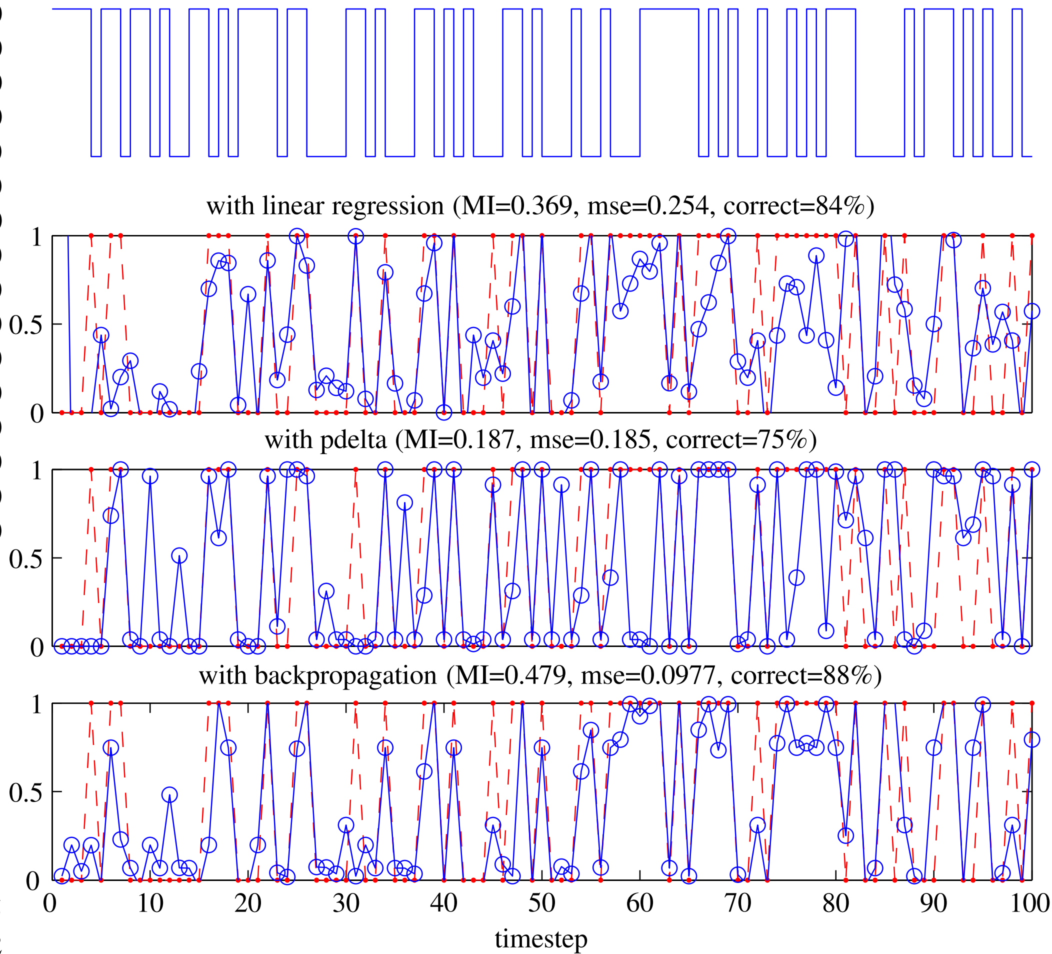


Figure 5.1: Visualization of the response of three different types of readout observing for 100 time steps the same liquid driven by a binary time-discrete input (top row). The target filter is the 3-bit time-delayed parity with $\tau = 3$ (dots connected by the dashed (red) line). The actual (binary) prediction is derived by thresholding the given responses with 0.5. All readouts have been trained on the same 1000 step training set. The liquid parameters are $N = 256$, $k = 6$, $\sigma^2 = 0.14$, $\overline{u} = 0$, $u_{\text{in}} = \pm 0.5$.

In an initial experiment, an input-driven network has been realized on the HAGEN prototype with a parameter set close to the critical line predicted by the mean-field theory in Sec. 2.3.2: All $N = 256$ neurons of the ANN ASIC are used, the number of incoming connections per neuron is

chosen to be $k = 6$, and the corresponding weights are drawn from a zero-centered Gaussian of $\sigma^2 = 0.14$ with weights in $[-1, 1]$. For the input, a random ($r = 0.5$) binary time-discrete $u(t)$ is used which is fed to each neuron with the weight $u_{\text{in}} = \pm 0.5$; $\overline{u}$ is set to 0; see Sec. 2.3.1 for details. The input stream is partitioned into four chunks of 500 consecutive bits; the resulting liquid states for two of these chunks are used for training, while the other two are used as the generalization test set. The HAGEN prototype is configured for time-varying inputs according to Sec. 4.2.2.

readout method	mutual information	mean average error	percent correct
linear regression	0.40 ±0.01	0.147 ±0.003	85.3 ±0.3
pool of perceptrons, p-delta	0.21 ±0.02	0.24 ±0.01	76.2 ±1
MLP, back-propagation	0.38 ±0.07	0.19 ±0.04	81.4 ±4.1

Table 5.1: Performance of the three different readout methods for 10 repetitive training runs on the same liquid. Given are the mean values and the standard deviation of the mean.

Fig. 5.1 shows the input $u(t)$ for a window of 100 time steps (top row) and the response of three different types of readouts (bottom three) observing a liquid randomly generated according to the parameters described above. The readout shown in the second row is obtained by a linear regression as described in Sec. 2.2.2 with the thresholding of the linear classifier not yet performed. The target filter is 3-bit time-delayed parity for $\tau = 3$, i.e., the readout is trained to answer at any time t whether an odd or even number of ones has been presented to the liquid at times $t-3$, $t-4$, $t-5$.; see Eq. 2.15. The target values are plotted by dots and interconnected by dashed (red) lines. The third row shows the output of a pool of 51 parallel single-layer perceptrons trained by a generalized delta rule (p-delta) [12]. This readout was initially used by Maass et al. to obtain the speech benchmark results in [131] and networks of this type have been shown to be universal approximators [13]. Finally, the bottom row yields the result of an MLP with sigmoidal activation function of the neurons, 5 hidden units, and back-propagation training. For training details of the latter two see [155].

PSfrag replacements

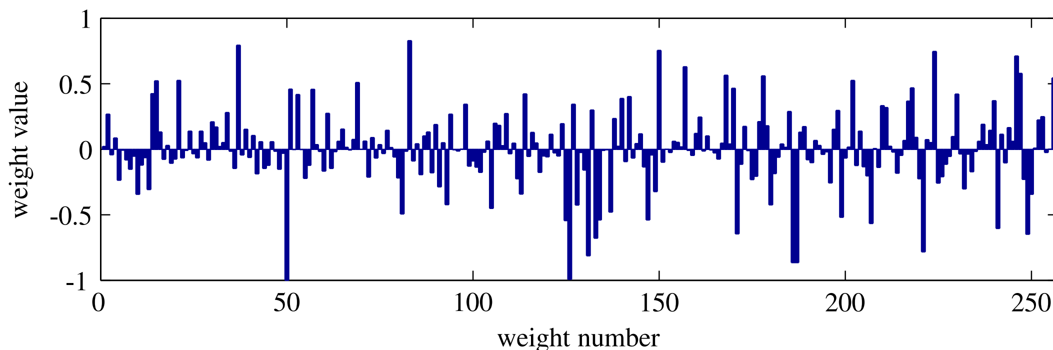


Figure 5.2: Normalized weight vector of the linear readout computed by the linear regression for the LSM shown in Fig. 5.1 and Fig. 5.3.

In the titles of each readout plot are given the mutual information according to Eq. 2.13, the mean squared error between the target and prediction values, and the percentage of correct responses; all measures are, for illustration, computed for the shown subset of 100 time steps and after thresholding the readout responses with 0.5. It is to be noted that the first four target values are in fact undefined since only starting with the fifth input there is the full information of the time-delayed parity task available. For a correct evaluation therefore either the leading time steps

have to be discarded or the number of consecutive steps has to be chosen large enough so that the initial values have a negligible contribution.

PSfrag replacements

For numerical reasons, the liquid states on which all three readouts get trained on, are rescaled to yield a zero mean and Gaussian noise with a sigma of 1% of the maximum resulting liquid state component is added. Repetitive training runs on the same liquid therefore yield varying performance. The results of Fig. 5.1 only show a single run and only the computed performance for a subset of 100 time steps of the generalization set. In order to compare the different readout performances it is thus necessary to evaluate repetitive training runs on the same liquid but with regenerated noise for each run. Tab. 5.1 shows the results for 10 repetitive runs obtained on the 1000 time step generalization set.

It can be seen that for the presented liquid and task, the performance of the linear regression readout is very competitive to the MLP with back-propagation training in terms of the mean squared error, i.e., the accuracy in approximating the absolute target value, as well as in correct predictions and mutual information. The smaller standard deviations in the mutual information for the linear regression suggests that it reliably yields good results while the back-propagation training shows a larger spread. The performance of the pool of perceptrons trained with the p-delta rule consistently shows poorer performance compared to the other two readouts. Since none of the readouts is especially tailored to the task and it is known that the p-delta rules needs appropriate fine tuning [155], the worse performance is likely due to non-optimal settings.

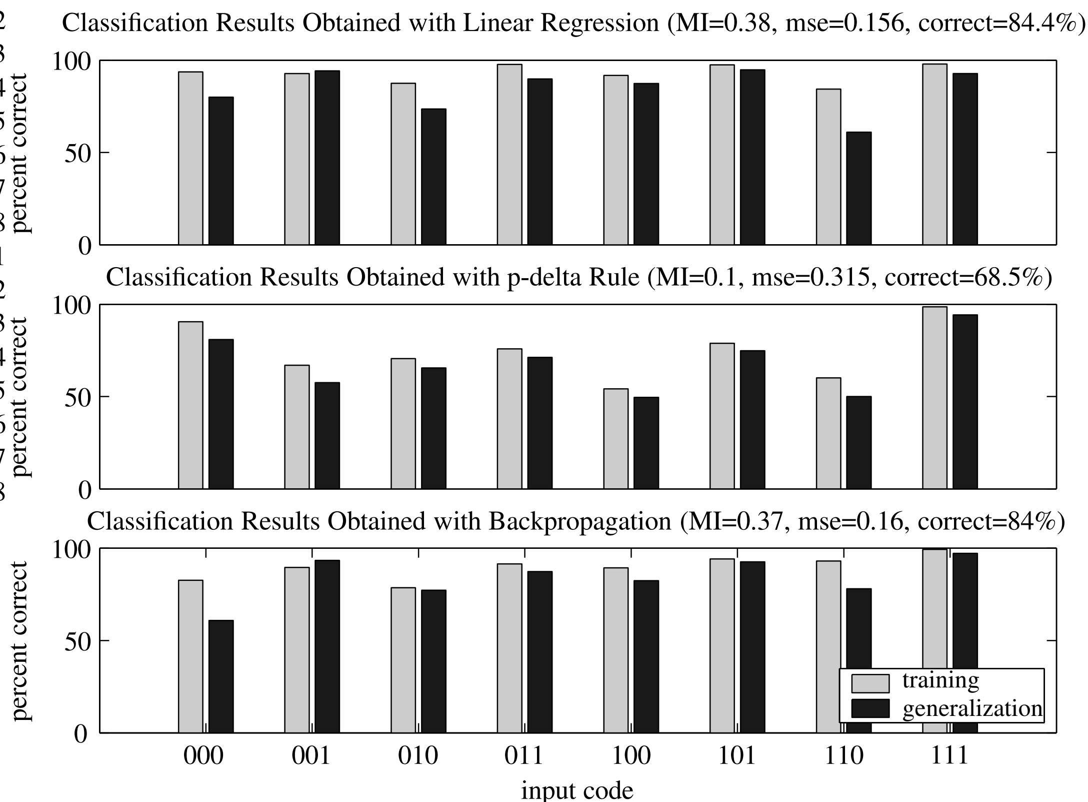


Figure 5.3: Performance of the liquid state machine partly visualized in Fig. 5.1; now the complete training (light gray) and generalization input (dark gray) is evaluated. The three rows show for each readout strategy the percentage of correct target predictions in dependence on the input code. In each title the performance figures for the generalization set are shown.

An important observation is that the linear readout is capable of extracting a target filter which is non-linearly dependent on the input; accordingly, the liquid is acting not only as a memory but also as a kernel. The tie in performance between the linear readout and the MLP indicates that here probably the memory capability of the liquid is the limiting factor which prevents the MLP from being able to perform better. As discussed in Sec. 2.1.2, it is clear that there can exist liquids that primarily act as memories and not as kernels. In such cases, an MLP readout would necessarily outperform a linear regression readout on a 3-bit parity task. These initial experiments are not meant to exhaustively compare different readout strategies but rather to illustrate that the computationally moderate linear regression yields robust and consistent results. They are additionally used to verify the implementation of the *liquid*HAGEN setup. Fig. 5.2, for example, shows the normalized weights of the linear readout used above, produced by the linear regression. From the relative weight values it can be seen that the linear readout utilizes many components of the liquid state vector. Comparisons of weights computed with the Learning Tool were used to verify the numerical implementation of the native readouts used in the next section.

Other aspects of the *liquid*HAGEN setup can be validated as well: Fig. 5.3 assesses the performance of the experiment which was partly shown in Fig. 5.1. Here, the responses for the three different readouts are shown for the complete training as well as the generalization set. The histograms show the percentage of correct responses by input code. From this it can be seen that the non-optimal performance is not systematic to a specific input code but more likely caused by the overall fading of the information in the liquid.

5.2.2 Observing the Edge of Chaos

The intention of the liquid state machine experiments presented here is somewhat different from the software simulations of input driven networks performed by Bertschinger and Natschläger [21, 151]. There, the authors simulated liquids of input driven networks in order to show the consistency with their mean-field theory and to explore the predicted phase transition between order and chaos for various parameters. Here, the focus is on assessing the properties and performance of a physically realized liquid state machine and thus to explore whether liquid computing is suited for a hardware realization. The results of this section have in parts been published in [195].

The following set of experiments starts by using the HAGEN prototype in normal operating conditions: The DAC and neuron offsets are compensated which allows the use of precalculated weights ([86] and see the results on the variable network resources in appendix D). Yet, mapping an input driven network to the hardware implies certain modifications: For example, the dynamic range of the weights is fixed by the achievable relative accuracy and the maximum weight value. Furthermore, the dynamic range of the neurons is limited as discussed in Sec. 3.3.3. Additionally, if the HAGEN prototype is operated as a recurrent network with many network cycles, the effect of the degrading weights (see Sec. 3.3.2) has to be considered. While this is a central aspect in the set of experiments that examines the suitability of liquid computing under non-ideal conditions presented later (see Sec. 5.3), here the refresh cycles are chosen such to minimize the effect.

In order to ensure these ideal operating conditions, systematic examinations of the respective technical parameters have been performed and are documented in the appendix E. For example using an input weighting of $u_{\text{in}} = \pm 0.5$ adjusts the interesting region in the k-σ^2 plane to an interval of σ^2 that can well be realized in the normalized weight range of $[-1, 1]$ (see Fig. E.1). Similarly, using symmetric input yields an overall better readout performance than just using a single input (see Fig. E.2); this is the reason for the input driven networks used here being a mixture of the type with fully symmetric neurons [21] and the fully asymmetric case [151] which was described in Sec. 2.3.1. Lastly, the dependency on the number of training and generalization patterns has

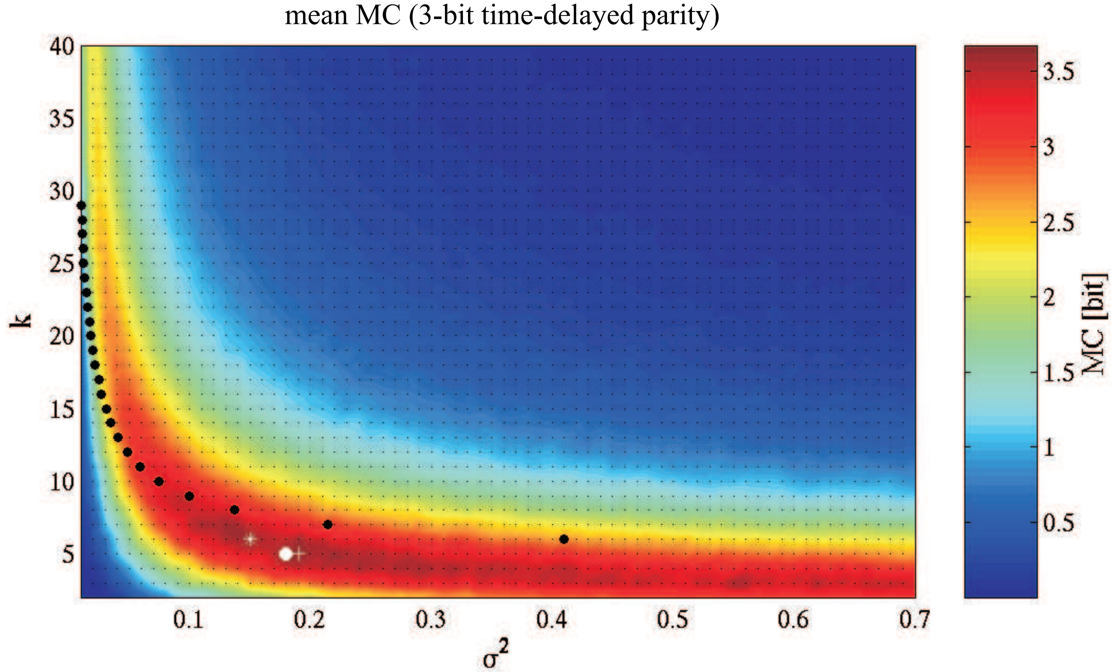


Figure 5.4: Parameter sweep in the liquid generation parameters k and σ. The color code visualizes the achieved memory capacity (MC) by linear classifier readouts in extracting the 3-bit time-delayed parity task from the respective liquids. Shown is the mean MC of 30 different liquids per data point (small black dots); the intermediate values are interpolated. The largest three mean MCs are marked in order with a white dot, white asterisk, and white plus. The large black dots indicate the prediction of critical dynamics by the mean-field theory.

been evaluated and led to the use of 4000 time step[1] input streams for the training and 8000 time steps for the generalization. The weights are refreshed between training and generalization. The input is randomly generated with $r = 0.5$.

While Bertschinger and Natschläger extensively study phase transitions by introducing asymmetries in the input ($\overline{u} \neq 0$), the purpose of the here presented experiments is to sweep parameters that are relevant for a hardware implementation: These are the number of incoming connections per neuron k and the magnitude of the respective weights relative to the input governed by σ^2.

The experiments presented in the preceeding section showed the results of a single LSM, i.e., one or more readouts were trained to extract the same target filter from one liquid. To characterize the dependency on the liquid generation parameters, it is necessary to train readouts for liquids generated with different parameters. Furthermore, it is desirable to choose a task that gives some indication of the kernel as well as the memory properties of the liquid. Therefore, the 3-bit time-delayed parity task as defined by Eq. 2.15 in Sec. 2.3.3 is used yielding the memory capacity MC for a given liquid. Due to the random nature of the liquid generation, training readouts on the time-delayed parity task should be repeated for several liquids resulting from the same parameter set.

[1]This number marginalizes the effect of the initial few time steps for which the target filter is not defined; see Sec.5.2.1.

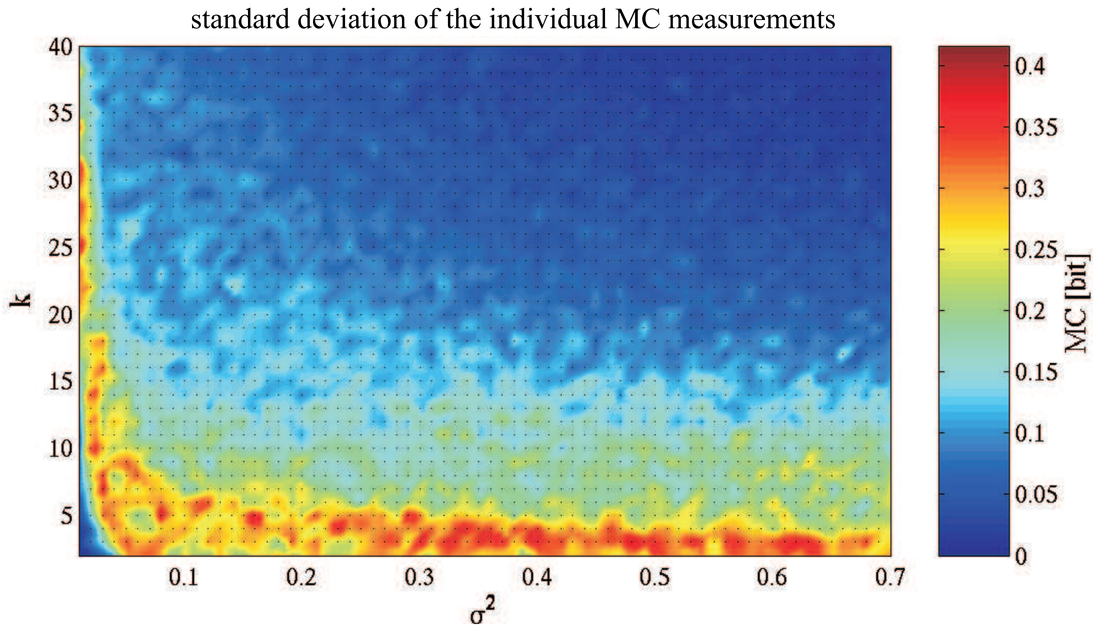


Figure 5.5: For each measured parameter set in Fig. 5.4, 30 liquids have been generated and appropriate readouts trained. This figure shows the standard deviations of the individually measured MC at each parameter set. Again, the small black dots indicate actual measurements; intermediate values are interpolated.

Fig. 5.4 shows the mean memory capacity at parameter sets k, σ^2 at each of which 30 different liquids have been generated and respective linear classifiers have been trained to extract 3-bit time-delayed parity. The MC value is encoded in color according to the color legend. Intermediate values are derived by a cubic interpolation of the actually measured data points (indicated by small black dots). Even though the number of inputs per neuron, k, is discrete, fractional values may be interpreted as the average number of connections in a network where some neurons have more connections than others. The largest three mean MCs are marked in order with a white dot, white asterisk, and white plus. The standard deviations for the distributions of the memory capacity measured at each parameter set are quite small (all below 0.45 bit) as shown in Fig. 5.5. Accordingly, the standard deviation of the given mean MC values, which is the plotted standard deviation of the individual measurement divided by the square root of the number of runs (30), is below 0.1 bit. The peak of the readout performance along a hyperbola-like band in the k-σ^2 plane is therefore significant.

The mean-field theory introduced in Sec. 2.3.2 predicts a phase transition from ordered to critical dynamics in the k-σ^2 plane, as indicated by the large black dots[2] in Fig. 5.4. The prediction of critical dynamics coincides quite well with the observed peak in readout performance. Yet, for small k and large σ^2 the readout seems to be more successful in extracting the target filter for slightly more ordered dynamics, i.e. below the critical line. For large k and small σ^2 the readout seems to be more successful slightly towards chaos, but the overall performance is worse for the latter case.

[2]The formalism of the mean-field theory assumes integer values of k; the interpretation of fractional values being the mean connectivity of an ensemble is legitimate but respective theoretical values cannot readily be computed. Therefore, the 'critical line' is indicated by dots.

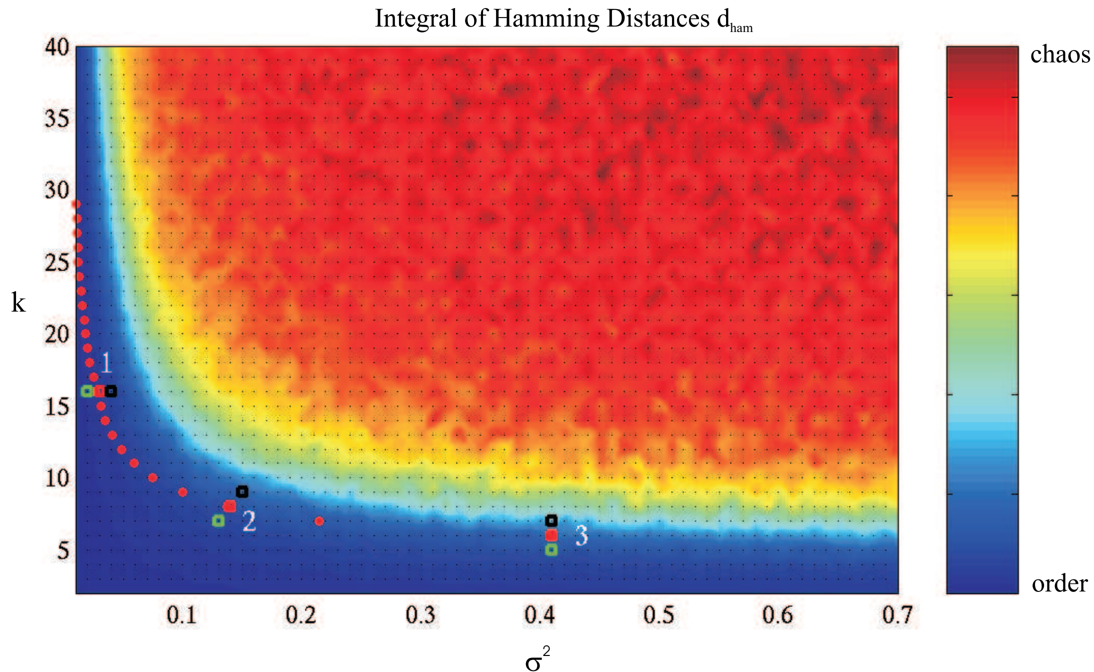


Figure 5.6: Visualization of the mean integral Hamming distances for liquids generated according to the swept parameters k, σ^2. The values are derived by generating 10 liquid per parameter set (small black dots) and driving each liquid by 50 pairs of initially different and eventually identical input streams. If the transient Hamming distance decreases, which yields small values of the integral Hamming distance, the dynamics are said to be ordered. Accordingly, large values coincide with growing or non-decaying Hamming distances and indicate chaotic dynamics. The plotted measure has arbitrary units as explained in Sec. 2.3.3; intermediate values are interpolated. The red dots are the prediction of critical dynamics by the mean-field theory. The indicated parameter sets 1,2, and 3 are plotted in detail in Fig. 5.7.

In order to interpret this observation, it has to be verified that the prediction of the mean-field theory is accurate for the input-driven networks realized here. This can be done experimentally by measuring the time development of the Hamming distance as introduced in Sec. 2.3.3. Essentially, a liquid is evaluated for two input streams that are initially different and thus cause different liquid states, and eventually the input streams become identical which then causes the distance between these liquids to decay or grow; the dynamics then is assigned to be ordered or chaotic accordingly. The integral of these Hamming distance time courses as defined in Eq. 2.18 yields a measure for the separation of the liquid. Fig. 5.6, for 10 liquids per parameter set and 50 pairs of input streams which are different for 25 time steps and identical for 50 time steps (see Eq. 2.17), the mean integral Hamming distance is plotted for the k-σ^2 plane. Since the integral Hamming distance has arbitrary units, it cannot directly predict the critical line. Yet, clearly visible are sets of adjacent parameters that yield the same mean Hamming distance. These equipotential lines are in good agreement with the prediction by the mean-field theory.

At the indicated points in Fig. 5.6 with numbers 1,2, and 3, Fig. 5.7 shows the detailed transients of the mean Hamming distance for a parameter set below (green), right on (red), and above (black) the predicted critical line. The given errorbars indicate the standard deviation of the mean; for visibility they are only shown for every tenth value. The green lines are predicted to be correlated with ordered dynamics and thus should decay to zero. The red ones are associated with

PSfrag replacements

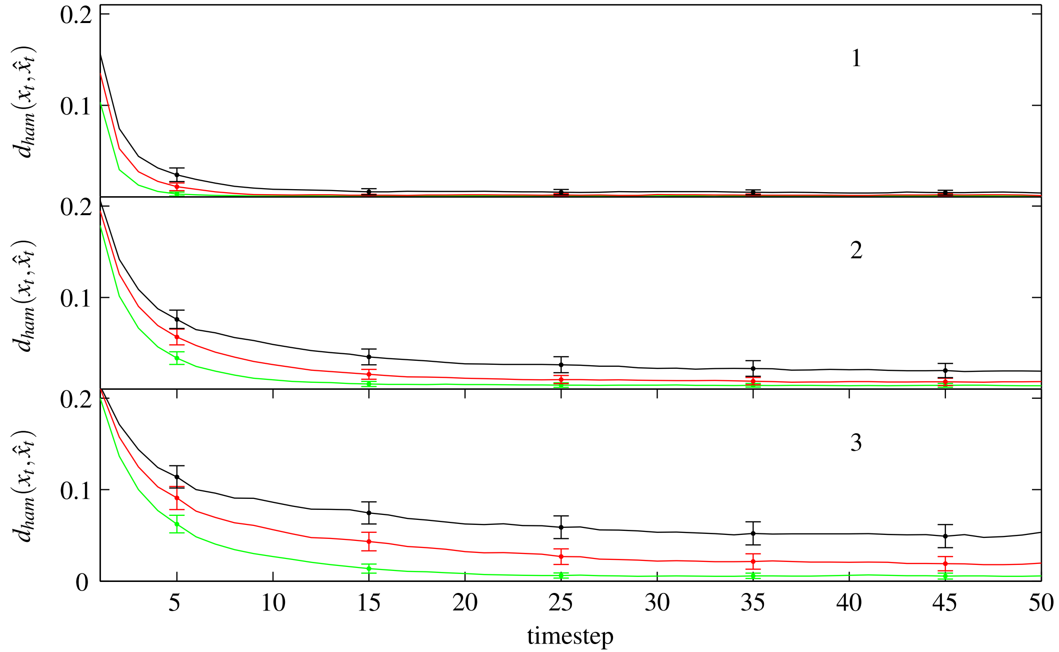


Figure 5.7: Time courses of measured mean Hamming distances for different parameter sets. In each plot, the colors correspond to parameter sets with predicted ordered dynamics (green), critical dynamics (red), and chaotic dynamics (black). The numbers finally indicate the region in the k-σ^2 plane the liquids were drawn from as indicated in Fig. 5.6.

critical dynamics and therefore should asymptotically approach zero as well, yet later than the green ones. Only the black ones should asymptotically be non-zero since they indicate chaotic dynamics. This behavior is well observed in the experimental data. Only for small k and large σ^2 the mean-field theory seems to underestimate the chaos, i.e., the predicted critical dynamics already shown signs of chaos as can be seen from the bottom plot. There the predicted ordered and critical dynamics show a non-zero asymptotic value. Yet, especially the Hamming distances in the bottom plot have a noticeable spread as can be inferred from the errorbars. The same liquid for different input streams as well as different liquid generated from the same parameter set seem to exhibit chaotic dynamics in some cases and ordered in others. This can be understood from Eq. 2.9: Few large (compared to the weight of the input) incoming connections to a neuron may be drawn in a configuration which in the extreme cases always cause a flip of the neuron or never; an individual liquid drawn for small k, large σ^2 therefore is less predictable in terms of the dynamics it will have. This is further supported by the large standard deviations for the MC measurements observed in this region (Fig. 5.5).

This explanation is in agreement with the observations of the readout performance in Fig. 5.4. For small k and large σ^2 the linear classifiers are most successful in extracting information for parameter sets that more reliably yield ordered dynamics. On the other hand, for large k and small σ^2 the readout tends to favor slightly more chaotic dynamics. This behavior of networks with higher connectivity may be related to observation made for liquids of spiking neurons [129]. There, the readout performance similarly peaks for slightly more chaotic dynamics.

A close-up sweep in Fig. E.3 of the appendix shows that the decrease in the overall performance along the critical line towards small σ^2 is only partly due to the finitely scanned σ^2. The decrease is likely to be caused by the finite accuracy of the hardware liquid due to single-synapse

offsets as well as temporal variations which effectively increase the variance of the weight distribution. Sec. 5.3.1 and Sec. 5.3.2 will examine the respective properties of the hardware liquid.

In order to examine whether the readout performance may yet be dependent on the target filter, a similar sweep to the one presented in Fig. 5.4 has been performed. However, instead of using the 3-bit time-delayed parity task, 50 linear classifiers are trained to simultaneously extract from the same liquid 50 randomly drawn Boolean functions that are dependent on 5 consecutive time steps. Again, the generation parameters k and σ^2 are swept; at each parameter set ten liquids have been generated and for each of the ten liquids 50 readouts get trained. The appropriate performance measure is the mutual information since the target filter are not shifted in time (see Sec. 2.3.3).

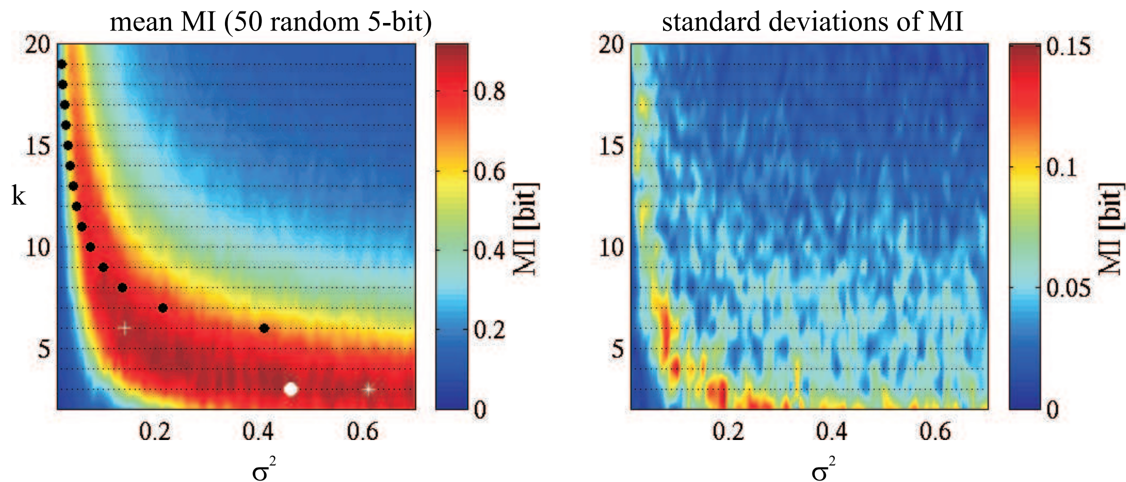


Figure 5.8: Left: Parameter sweep in the liquid generation parameters k and σ^2. Plotted in color code is the mean mutual information of 50 linear classifiers extracting simultaneously 50 randomly drawn 5-bit Boolean target functions. Small black dots indicate parameter sets at which ten liquids were drawn and the respective classifiers trained for each; intermediate values are interpolated. The large black dots indicate the theoretical predication of the critical dynamics. **Right:** Plotted are the standard deviations of the individual MI measurements at each parameter set.

The left part of Fig. 5.8 shows the mutual information resulting from averaging over the individual performances of the classifiers for all 10 runs per parameter set. The actual measurements are indicated by small black dots; intermediate values are interpolated. The large black dots indicate the mean-field prediction of critical dynamics. The right hand side of Fig. 5.8 shows the standard deviations of the individual MI as given by all observed classifier performances per parameter set. Since the standard deviation of the mean is smaller by the square root of the number of runs, it can be concluded that the observed peak in the mutual information along the hyperbola-like band is significant. If compared to Fig. 5.4, it can be seen that readout performance is essentially independent of the trained target filter. This is in agreement with the simulation experiments of Bertschinger and Natschläger [21, 151].

As postulated by the authors there, the presented measurements give strong evidence for the readout performance being optimal in the vicinity of critical dynamics for LSMs comprised of input driven networks and linear readouts. While it has been shown here for the k-σ^2 plane, in [21, 151] this correlation has been observed for the $\bar{u}$-σ^2 and μ-σ^2 phase transitions[3]. In all cases the mean-field theory is capable of accurately predicting the phase transition from ordered

[3] μ is introduced as the mean of the Gaussian distribution from which the weights are drawn.

to chaotic dynamics. It can therefore be used to derive mechanisms that adjust the dynamics to the critical regime as proposed in [151]. A hardware implementation on the other hand may favor certain parameter regions due to its connectivity, dynamic range, and noise.

5.2.3 Scaling Behavior

The dynamics of an input driven network are governed by the number of incoming connections per neuron k and the probability of a neuron being flipped by one of these incoming connections, which is essentially controlled by σ^2 and the weight of the input $\overline{u} \pm u_{\text{in}}$; see Eq. A.17 in Sec. 2.3.2. The phase transition between ordered and chaotic dynamics is thus independent of the total number of neurons N. Yet, the number of neurons increases the dimensionality of the liquid state. According to the fundamental considerations on liquid computing in Sec. 2.1.2 the projection into a higher dimensional state is likely to ease the task of the linear readout in extracting filters that are non-linear functions of the inputs.

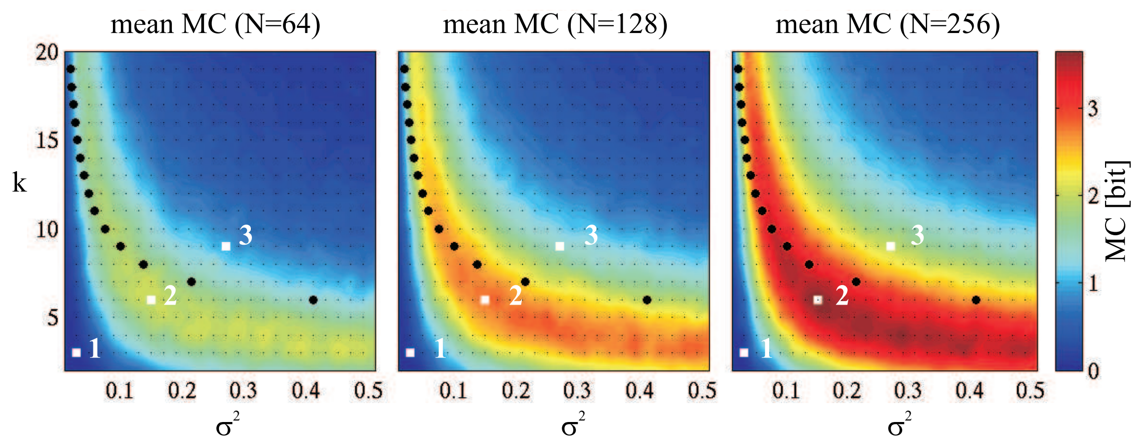


Figure 5.9: Set of experiments in which the number of neurons N in the input driven network is doubled in each step: 64, 128, and 256 neurons from left to right. Shown is the mean memory capacity for the 3-bit time-delayed parity task in a parameter sweep of k and σ^2 (similar to Fig. 5.4). Two findings can be stated: (1) The peak in readout performance is independent of N as can be seen from the relative location to the predicted critical dynamics (large black dots); (2) The readout performance grows with the number of neurons.

It can therefore be expected that the readout performance of an LSM with an input-driven network liquid scales with the number of neurons therein. In [21] this has been shown for simulated networks. Fig. 5.9 shows a respective set of experiments in the *liqduid*HAGEN setup for the 3-bit time-delayed parity task. Shown is the mean memory capacity as measured in a parameter sweep of k and σ^2 and 30 different liquids per parameter set. From left to right the number of neurons in the liquid is set to 64, 128, and 256. These numbers coincide with the use of one, two or four network blocks. The right most plot is equivalent to the plot in Fig. 5.4 yet with less measurements per parameter set and less parameter sets evaluated. The predicted critical dynamics are plotted by large black dots. The essential observations of these plots are: First, already in the 64 neuron case the mean-field prediction of critical dynamics is in good agreement with the measured peak in readout performance. Second, the readout performance becomes better with increasing N.

The first observation justifies picking out individual parameter sets and plotting the achieved readout performance for increasing N. This is done in Fig. 5.10 for parameter sets from the ordered, critical, and chaotic regime. Plotted is the mean memory capacity and the standard de-

PSfrag replacements

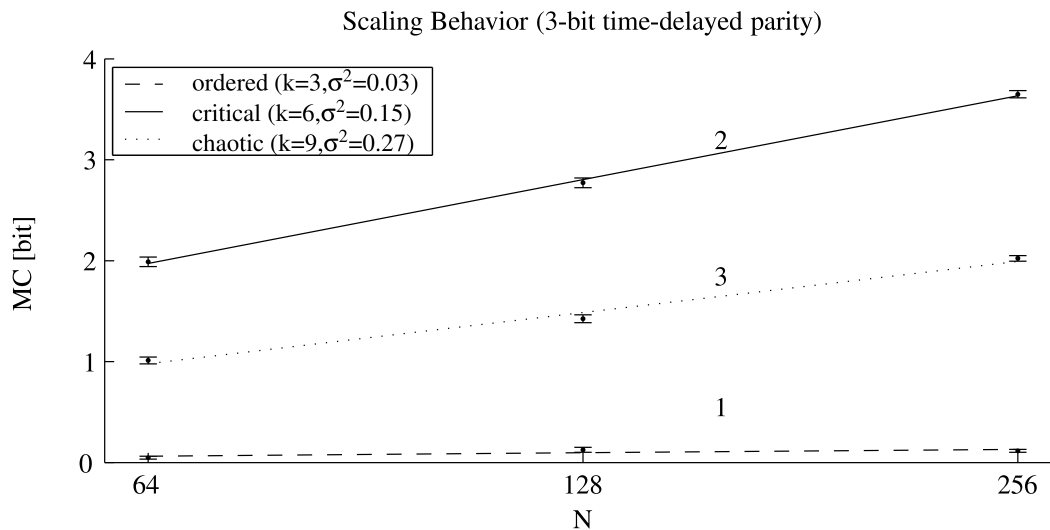


Figure 5.10: Readout performance versus the number of neurons N in the liquid for the 3-bit time-delayed parity task. Plotted are the mean values of 30 liquids generated per parameter set. The parameter sets are chosen from the ordered (dashed line), critical (solid line), and chaotic (dotted line) regime. The given errorbars indicate the standard deviation of the mean. The connecting lines are a least-squares fit.

viation of the mean is given by the errorbars. It can be observed that the readout performance for critical dynamics exhibits the steepest increase while readouts trained on ordered or chaotic dynamics show a weaker increase. The almost logarithmic scaling with the number of neurons in the network is in agreement with the observations in [21].

On the one hand, the observed scaling shows that using input driven networks as liquids allows an immediate gain in achievable readout performance simply by increasing the number of neurons. Since the *evaluation* of the readout only scales linearly with the number of neurons, this can well be realized. On the other hand, the only logarithmic dependency clearly burdens the applicability of liquid filters generated according to the above input driven network strategy, especially since the computation of the linear regression during training has a cubic dependency on the number of neurons. The implications of this will be considered in the closing discussion.

5.3 Fault-Tolerance

5.3.1 Robustness to Substrate Variations

The experiments performed so far were conducted with normal operation conditions of the HAGEN prototype, i.e., the compensation of the neuron and DAC offsets being activated. It was thus possible to show very good agreement between the measured dynamics and the theoretical prediction. Yet, as it was motivated in Sec. 2.2.1, the liquid computing approach promises an inherent tolerance to variations in the substrate. If for example the synapse offsets were Gaussian distributed with some sigma σ_{synoff}, it would suffice to activate an appropriate number of inputs k as predicted by the mean-field theory and critical dynamics are readily achieved. Even if there were a systematic shift in the mean of the Gaussian distribution, appropriate dynamics can be predicted [151].

With the help of the calibration procedure introduced in [86] and shortly described in Sec. 4.4.2, the offsets present in a HAGEN ASIC can be measured. Fig. B.3 in the appendix yields the results obtained for the die used here. It can clearly be seen that the neuron offsets are the dominating effect with a standard deviation of the individual measurement $\sigma_{\mathrm{neuroff}}$ of approximately 100 LSB or about 10 % of the dynamic range of a synapse (less than 1 % of the dynamic range of the neuron itself). The measured spatial synapse offsets in the HAGEN prototype yield a individual standard deviation σ_{synoff} of 2.3 LSB which is 0.23 % of the dynamic range of a single synapse. The DAC offsets resulting from the bias currents (c.f. Sec. 3.3.4) are necessarily positive and exhibit a mean $\mu_{\mathrm{dacoff}} \approx 27$ LSB and a standard deviation of $\sigma_{\mathrm{dac}} \approx 12$ LSB.

In the course of the preceeding experiments, the individual synapse offsets have not been calibrated. If the width of the synapse offset distribution ($\sigma^2_{\mathrm{synoff}} = 0.0023^2 \approx 5 \cdot 10^{-6}$) is compared to the examined weight distributions $\sigma^2 = 0.01 \ldots 0.7$ it becomes clear that the effect is negligible. Only the neuron offsets are in the order of magnitude of the smallest examined weight distributions. The results of the appropriate experiments are shown in Fig. 5.11. The left plot shows the achieved mean memory capacity for a k, σ^2 parameter sweep of the HAGEN prototype with DAC and neuron offsets compensated. On the right hand side, the same die is used without any calibration. The target filter is 3-bit time-delayed parity; 30 liquids per measured parameter set have been evaluated. Large black dots in both cases indicate the mean-field prediction of the ideal case.

The most important observation is that for a large part of the swept parameter space no degradation in performance is observed. Yet, for small σ^2 it can be seen that the peak in readout performance along the critical line is slightly shifted to the left. This becomes more clear in the difference plot shown on the left hand side of Fig. 5.12. It visualizes the memory capacity obtained by subtracting the mean values of the uncalibrated experiment from the calibrated:

$$\overline{MC}_{\mathrm{cal}}(k,\sigma^2) - \overline{MC}_{\mathrm{nocal}}(k,\sigma^2). \tag{5.1}$$

Blue color indicates a larger MC for the non-calibrated experiment; similarly, the red color code visualizes a larger MC for the calibrated case according to Eq. 5.1; intermediate values are interpolated after the subtraction.

The right hand side of Fig. 5.12 shows the difference plot of the calibrated experiments with a repetition of it for reference. While the right hand side exhibits the expected random deviation for the repetition of the same experiment, the left plot exhibits a systematic shift of the memory capacity towards small σ^2 (for large k): An explanation for this are changed dynamics of the liquid: The variation in the neuron offsets increases the effective variation of the nominally chosen weight distribution σ^2. For narrow weight distributions (small σ^2) this effect is necessarily stronger than for liquids drawn from the small k, large σ^2 region.

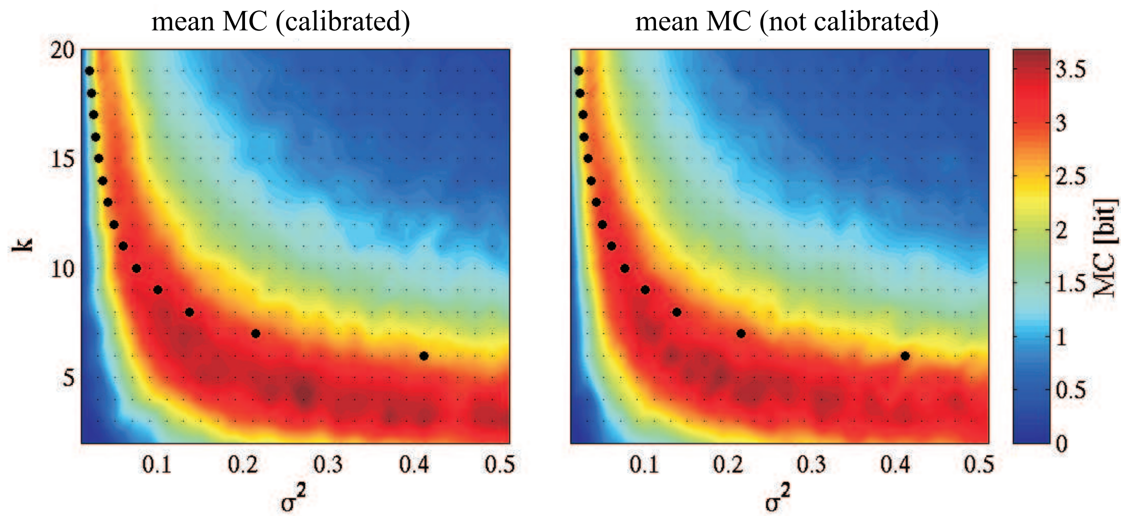


Figure 5.11: Readout performance achieved on a HAGEN prototype with compensated DAC and neuron offsets (left) and using the same HAGEN die without any compensation (right). While the overall performance is essentially identical, for large k and small σ^2 the readout peak in the uncalibrated experiment is shifted towards smaller σ^2. A difference plot of these two experiments is given in Fig. 5.12. At each measured parameter set (small black dots), 30 liquids have been evaluated; shown is the mean MC for the 3-bit time-delayed parity task. The large black dots in both plots indicate the prediction of the mean-field theory for the ideal case.

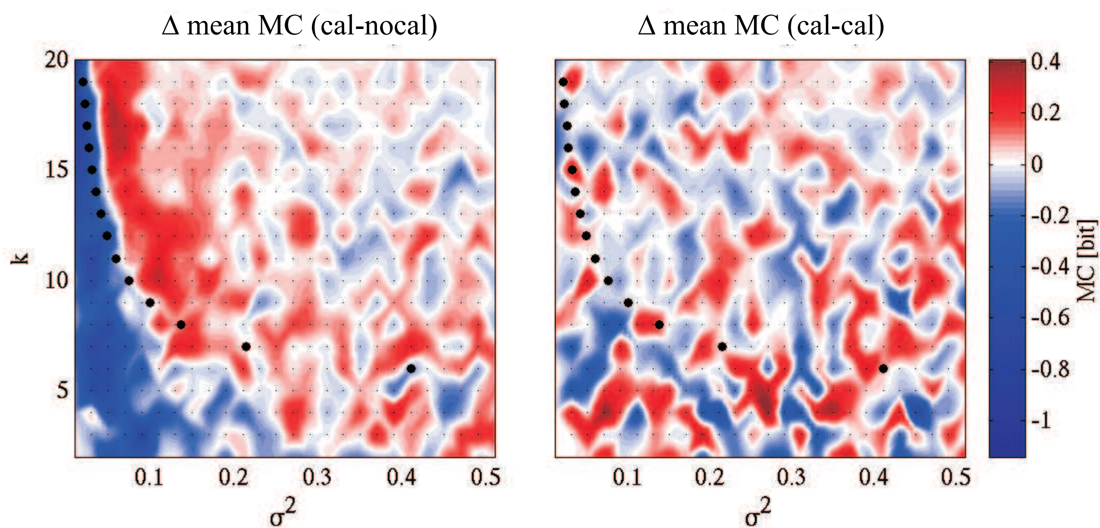


Figure 5.12: Left: Difference plot of the mean memory capacities achieved operating the same HAGEN die once calibrated and then uncalibrated. The individual parameters sweeps are shown in Fig. 5.11. Clearly visible is the shift in the dependency of the readout performance to smaller (nominal) σ^2 for large k. This is expected to result from effectively larger weights due to the variations in the neurons. **Right:** For reference, the calibrated experiment has been performed twice and the difference plot is shown. As expected, it shows a random pattern.

The results for the small k, large σ^2 region of the parameter space demonstrate that dynamics can be chosen appropriately in order to be tolerant to the exhibited hardware variations without a loss in readout performance.

5.3.2 Graceful Degradation with Time

The previous section dealt with static variations and showed that the chosen liquid computing approach robustly copes with the various offsets inherent to the HAGEN prototype. Another effect of the used hardware is the leakage of the weight capacitors as described in Sec. 3.3.2, which causes the weights to become larger over time. For input patterns that are not correlated in time such as the common classification tasks used for benchmarking neural networks [166], the weights may be refreshed in between the evaluation of input patterns. During the operation on time series, on the other hand, a weight refresh interrupts the processing. For the use with continuous real-time data streams therefore ultimately an architecture of mutually active network blocks needs to be employed as mentioned in Sec. 3.2.

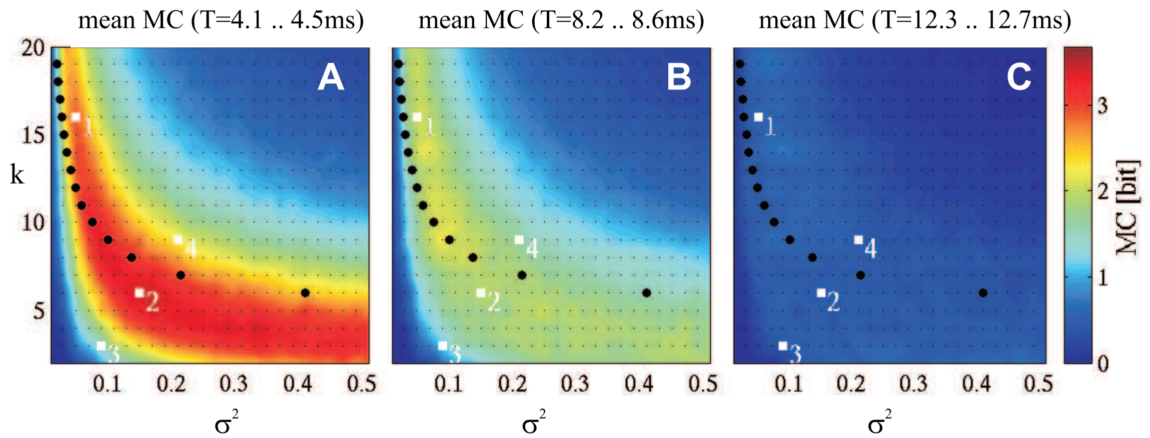


Figure 5.13: Three ‘snapshots’ of the time course of the mean memory capacity. From left to right (A to C), the ~ 0.4 ms time window for which the MC is calculated is shifted in time. The 30 liquids generated at each parameter set and trained readouts for the 3-bit time-delayed parity task are thus the same in the three plots, only evaluated at later times after a weight refresh.

The *liquid*HAGEN setup allows the examination of how the performance of an LSM develops over time if no weight refresh is performed. In a first experiment, input driven networks have been generated according to sweep parameters k and σ^2 with the remaining parameters identical to previous experiments: $N = 256$, $\overline{u} = 0$, $u_{\text{in}} = \pm 0.5$. Unless otherwise stated, the used benchmark task is 3-bit time-delayed parity. A 4000 time step input stream has been used for training. After a refresh of the weights, a 40000 time step generalization stream has been used as input. Since for the degradation over time, the absolute duration is of interest, in what follows the time step unit is replaced by its equivalent in seconds. Given the used interface frequency of 88 MHz, the clock pattern shown in Sec. B.4 of the appendix, and the operation mode that allows the programming of the inputs and readout in each network cycle for all 256 output neurons (c.f. Sec. 4.2.2), the 40000 time steps are equivalent to a time span of about 16.4 ms.

In order to describe the performance over time, the definition of the mutual information for a window in time as given by Eq. 2.13 is used. For input/output pairs from windows of about 0.4 ms (1000 input time steps) the mutual information is computed yielding an average over that

PSfrag replacements

time interval. For time-delayed tasks, the respective memory capacity is derived from summing all mutual informations computed for the given interval.

Fig. 5.13 gives three ‘snapshots’ of the time course of the mean memory capacity calculated for three different time windows 4.1-4.5 ms (A), 8.2-8.6 ms (B), 12.3-12.7 ms (C); these are 1000 time step long windows with the distance of 10000 time steps respectively. Note: Plot A essentially shows a performance about a millisecond later compared to performance plots shown before. About 8 ms after the weight refresh (B), the performance is noticeably decreased; another 4 ms later, as shown in C, the performance of the LSMs is very poor. It can be observed that the performance decrease is affecting all types of input driven networks regardless of the chosen generation parameters.

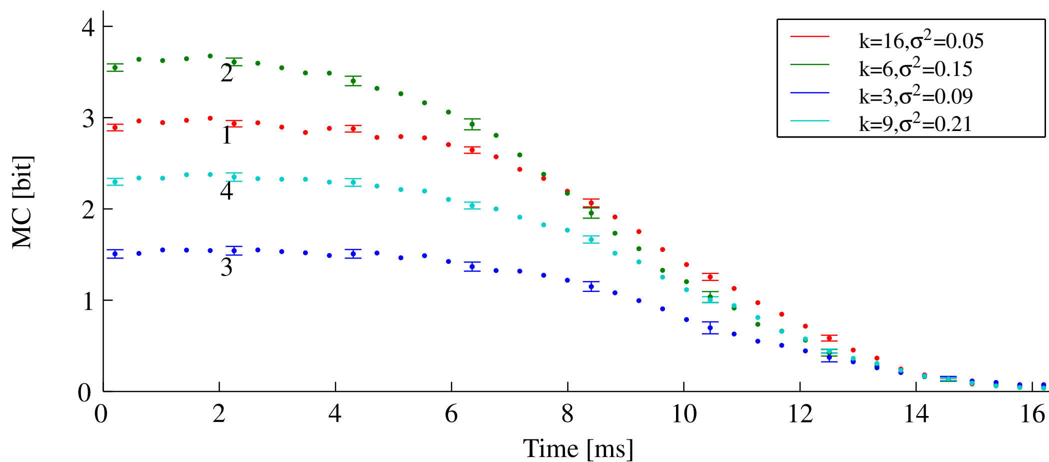


Figure 5.14: For four indicated parameter sets of Fig. 5.13, this figure plots the complete time course of the mean memory capacity in steps of about 0.4 ms. The given errorbars are the standard deviation of the mean (only shown for every fifth value). It can be seen that they all exhibit a initial plateau the height of which is dependent on the parameter set. MCs starting at initially higher values tend to decrease earlier, yet they all consistently loose their memory capacity completely after about 16 ms.

This can be seen in more detail in Fig. 5.14. For the four indicated parameter sets (numbers 1 to 4 in Fig. 5.13), the decay of the mean memory capacity for the 30 liquids generated at the respective parameter sets k, σ^2 is shown. Number 1 and 2 are in the close vicinity of the critical line (large k small σ^2 and medium k medium σ^2); parameter sets 3 and 4 are from the ordered and chaotic regime respectively. The mean memory capacity calculated for windows of about 0.4 ms is plotted as a dot in the middle of the respective interval; the given limits are the standard deviation of the mean. From the narrow errorbars it can be inferred that different liquids drawn from the same parameter set indeed show a very similar degradation over time. While the differences between liquids from different parameter sets are prominent in the absolute memory capacities, all liquids exhibit vanishing readout performance after about 16 ms. It can be observed that the MC for the liquids with medium k and medium σ^2, which shows the best performance of the four, is the first to exhibit a performance decrease (already after about 4 ms). In order to understand this, Fig. 5.15 breaks down the individual mutual informations of the readouts contributing to the memory capacity in the 3-bit time-delayed parity task, i.e., the figure shows the decaying performance of the readout extracting the parity for $\tau = 0$, $\tau = 1$ and so on; c.f. Eq. 2.15. The color code and numbering is the same to the one used in Fig. 5.14.

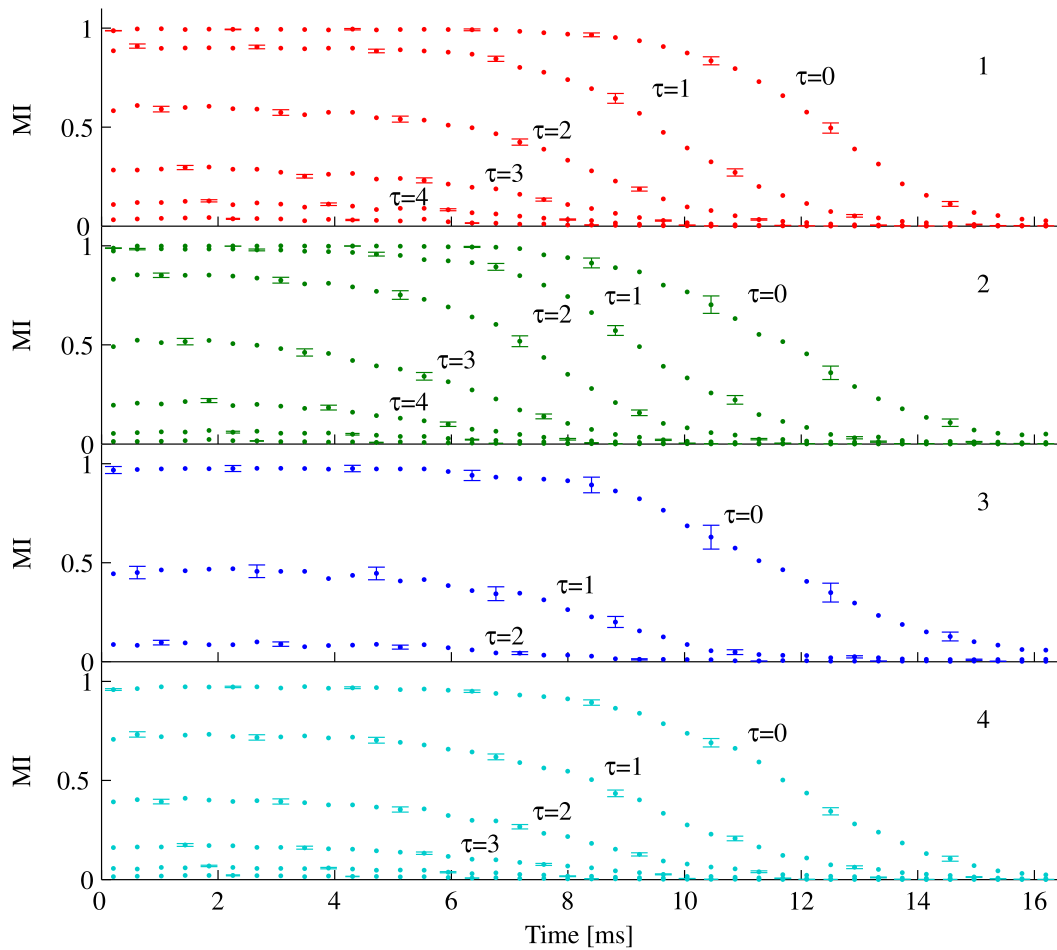


Figure 5.15: For the four parameter sets indicated in Fig. 5.13 and plotted in detail in Fig. 5.14, the performance here is broken down to the mutual information of the individual readouts extracting the 3-bit time-delayed parity task for $\tau = 0$, $\tau = 1$ etc. The sum of these for each time step is the memory capacity shown in the above figure. Shown is the mean mutual information and the errorbars show the deviation of the mean for every fifth value.

It can be seen from the decay of the mutual information that the observed earlier onset in decrease of the memory capacity for parameter set 2 ($k = 6$, $\sigma^2 = 0.15$) is primarily caused by the degradation of the readouts for larger τ, i.e., readouts that have to infer the correct prediction from input information more distant in the past. For parameter sets 3 and 4 the performance in these more difficult tasks ($\tau > 0$) is already poorer in the beginning or give no relevant contribution to the memory capacity at all: This is clearly visible in plot number 3 where the only noticeable performance is observed for $\tau = 0..2$.

In order to analyze this degradation further, two experiments are performed: First, the time course of the integral Hamming distance is measured for the k-σ^2 plane. This allows visualization of the change in dynamics of the liquid. In a second set of experiments, the readouts are not trained on an input stream of 4000 time steps but rather on shifted subwindows of a longer input stream, i.e., windows of about 0.4 ms (1000 time steps respectively) starting at times that have previously been used in Fig. 5.13 for the evluation of the generalization performance. Windows of the same length and shift are used on the generalization stream to test the performance. With this it can be

investigated whether the degradation is caused by liquid dynamics drifting to order or chaos, or merely by a change in dynamics that causes the readouts to extract wrong information.

Liquid Dynamics over Time Fig. 5.16 shows the mean integral Hamming distance as previously shown in Fig. 5.6 except that the Hamming distances are computed in the time windows 4.1-4.5 ms (plot A), 8.2-8.6 ms (plot B), and 12.3-12.7 ms (plot C). To accomplish this, a sufficient number of pairs of input streams are evaluated by the network, thus allowing the selection of pairs evaluated in the corresponding time windows. The individual pairs are comprised of a leading 50 input steps that are different and 50 input steps that are identical (c.f. Sec. 2.3.3). Per parameter set 30 liquids have been evaluated. The arbitrary color code is adjusted as to emphasize changes in the vicinity of the predicted critical line of an ideal input driven network.

The results of Sec. 5.3.1 suggest that weights drifting towards larger values will especially affect liquids drawn for large k and small σ^2. This behavior is observed for parameter set 1 and the slight shift in the measured integral Hamming distance towards smaller nominal σ^2 is indicated by black arrows (plot C). The numbered parameter sets are identical to the ones used previously.

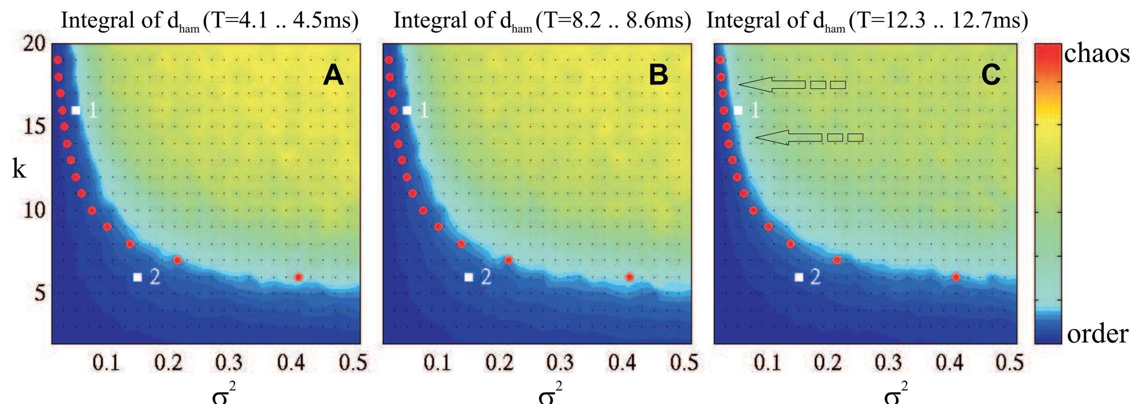


Figure 5.16: Three 'snapshots' of the time course of the mean integral Hamming distance measured for 30 liquids per parameter set. From left to right the same liquids are evaluated at different times after the weight refresh. The liquid dynamics are measured for the time windows which correspond to the measurements of the memory capacity shown above. The arbitrary color code is optimized as to emphasize the slight shift of chaoticity towards smaller nominal σ^2.

Fig. 5.17 shows a detailed examination of the decay of the mean Hamming distance for 100 liquids drawn for $k = 16$, $\sigma^2 = 0.05$ (parameter set 1) and $k = 6$, $\sigma^2 = 0.15$ (parameter set 2). The top plot shows the decay recorded at parameter set 1 in the respective time windows of Fig. 5.16; the colors of the curves correspond to the time window, i.e., 4.1-4.5 ms (blue), 8.2-8.6 ms (green), and 12.3-12.7 ms (red). The errorbars show the deviation of the mean; for clarity only every tenth errorbar is plotted. Since the observed decay is not governed by an absolute time but rather by the number of discrete time steps, the Hamming distance is plotted versus the time steps.

The observations of Fig. 5.14 already showed that the degradation for large k, small σ^2 liquids (parameter set 1) is essentially the same as for small k, large σ^2 liquids which leads to the conclusion that the drift towards chaos is not the dominating effect for the degradation. Nevertheless, a drift towards chaos is clearly observed especially for large k, small σ^2 liquids as can be seen from the upper plot in Fig. 5.17. There it can be seen that with increasing time the mean asymptotic value increases. This is to say that the dynamics become more chaotic over time. This effect

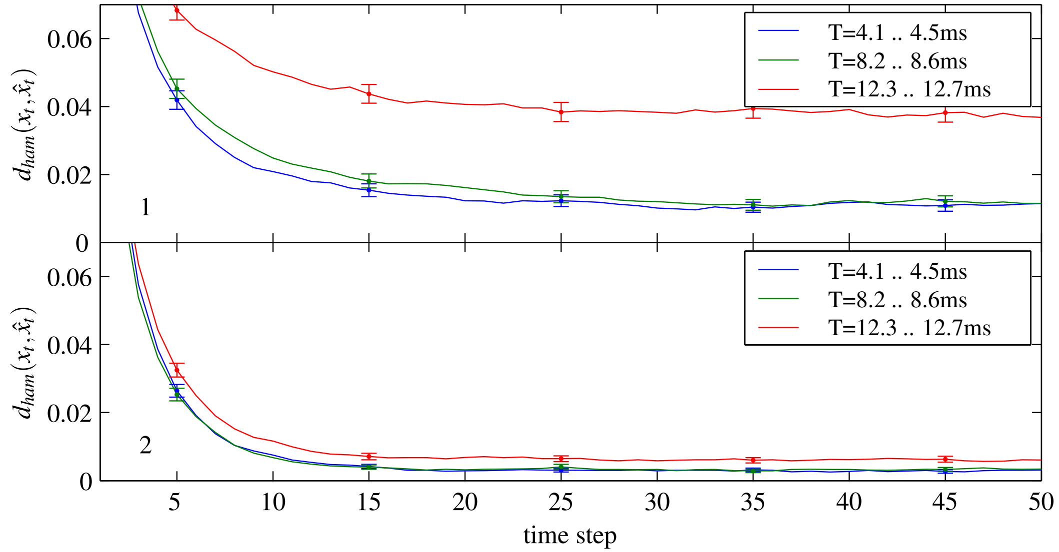


Figure 5.17: For the two parameter sets indicated with numbers 1 and 2 in Fig. 5.16 a detailed time course is plotted showing the mean Hamming distance of liquid states driven by pairs of initially different end eventually identical input streams, c.f. Sec. 2.3.3. The colors of the curves indicate the time windows for which the Hamming distances are measured. It is observed that the asymptotic values increase with time, i.e., the dynamics drift towards chaos. In agreement with measurements of Sec. 5.3.1 this effect is stronger for liquids drawn for smaller σ^2 (top plot).

is much smaller for liquids with weight distributions of small σ^2 (bottom plot) as expected from earlier measurements in Sec. 5.3.1.

Trainability on Degraded Liquids In order to see how much the drift towards more chaotic dynamics influences the trainability, liquids are generated with several k, σ^2 combinations and the training of the readout is performed on inputs some time after the weights have been refreshed. Precisely, the training is performed on a 0.41 ms time window (1000 time steps) starting at 0 ms, 4.1 ms, 8.2 ms and so on. Fig. 5.18 shows the generalization performances for six time windows beginning with the time window that starts at 4.1 ms (increasing starting time from plot A to F). The mean memory capacity is computed for respective time windows on a sufficiently long generalization input, i.e. for 1000 time steps starting at the appropriate times. At each parameter set k, σ^2, 15 liquids have been generated which are used to train the readouts for all time windows. For reference, the prediction for critical dynamics in the ideal (non-degraded) case according to the mean-field theory is indicated by large black dots in all six plots.

From the plot C (T = 12.3-12.7 ms) it can be seen that it is quite possible to successfully train readouts on degraded liquid dynamics that cause an LSM trained on freshly updated weights (c.f. Fig. 5.13) to fail almost completely. This is in agreement with the observations made on the liquid dynamics directly since it was already concluded that the reason for the performance decrease in Fig. 5.13 is due to changes in the liquid dynamics that cause the readout to misinterpret the liquid states. As expected by the shift of the onset of chaos for large k, small σ^2 liquids in Fig. 5.16, the readout performance is shifted towards smaller nominal σ^2.

Yet, the overall performance of readouts trained on even more degraded liquids (D-F) shows a visible decrease (besides the mentioned parameter sets with small σ^2 that profit from the drift towards chaotic dynamics). This is shown in detail for the four indicated parameter sets (numbers

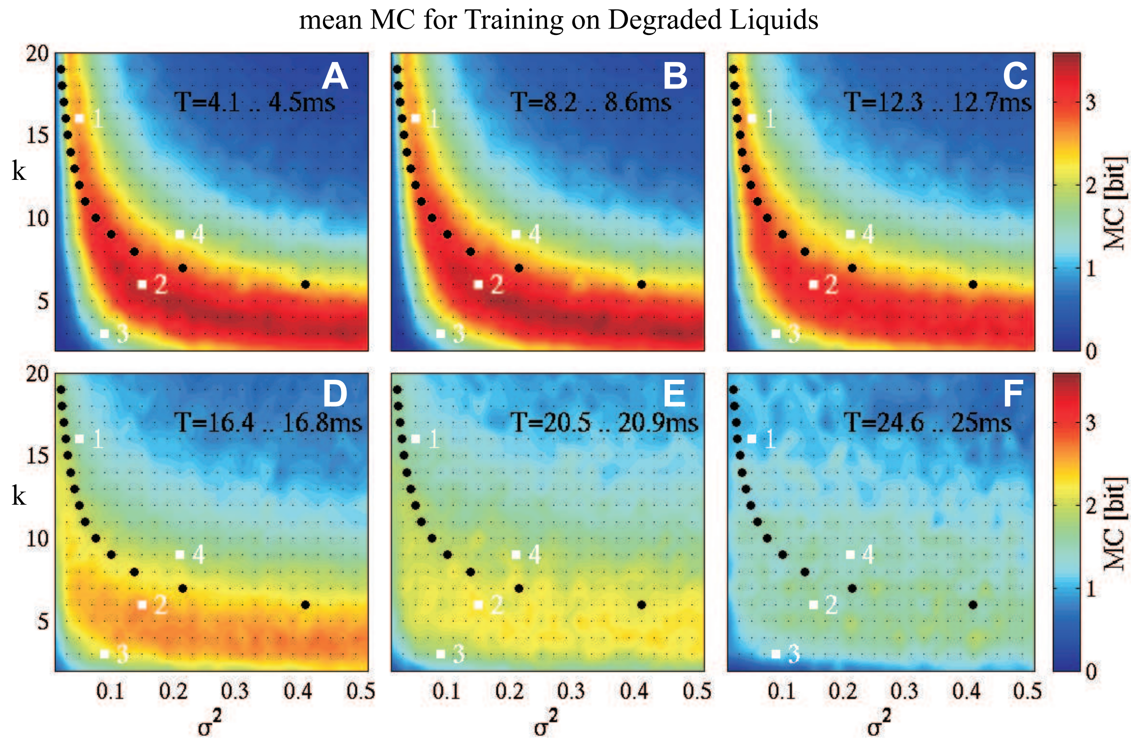


Figure 5.18: Visualization of the generalization performance for LSMs trained on increasingly degraded weights. In order to derive the plots, a single k, σ^2 parameter sweep is performed with 15 liquids generated per parameter set and exposed to a sufficiently long input stream. The liquid responses from the indicated time windows are taken to train readouts on the 3-bit time-delayed parity task. Shown is the generalization performance during the exact same time windows. It can be seen that the peak performance initially shifts towards smaller nominal σ^2 and slightly decreases (A-C). For later times the performance decrease is stronger, yet a medium performance region extends towards larger k (D-F).

1 to 4) in Fig. 5.19. There, the mean memory capacity is plotted for the six time windows shown in Fig. 5.18 and additionally for the windows at 0-0.4 ms and 29.1-29.5 ms. The connecting lines are a visualization and represent neither measurements nor a fit. The errorbars are the deviation of the mean.

The LSMs trained on previously ordered dynamics (parameter set 3: $k = 3$, $\sigma^2 = 0.09$) show an intermediate increase in training performance (B to D). Conversely, the LSMs of parameter set 1 ($k = 16$, $\sigma^2 = 0.05$), which already in the non-degraded case peak at slightly more chaotic dynamics as discussed in Sec. 5.2.2, exhibit already a decrease between the time windows 8.2-8.6 ms and 12.3-12.7 ms shown in plots B and C. Both observations are expected for a shift in the onset of chaos towards smaller nominal σ^2. Unanimously, liquids for all parameter sets show a steepening decrease in readout performance for training windows more than 20 ms past the last weight refresh. Eventually, after about 30 ms no reasonable readout performance can be achieved.

Parameter set 4 ($k = 9$, $\sigma^2 = 0.21$) from the chaotic regime yields a constant readout performance for the longest period after a weight refresh (up to the window 20.5-20.9 ms, plot E). This is as well observed in the parameter sweeps of Fig. 5.18: Especially for the training time windows shown in the lower row, the region of intermediate readout performance (a MC of around 2 bit; color coded green) extends towards larger k independent of σ^2. An explanation for this—and as

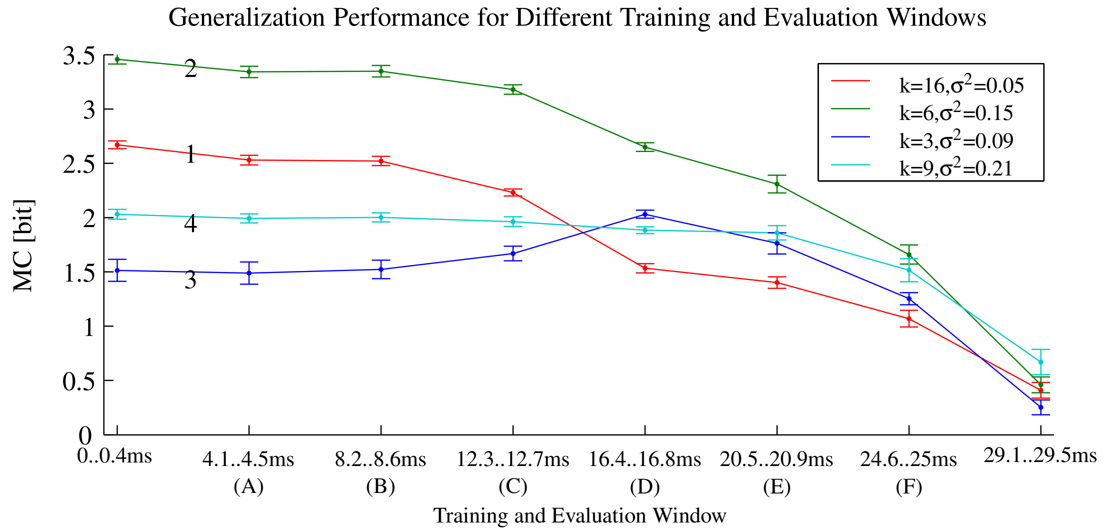


Figure 5.19: For the four parameter sets in Fig. 5.18 indicated by the numbers 1 to 4, the achieved readout performance is plotted for the 0.41 ms training/generalization windows starting at increasing times. The connecting lines are merely for identifying the corresponding values and are no interpolation; the errorbars indicate the standard deviation of the mean. For intermediate time windows (8.2-8.6 ms and 16.4-16.8 ms), the parameter sets with small σ^2 (set 1 and 3) show a behavior that is to be expected for a shift of the edge of chaos towards smaller nominal σ^2: The performance in the ordered regime (set 3) grows, the performance in the already slightly chaotic regime decreases (set 1). Independent on the parameters, all liquids unanimously show an accelerating decrease in training performance for times 20 ms or more after the last weight refresh.

well the final poor training performance observed for all parameter sets—may be that the leakage of the weights indeed is not uniform for all synapses due to variations in manufacturing. In this case, the degradation of the weights will be small for some and larger for others and thus may affect the dynamics differently depending on the original generation parameters. Eventually, the actual weights may become very different from the weight distributions originally used for input-driven networks causing the kernel and memory capabilities of the liquid to collapse.

5.3.3 Graceful Degradation with Individual Synapse Faults

In the previous section, it was mentioned that the degradation of the weights is likely to be non-uniform. There may be synapses that show an increased leakage compared to the mean leakage; this can cause individual synapses to rapidly assume their maximum amplitude (the sign may be positive or negative). The reasons for this arise from manufacturing, for example the thickness of the gate oxide may vary which affects the isolation performance of the transistor S4 connecting the weight capacitor with I_{park} (c.f. Sec. 3.3.2).

To examine this type of fault closer, it is possible to artificially set individual non-zero synapses to their maximum value while maintaining their sign. The following experiment is used to examine the influence of this immediate synapse fault: A liquid is drawn from a given parameter set k, σ^2 and a readout is trained for the 3-bit time parity problem ($\tau = 0$) on a 4000 time step training input. The mutual information is tested on a 2000 time step generalization input. Then, individual non-zero synapses of the input driven network are successively set to their maximum amplitude and the whole weight array is refreshed. After each introduced fault, the mutual information achieved on the generalization input is measured.

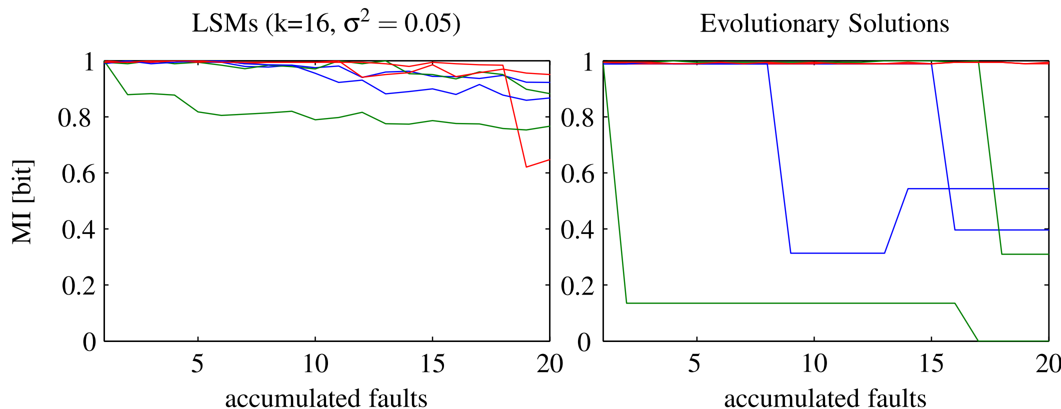


Figure 5.20: Left: The figure shows the degradation of the readout performance versus the accumulated synapse faults on the 3-bit time parity task. Three LSMs with liquids drawn for $k = 16$, $\sigma^2 = 0.05$ are tested twice for two series of random faults; the results obtained with the same liquid have the same color. While the plot gives no quantitative statement, it illustrates that LSMs primarily degrade gracefully, yet may observe sudden drops occasionally. **Right:** Three feed-forward networks with a tapped delay line memory are trained by an evolutionary algorithm on the 3-bit time parity task as well. Plotted is the achieved generalization performance versus the accumulated synapse faults for two series of random faults each (same color indicates same network). The figure is meant to illustrate that these networks tend to remain unaffected until a fault causes a sudden drop in performance.

The left part of Fig. 5.20 shows the mutual information versus the accumulated faults for three different liquids drawn for $k = 16$, $\sigma^2 = 0.05$. Each liquid is tested for two different series of faults. Similar colors indicate the same liquid. The right hand side of Fig. 5.20 shows a similar experiment with neural networks trained by an evolutionary algorithm on the same task: Three solutions obtained by repetitive training runs are tested for two series of faults. To simplify the training, a delay line for 2 time steps is implemented by appropriate weights and cycles in order to reduce the training task to a spatial parity problem easily solvable by a feed-forward architecture. For each of the 3 time steps (the current and the two delayed), as well the inverse is presented to the network, i.e., 6 input neurons are used. The remaining topology is comprised of 6 intermediate neurons fully connected to all inputs and to the single output neuron. This totals to 42 synapses to be trained by the evolutionary algorithm. Details of the algorithm used are given in Sec. 4.4.1; the parameters use were $\phi = 30$, $\chi = 0.5$, $\mu_u = 0.2$, $\mu_n = 0.1$, $e = 2$.

Both types of networks exhibit a decrease in performance while accumulating faults, yet there are qualitative differences: The LSMs primarily show a gradual decrease (even though sudden drops may occur as indicated by the one liquid dropping at the 19th accumulated fault). The networks trained by the evolutionary algorithm, on the other hand, primarily tend to remain unaffected until a sudden drop in performance occurs. Fig. E.4 in the appendix shows the same plot for 100 networks each and supports that the behavior in Fig. 5.20 indeed is representative.

In order to quantify this observation, a series of experiments is performed that tests 100 networks for 200 different series of accumulating faults each. This is done for four different types of networks: The first two are LSMs with liquids drawn from two different parameter sets ($k = 16$,

PSfrag replacements

$\sigma^2 = 0.05$ and $k = 6$, $\sigma^2 = 0.15$). The remaining two are the evolutionary solutions as well as triple-redundant[4] evolutionary solutions.

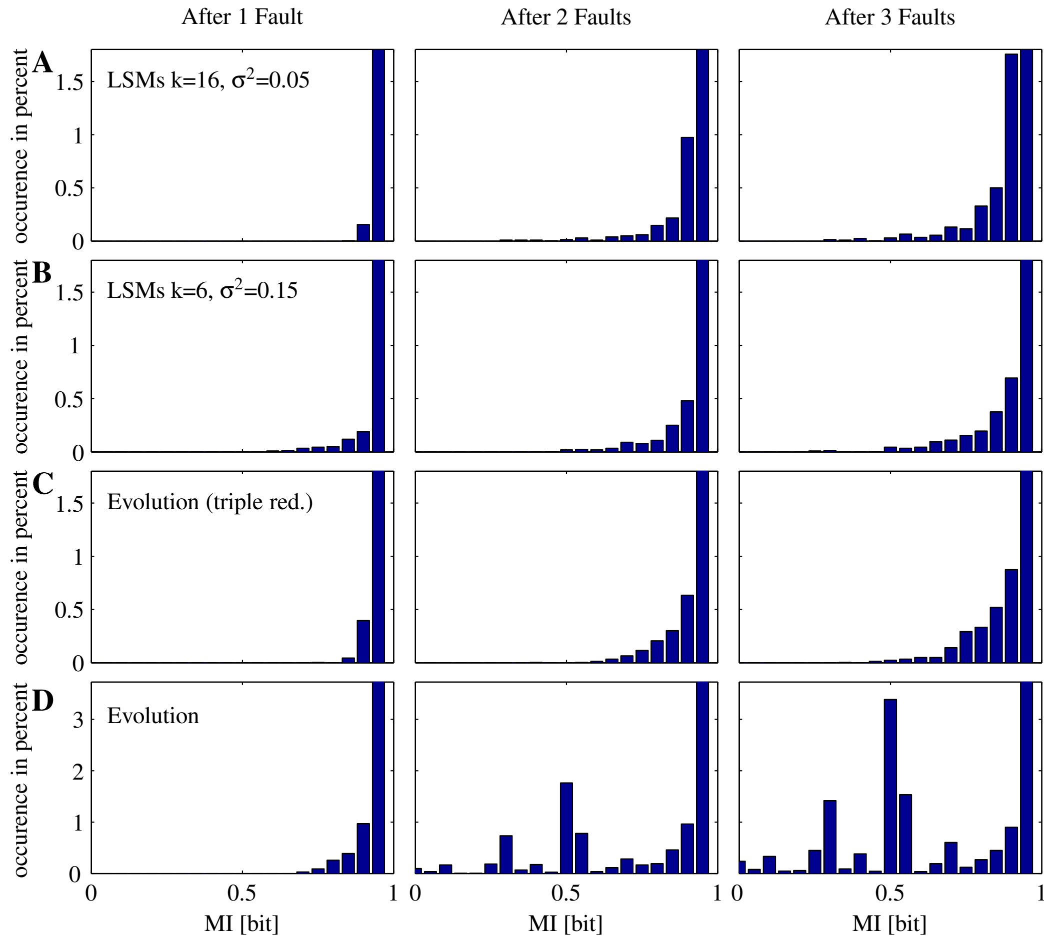


Figure 5.21: For two types of liquid (rows A and B), a triple-redundant evolutionary solution (row C), and a simple evolutionary solution (row D) the occurrence probabilities are plotted for exhibiting a certain drop in mutual information due to 1 (left column), 2 (middle column), or 3 (right column) accumulated faults. For each type of solution, 100 different networks are tested for 200 different fault series each. The two LSM solutions exhibit a similar behavior to the triple-redundant evolutionary solutions, i.e., a decreasing probability for the encounter of smaller MI values. The non-redundant evolutionary solutions show an overall higher (y-axis!) and non-decreasing probability for small MI values for more than one fault.

In total, for each of the four cases, 20000 performance series (MI versus accumulated faults) are measured. For 1, 2, and 3 accumulated faults the measured mutual information is binned to three histograms which are the columns in Fig. 5.21. The rows correspond the the two liquid implementations (rows A and B), the triple-redundant (row C), and the non-redundant networks (row D) trained by the evolutionary algorithm.

What can immediately be seen is the similarity in shape of the two liquid solutions (rows A and B) and the triple-redundant evolutionary solution (row C). In the majority of the cases the network

[4] In order to get a triple-redundant evolutionary solution, three individual evolutionary solutions are trained and a majority vote is implemented in software.

performance MI is not decreased below 0.95 bit (right most bin). And with decreasing probability the mutual information falls below values of 0.9 bit, 0.85 bit etc. It is interesting to note that this is also observed for the triple-redundant solution. At least for the 1-fault case one may expect perfect performance (MI = 1 bit). Yet, the observation can be explained: The individual solutions, between which the majority vote is performed, are determined by a heuristic that may rely on very subtle weight changes. Due to timely variations in the operating conditions, therefore, a found solution may not work properly in all cases. Although the criterion ending the training process evaluates the performance repetitively, here the criterion obviously is not strict enough to ensure robust individual solutions. A majority vote of three non-perfect individual solutions therefore may encounter, already with no or one introduced fault, the observed behavior.

The single evolutionary solutions (row D), on the other hand, show a quite different performance, especially if affected by two (middle plot) or three faults (far right): There is a non-decreasing probability of collapsing completely (MI = 0 bit). The overall higher failure probability should also be noted (change of scaling).

From the comparison of the first and second row in Fig. 5.21 it may further be observed that LSMs with liquids drawn from the two different parameter sets differ slightly: In the case of two or more accumulated faults (the two right most plots of rows A and B) the occurrences of smaller MI is quite similar, but LSMs with small σ^2 liquid tend to be slightly more degraded (higher probability for the 0.9 to 0.95 bit bin in row A). This becomes more obvious when the measured data is analyzed differently: In Fig. E.5 of the appendix, the measured performance series are inspected for whether or not the mutual information falls below a certain threshold and the number of faults causing this degradation is recorded. For the measured 20 faults per series, the number of LSMs falling below an MI of 0.9 bit is higher for LSMs with smaller σ^2. On the other hand, both figures show that the LSMs with smaller σ^2 are more tolerant to single faults.

These results are not meant to assess in general the quality of solutions obtained by evolutionary algorithms, rather to set the observed degradation performance of the LSMs in context. This is important since no effort has been dedicated to optimize the evolutionary reference solution in terms of topology or fault-tolerance. Yet, in the triple-redundant case a minimum of fault-tolerance is to be expected. Therefore, the single evolutionary and the triple-redundant solution support the fact that the LSM solutions exhibit a performance that is more similar to a redundant solution even though no special precautions have been taken. In conjunction with the qualitative observations partly shown in Fig. 5.20, this supports the initial conjecture that the readout is not relying on dynamics resulting from individual neurons and synapses but rather on transient features resulting from several connections. If these connections fail, a graceful degradation is likely to result. The implications of the observed fault tolerance will be analyzed in the closing discussion.

Discussion

This thesis described the construction and operation of a neural network experiment, where the term *experiment* is chosen to illustrate the idea of a system that is comprised of hardware and software and allows a physical test of its underlying propositions and paradigms. In contrast to a pure software system, the interconnection of various levels of hardware in conjunction with the furthermore necessary interaction with high-level software increase the design, construction, and operation effort. A substantial amount of the work invested during this thesis was dedicated to derive a stable experimental framework and to provide several operational instances of the neural network experiment.

However, there are good reasons to undergo the endeavor of bringing dedicated mixed-signal circuits, programmable logic, and software together: It makes it possible to do research on the computational substrate itself. This is necessary, since the future of VLSI faces architectural problems due to the progressing miniaturization. The approach of merely shrinking existing digital circuits to smaller feature size technologies leads to an diverging power dissipation. Furthermore, the already inherent static variations of CMOS devices are expected to worsen and eventually lead to devices that are unreliable in the course of their operation which may severely affect the yield. A shift in perspective is thus inevitable: Future designs will not be constrained by a limitation of available resources but rather by power and reliability constraints.

The HAGEN prototype ASIC described in Ch. 3 is a mixed-signal realization of a perceptron paradigm that allows an efficient CMOS implementation due to its regularity and the simplifications arising from the chosen McCulloch-Pitts neurons. Representing the synaptic weights by currents allows to ensure a power consumption that is essentially proportional to the weight amplitudes. Furthermore, employing a current-steering approach for the evaluation of the postsynaptic activity keeps the power consumption signal-independent. The combination of both aspects allows the implementation of low-power, yet small and accurate synapses and to have the power consumption fully controlled by the programming of the synaptic weights. In this paradigm, unused synapses therefore merely are small areas of currently unused silicon but neutral in terms of power consumption. In combination with the small connectivity requirement of the individual synapses—they can be connected to the input and output neuron via their neighboring synapses—this makes the realized network paradigm a potential candidate for future scaled down feature sizes.

In addition, to be a candidate architecture for scaling with the number of available resources, the network paradigm realized by the HAGEN prototype is based on confining the analog circuits of the synapses and neurons to well-defined *network blocks* with a digital interface. This allows tailored analog circuits in size, speed, and power consumption while maintaining a straight forward way to form large networks: Due to the discrete time operation and the digital interface, a consistent network can be formed simply by replicating network blocks and providing digital routing between them. These concepts were described in Ch. 1.

The framework presented in Ch. 4 is dedicated to providing the necessary means for evaluating these propositions in appropriate experiments. The nature of these experiments is diverse and ranges from characterizing immediate substrate properties, e.g., the measurement of the neuron offsets, to using the neural network for an actual application. Key to the experimental framework therefore is the abstraction on the functional level which makes it easy to exchange training strategies and even to migrate functionality from software to the programmable logic. In the thesis presented here, the modularity on the substrate level of the framework has been emphasized: The separation into dedicated mixed-signal VLSI circuits, reconfigurable logic in an FPGA, and a embedded microprocessor, which are interchangeable, makes it possible to easily advance the experiment. On the one hand, this has been illustrated for two different ANN ASICs; on the other hand, the advanced framework implementation employing an FPGA with an integrated PowerPC core forms the basis for the construction of experiments that operate several ANN ASICs as distributed resources of the same logical network.

With the help of the experimental framework, the HAGEN prototype has been extensively explored. Measurements on the offsets inherent to the synapses and neurons were performed [86] which essentially show that the synapse array does not need any calibration and that the expected DAC and neuron offsets can be inferred and compensated as expected. Those results show that the current-steering approach allows a static accuracy of 9 to 10 bit of the nominal 11 bit weights for the individual synapse. Results on the variable network resources (presented in Sec. D of the appendix) furthermore show that the relative homogeneity of the synapses is such to interface multi-bit inputs with up to 7 bit accuracy without any individual synapse offset compensation. In [86], those multi-bit inputs are used in conjunction with evolutionary algorithms to successfully demonstrate that the network paradigm of the HAGEN prototype yields competitive and even better results in common pattern classification benchmark problems. The iterative evolutionary training, which was tailored to make use of the program and recall speed of the hardware as well as the parallelism, represents a promising method for use in real-world applications.

In this thesis the neural network experiment has been used to explore a recently published strategy which operates a neural network recurrently in order to do computations on time series. The *liquid computing* or *echo state network* approach promises to have some favorable properties for a hardware adaptation which for the first time have been verified in a physical experiment: Due to the primarily random weights used to configure the recurrent neural network, it was shown to be tolerant to offset variations inherent to the HAGEN prototype. Furthermore, the graceful degradation with respect to faults occurring during operation exhibited by this *computing without stable states* leads to the conclusion that it is not relying on individual connections in the network.

The extensive parameter scans for exploring the dynamics of the resulting recurrent networks demonstrate that it is possible to tailor a liquid state machine to the hardware, to the number of expected faults, and the desired performance. Critical dynamics in the *input driven network* liquids [21] either require many inputs per neuron and weights drawn from a narrow weight distribution or fewer connections drawn from a broader weight distribution. Since both configurations result in a low mean average activity, the HAGEN prototype is well suited to realize those liquids. If one chooses the variance of the weight distribution large enough, the offset variations of the neurons become irrelevant. Due to their possibly large weights, those sparse liquids furthermore exhibit a robustness if several of their weights are set to maximum amplitude. Those investigations simulate a rapid weight leakage of individual synapses. Liquids with many connections yet smaller weights show a good robustness to individual faults.

Yet, it was also observed that the drift of the liquid dynamics caused by the weight leakage of the HAGEN substrate affects the overall readout performance independently of the connectivity of the liquid. It could be shown that by re-training the readout the leakage effect can be compensated

until eventually the dynamics either are chaotic or changed in a way that is less favorable in terms of temporal memory or separation property. An interesting way to overcome this has been published in [151]: The criticality condition of the input driven network liquids can be used to derive a local rule that allows each neuron to scale its weight as to adjust the liquid towards critical dynamics. The required continuously re-training of the readout can be realized by a perceptron with online-learning [111].

While the theoretical tractability of input driven networks is very useful for predicting criticality, the observed scaling behavior, which only shows a logarithmic increase in memory capacity with the number of neurons, limits their applicability. The key to overcome this is part of the liquid computing approach: By introducing different time constants to the recurrent connections of the liquid, time can be used to increase the dimensionality of the liquid filter. This is shown by the simulation experiments of Maass et al. [131] where 135 leaky I&F neurons suffice to solve a quite challenging speech recognition task. The liquid state available to the readout was the accumulated spike responses recorded by 135 leaky integrators. The HASTE extensions to the neural network experiment target the augmentation of the HAGEN perceptron model in this direction and may eventually yield liquids with a higher memory capacity on the HAGEN substrate.

Unlike Maass et al. [132], it could not be observed that the readout is *profiting* from variations of the substrate. The reason for this lies in the nature of the variations they introduce: They randomly draw the time constants of the synaptic connections according to distributions found in biological data. This diversity increases the ‘richness’ of the dynamics in the time domain whereas the offset variations of the HAGEN prototype leave the discrete time constants untouched that arise by one or more network cycles. Therefore, introducing connections with variable delay to the HAGEN architecture may be a possibility to increase the readout performance. As described in Ch. 1, the underlying network model is well suited to accommodate those connections. Once the distributed implementation of the neural network experiment allows to route neuron data between several HAGEN ASICs at the adequate rate, this may be used to examine whether better liquids can be obtained.

Eventually, input driven network liquids implemented on the HAGEN architecture may be used to realize non-linear channel equalization in mobile telecommunication. This application has for example been proposed by Jaeger [107] to be a suitable application for this type of recurrent neural networks. The HAGEN prototype not only showed its suitability to implement liquids but also is dedicated to process incoming data streams of 12 bit width with up to 50 MHz and a power consumption suited for mobile environments. The neural network experiment therefore could be used to build a prototype for this kind of application.

For the exploration of liquid computing as an alternative computing paradigm, nevertheless more complex dynamical systems need to be explored. Besides the exploration of biological issues, a future ANN ASIC [185] implementing leaky I&F neurons and *spike-time dependent plasticity* (STDP) that can be integrated to the neural network experiment will allow the continuation of the investigations on liquid computing as an architectural alternative to classical computing paradigms. Adequate dynamics of those liquids of course require revisiting the question of which parameters sets reliably yield a good readout performance. Furthermore, it needs to be answered how to maintain suitable dynamics with changing input activity or degrading connections. Those questions may be closely related to mechanisms that ensure stable activity in biological networks of spiking neurons and will be fruitful area of future research.

Appendix A

Liquid Computing Supplements

A.1 Linear Regression Training

Without loss of generality, Eq. 2.4 can be assumed to be comprised of only one readout, which yields the standard form of an overdetermined matrix equation

$$\mathbf{Ax} \approx \mathbf{b}, \tag{A.1}$$

with $\mathbf{A}$ being a real $\widetilde{m} \times \widetilde{n}$-matrix having full rank and $\widetilde{m} \geq \widetilde{n}$, x being a $\widetilde{n} \times 1$-vector and b a $\widetilde{m} \times 1$ vector; $\approx$ means the least-squares approximation.

To solve Eq. A.1 one essentially has to compute the normal equations

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b} \tag{A.2}$$

and solve for $\mathbf{x}$ by computing the inverse $(\mathbf{A}^T\mathbf{A})^{-1}$. In order to avoid the computation of $\mathbf{A}^T\mathbf{A}$ and the inverse, the standard way to solving a set of overdetermined linear equations is to employ a QR-factorization which improves stability and numerical precision. In this method, $\mathbf{A}$ is decomposed to an orthogonal $\widetilde{m} \times \widetilde{m}$-matrix $\mathbf{Q}$ (with $\mathbf{Q}^T\mathbf{Q} = \mathbb{1}$) and an upper triangular $\widetilde{n} \times \widetilde{n}$-matrix $\mathbf{R}$:

$$\mathbf{A} = \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}. \tag{A.3}$$

Eq. A.2 then reduces to

$$\begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix} \mathbf{x} = \mathbf{Q}^T\mathbf{b}. \tag{A.4}$$

It can be seen that only the matrix-vector product with the first n rows of $\mathbf{Q}$ has to be computed and the equation is easily solved for $\mathbf{x}$ by back-substitution.

The actual QR-factorization is realized by the successive application of the Householder transformation [29, 167]:

$$\mathbf{H} = \mathbb{1} - \frac{2\mathbf{uu}^T}{|\mathbf{u}|^2}, \tag{A.5}$$

where $|\cdot|$ is the Euclidean norm and $\mathbf{uu}^T$ the outer product. If this symmetric and orthogonal matrix is applied to a column vector $\mathbf{v}$ and $\mathbf{u}$ is chosen according to $\mathbf{u} = \mathbf{v} - |\mathbf{v}|\mathbf{e}_1$ where $\mathbf{e}_1$ is the canonical basis vector, the following holds:

$$\mathbf{Hv} = \mathbf{v} - \frac{2\mathbf{u}(\mathbf{v} - |\mathbf{v}|\mathbf{e}_1)\mathbf{v}}{2|\mathbf{v}|^2 - |\mathbf{v}|\mathbf{ve}_1 - |\mathbf{v}|\mathbf{e}_1\mathbf{v}} = \mathbf{v} - \mathbf{u} = |\mathbf{v}|\mathbf{e}_1. \tag{A.6}$$

For the factorization the first Householder transformation, $\mathbf{H}_1$, is chosen such that it zeros the first column below the first element in the matrix A. The second, $\mathbf{H}_2$, zeros the second column below the second element, respectively, and so on. Then an upper triagonal matrix is obtained by:

$$\mathbf{R} = \mathbf{H}_{n-1} \cdots \mathbf{H}_1 \mathbf{A}. \tag{A.7}$$

And due to the orthogonality of the individual $\mathbf{H}_i$, the orthogonal matrix $\mathbf{Q}$ is given as:

$$\mathbf{Q} = (\mathbf{H}_{n-1} \cdots \mathbf{H}_1)^{-1} = \mathbf{H}_1 \cdots \mathbf{H}_{n-1}. \tag{A.8}$$

In case of the linear readout, the matrix $\mathbf{A}$ is comprised of liquid states sampled at different times t. Accordingly, the matrix initially is comprised of zeros and ones which is not favorable for the numerical stability of the factorization algorithm since it even might be singular. For these reasons, the column vectors of $\mathbf{A}$ are rescaled as to have a zero mean and Gaussian noise is added to each rescaled weight [155].

A.2 Mean-Field Theory for Input Driven Networks

Natschläger and Bertschinger [151, 150] derive a mean-field theory for input driven networks of $\{0,1\}$ threshold neurons by considering the limit of the number of neurons $N \to \infty$. This allows to treat the network activity, defined as the normalized sum of components i of a liquid state $x(t)$ at some time t that are one,

$$a(t) := \frac{1}{N} \sum_{i=1}^{N} x(t)_i, \tag{A.9}$$

as the probability that a given component i of the liquid state vector assumes the state one, i.e., $a(t) = \Pr\{x(t)_i = 1\}$. This is fundamental to the mean-field theory since now the activity of the next time step may be derived

$$a(t+1) = \Pr\{x(t+1)_i = 1\} = \Pr\left\{ \sum_{j:\omega_{ij} \neq 0} \omega_{ij} \cdot x(t)_i + u(t) \geq 0 \right\}, \tag{A.10}$$

which is only dependent on the previous activity and the input at time t.

Since the weights are drawn from a Gaussian distribution with variance σ^2, one can break down the sum $\sum_{j:\omega_{ij}\neq 0} \omega_{ij} \cdot x(t)_i$ to k Gaussian distributions of variance $n\sigma^2$ for k being the number of incoming connections per neuron and n the number of active connections thereof, i.e. $0 < n \leq k$. The probability of a component being one in the next time step if n of its inputs are active thus is given by:

$$\Pr\{x(t+1)_i = 1 | n\} = 1 - \Phi(-u(t), n\sigma^2), \tag{A.11}$$

where $\Phi(x, \sigma^2) = \frac{1}{2}(1 + \operatorname{erf}(\frac{x}{\sqrt{2}\sigma}))$ is the Gaussian cumulative density function. The probability of n inputs out of k being active can be given according to the binomial distribution and thus the update rule for the network activity is:

$$a(t+1) = (1 - a(t))^k \Theta(u(t)) + \sum_{n=1}^{k} \binom{k}{n} \cdot a(t)^n (1 - a(t))^{k-n} (1 - \Phi(-u(t), n\sigma^2)). \tag{A.12}$$

The case $n = 0$ describes whether the input is larger zero or not and is appropriately represented by the Heaviside step function, Θ.

This activity update rule is used by Natschläger and Bertschinger to predict the transient behavior of liquid states and ultimately to derive an update rule for the Hamming distance of the form: $d_{\text{ham}}(t+1) := H(d_{\text{ham}}(t), a(t), \hat{a}(t), u(t))$, where $a(t), \hat{a}(t)$ are the activities of two different liquid states. In order to do this, they look at the following scenario of two initially different liquid states $x(t)$, $\hat{x}(t)$ that are driven by the same input:

$$\begin{array}{ll}\text{trajectory 1:} & x(0) \xrightarrow{u(0)} x(1) \xrightarrow{u(1)} \cdots x(t) \xrightarrow{u(t)} x(t+1) \cdots \\ \text{trajectory 2:} & \hat{x}(0) \xrightarrow{u(0)} \hat{x}(1) \xrightarrow{u(1)} \cdots \hat{x}(t) \xrightarrow{u(t)} \hat{x}(t+1) \cdots\end{array}$$

What essentially has to be derived are the probabilities for the components i of the two liquid states $x(t+1)$, $\hat{x}(t+1)$ being different given the liquid states and the input at time t:

$$\Pr\{x(t+1)_i \neq \hat{x}(t+1)_i \mid x(t), \hat{x}(t), u(t)\}. \tag{A.13}$$

These probabilities can be derived by computing the probabilities for $x(t+1)_i < 0 \wedge \hat{x}(t+1)_i \geq 0$ and vice versa using Eq. A.11 and considering carefully the combinatorial cases that may arise, which eventually yields $H(d_{\text{ham}}(t), a(t), \hat{a}(t), u(t))$. This is elaborated in [150].

In order to analyze the dynamics of the network, they consider the gradient of the Hamming distance of two consecutive states:

$$\alpha := \frac{\partial d_{\text{ham}}(t+1)}{\partial d_{\text{ham}}(t)}. \tag{A.14}$$

If $|\alpha| > 1$, the Hamming distance grows; for $|\alpha| < 1$ the distance becomes smaller. Relevant is the gradient at $d_{\text{ham}}(t) = 0$: If there holds $|\alpha| < 1$, the zero Hamming distance is the only stable fixed-point, thus all state differences decay to zero and the network is in the ordered phase. If $|\alpha| > 1$, the network dynamics are said to be chaotic. Consequently, the transition between order and chaos, the critical dynamics, is given for $|\alpha| = 1$:

$$\alpha_{\text{critical}} = \left.\frac{\partial d_{\text{ham}}(t+1)}{\partial d_{\text{ham}}(t)}\right|_{d_{\text{ham}}(t)=0} = 1. \tag{A.15}$$

Natschläger and Bertschinger show that the evaluation at $d_{\text{ham}}(t) = 0$ reduces the expression for the probability introduced in Eq. A.13 to:

$$\begin{aligned} Q(n,u) &:= \Pr\{x(t+1)_i \neq \hat{x}(t+1)_i \mid x(t), \hat{x}(t), u(t)\} \qquad \text{(A.16)} \\ &= \int_{-\infty}^{-u(t)} \phi(\xi, n\sigma^2)(1 - \Phi(-u(t) - \xi, \sigma^2))d\xi + \int_{-u(t)}^{\infty} \phi(\xi, n\sigma^2)\Phi(-u(t) - \xi, \sigma^2)d\xi, \end{aligned}$$

where n is the number of active components in the previous state and $\phi(x, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}}$ is the Gaussian density function and $\Phi(x, \sigma^2)$ the cumulative Gaussian density function as mentioned earlier.

Finally, considering the various cases for active components n of the incoming connections, the criticality criterion (Eq. A.15) given in a very simplified version in the main text (Eq. 2.9) reads:

$$\mathrm{P}_{\text{bf}} := \sum_{n=0}^{k-1} \binom{k-1}{n} a^n (1-a)^{k-1-n} (rQ(n, \overline{u} + 0.5) + (1-r)Q(n, \overline{u} - 0.5)) = \frac{1}{k}, \tag{A.17}$$

where r is the probability of the input being $\overline{u} + 0.5$. The dependency on the activity makes a numerical solution of Eq. A.17 necessary, i.e., for the activity a the steady state of the averaged network activity $a^\star$ is computed and used.

Appendix B

HAGEN Prototype Supplements

This page remains empty. The supplementary information to the HAGEN prototype starts on the next page.

B.1 Inter-Block Routing

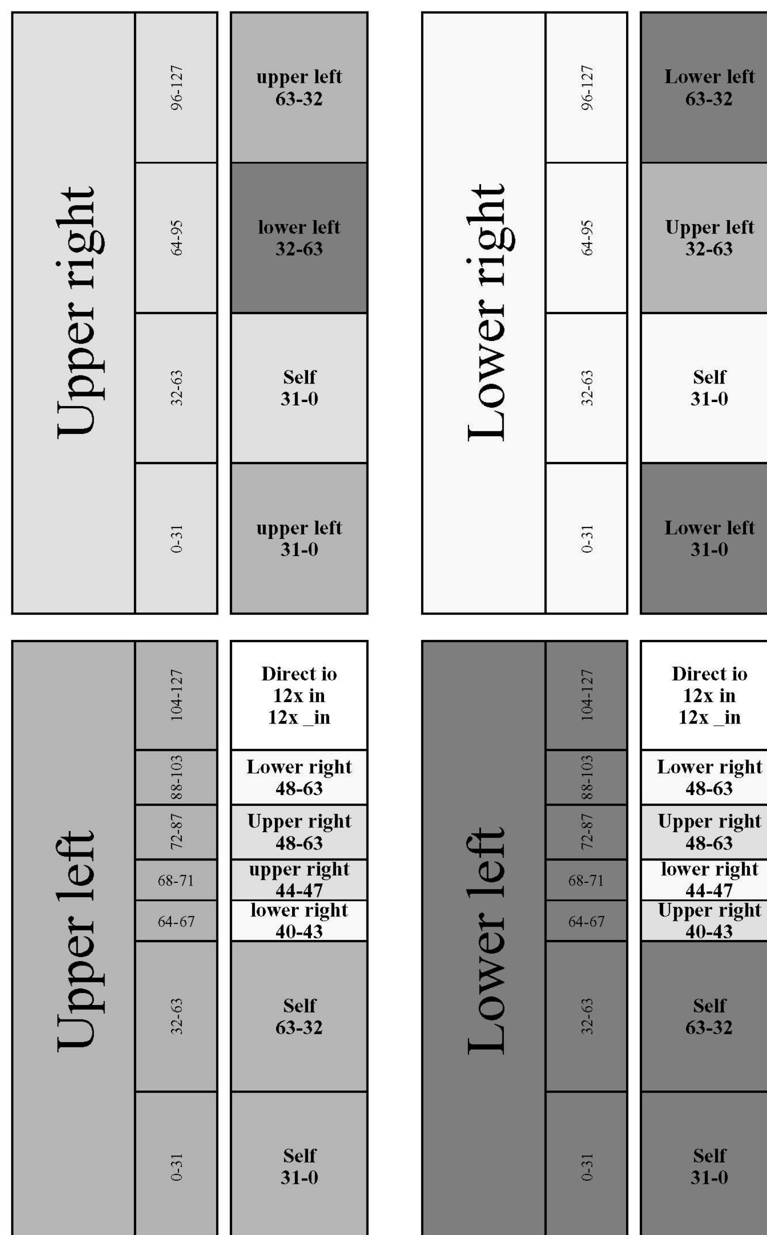


Figure B.1: Intra- and inter-block routing of the HAGEN prototype. Figure by T. Schmitz.

B.2 Bonding Diagram

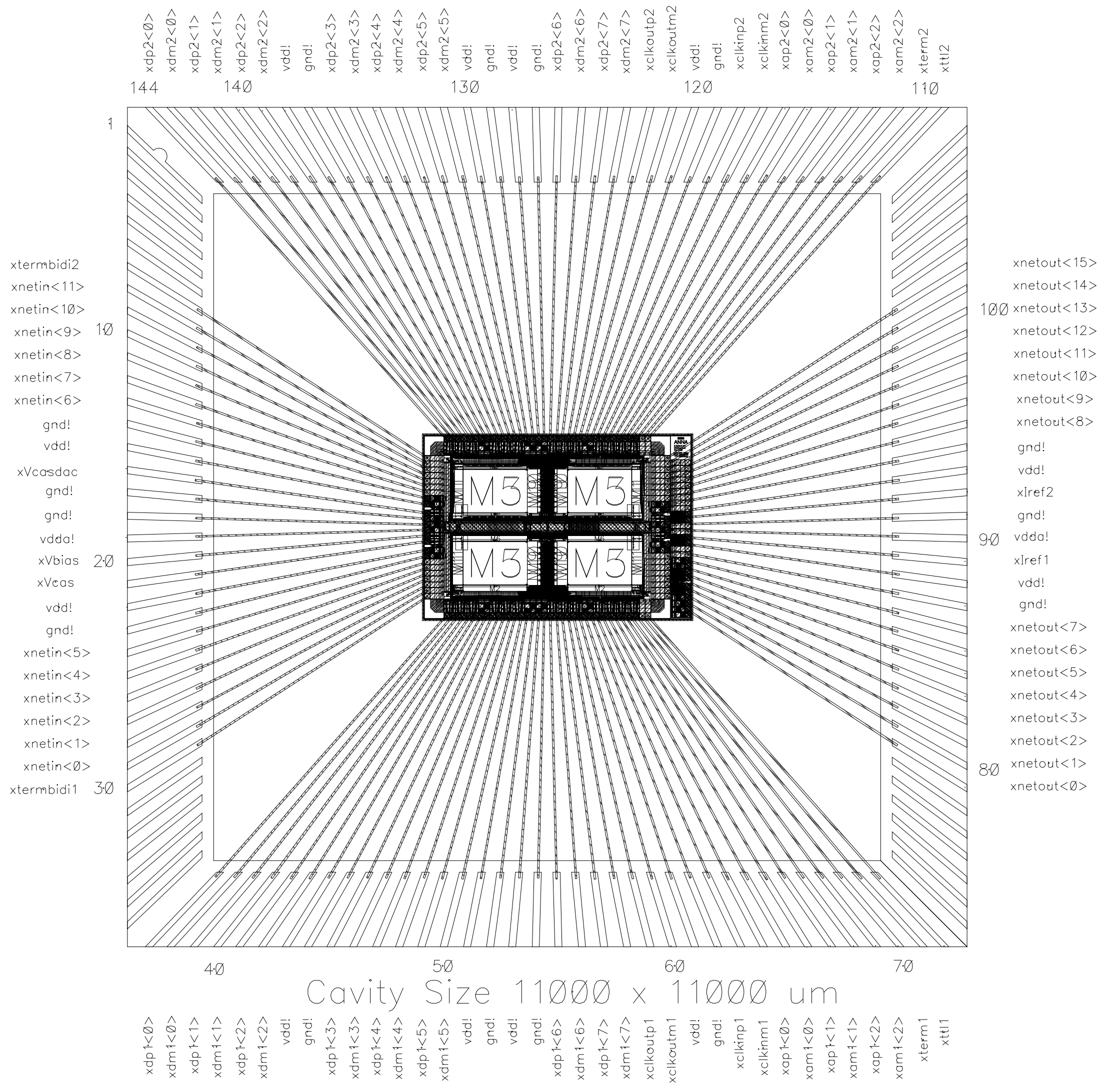


Figure B.2: Bonding diagram of the HAGEN prototype. All pins of the HAGEN prototype are bonded. Note: Attached to the right hand side of the HAGEN prototype there is a row of LVDS test structures that need not to be bonded for regular operation.

B.3 Interface

B.3.1 Command-Address Symbols

CA<5:0>	command	description
000xxx	idle	This is the well-defined idle state in which all other data is ignored.
010xxx	write sca	The 16 bit word of DATA<15:0> transferred in the same clock cycle is written to the register SCA.
011xxx	write scb	The 16 bit word of DATA<15:0> transferred in the same clock cycle is written to the register SCB.
100aaa	write neuron	The 3 bit address *aaa* determines a set of 16 input neurons that are programmed with the 16 bit data of DATA<15:0> transferred in the same clock cycle.
101aaa	write dac	The 3 bit address *aaa* determines which of the 8 DACs receives the 12 bit weight data in the lower 12 bits of DATA<15:0> transferred in the same clock cycle.
110aaa	read neuron	This causes the interface to activate its *clkout* and send the output data of 16 neurons determined by the address *aaa*.
111xxx	read scb	This causes the interface to activate its *clkout* and send the 16 bit contents of register SCB.

Table B.1: Description of the Command-Address symbols. The MSB is given first; ‘x’ denotes a *don’t care*, ‘a’ an address bit, and ‘0’/‘1’ the bit pattern on which the command is decoded.

B.3.2 Slow-Control Registers A and B

bit	name	description
0	neval_low	low word bit for generating neval
1	nread_low	low word bit for generating nread
2	nclear_low	low word bit for generating nclear
3	ceval_low	low word bit for generating ceval
4-7	saaddr<0:3>	address of the 16 weight storage units associated with a DAC
8	neval_high	high word bit for generating neval
9	nread_high	high word bit for generating nread
10	nclear_high	high word bit for generating nclear
11	ceval_high	high word bit for generating ceval
12	banksel	activates the alternate DAC input register
13	rwen	enables the right synapse array for writing
14	lwen	enables the left synapse array for writing
15	dacclr	clears all DACs

Table B.2: Description of the 16 SCA bits of the half chip interface.

bit	name	description
0-6	caddr<0:6>	column address used during programming
7	-	not used
8	fbdata	used with column enable to determine feedback input
9	lcen	left column enable used during programming
10	rcen	right column enable used during programming
11	lwrzoff	left zero offset activated for writing
12	rwrzoff	right zero offset activated for writing
13	lboost	activation of boost option in left weight storage units
14	rboost	activation of boost option in right weight storage units
15	lien	highest bit for input neuron addressing; on high left side

Table B.3: Description of the 16 SCB bits of the half chip interface.

B.4 Clock Pattern

B.4.1 Preamble-Pattern

The HAGEN prototype does not provide an explicit neuron reset that can set the neurons to a predefined output. Rather, with the help of a specific control sequence—a so-called *preamble*—the neurons can be set to a state given by their internal offset. This can be achieved by keeping *ceval low* while otherwise using the clock pattern of Fig. 3.9 in Sec. 3.3.1 of the main text. This causes the neurons to evaluate their own offset. The respective pattern is shown in Tab. B.4. With each edge of the interface clock a transition occurs, i.e., four clock cycles are necessary to perform the pattern. An alternative strategy is to dedicate a synapse column set to maximum negative weight and have it evaluated as the first cycle of each network evaluation.

ceval	*nclear*	*nread*	*neval*
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	1
0	0	0	1
0	0	0	1
0	0	1	1
0	0	0	1

Table B.4: Preamble-pattern: Time sequence of the signals resetting the output neurons to the state given by their internal offsets.

B.4.2 A Loopable Clock Pattern

ceval	*nclear*	*nread*	*neval*
1	1	0	0
1	1	0	0
1	1	0	0
1	1	0	0
1	1	0	0
1	1	0	0
1	0	0	0
1	0	0	0
0	0	0	1
0	0	1	1
0	0	1	1
0	0	0	1

Table B.5: Clock-pattern: Time sequence of the signals controlling a network cycle.

A clock pattern that allows a good network performance in practice is given in Tab. B.5. It has been tested for interface speeds of up to 100 MHz. With each edge of the interface clock a

transition occurs, i.e., six clock cycles are necessary to perform the shown pattern. Consequently, this clock pattern yields a maximum f_{net} of 16.7 MHz if only direct I/O is used (c.f. Sec. 3.2) or for repetitive cycles without I/O. The clock pattern is loopable (c.f. Sec. 3.5.2) and therefore can be used for any number of network cycles. At an interface speed of 100 MHz this pattern provides a $t_{\text{settling}} = t_{\text{compeval}}$ of 30 ns (c.f. Sec. 3.3.1).

B.5 HAGEN Measurements

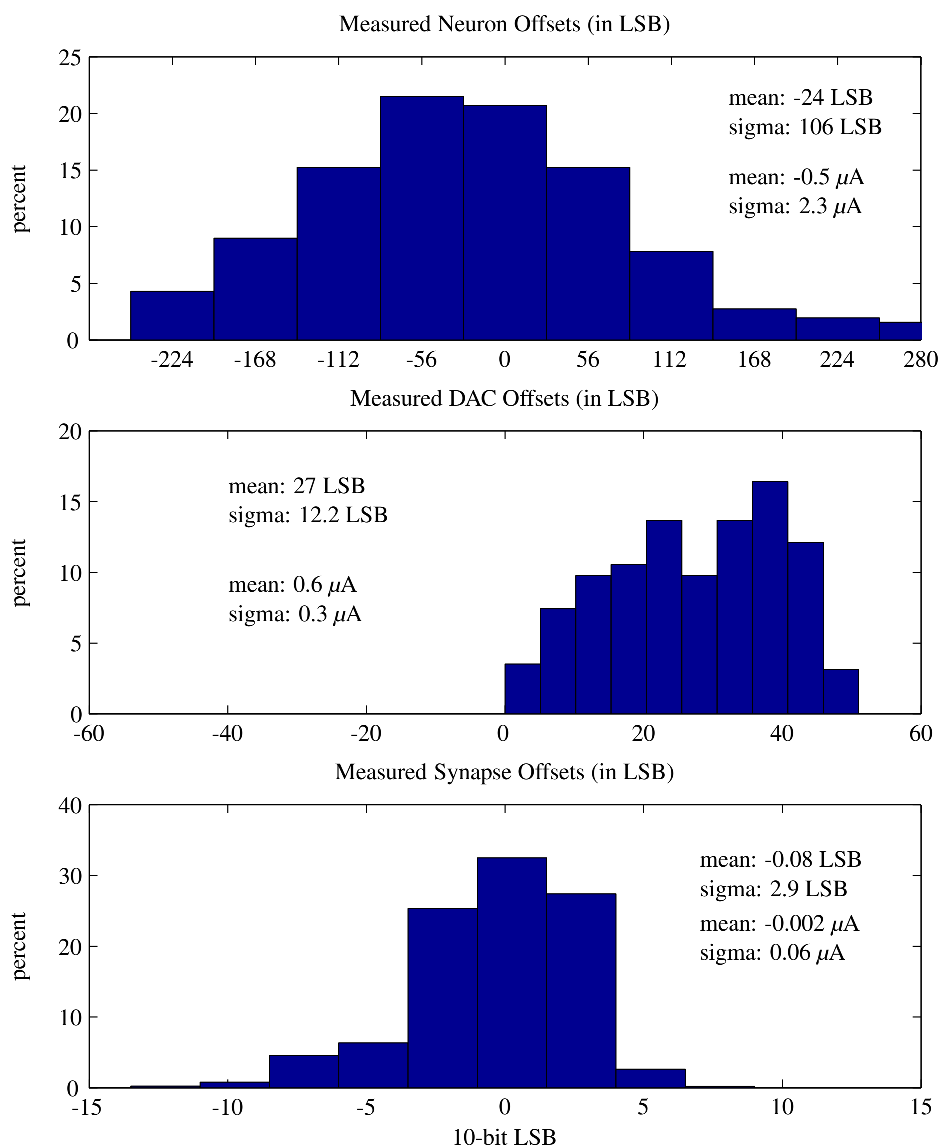


Figure B.3: Measured offsets for the neurons, on-chip DACs, and synapses (top to bottom) of HAGEN 13 used for the experiments of Ch. 5. The offsets are given in 10-bit LSB of the respective on-chip DAC with which they are measured. According to Eq. 3.3, the mean and the standard deviation are as well given in terms of a current. The results are obtained using the method described in [86]. The DAC offsets (middle plot) can only be positive by design; see Sec. 3.3.4.

Appendix C

Experimental Framework Supplements

This page remains empty. The presentation of supplementary information on the experimental framework starts on the next page.

C.1 Peripheral Electronics

signal	pin	pin	signal
GND	1	35	GND
BIAS2	2	36	BIAS0
BIAS1	3	37	BIAS3
+5VA	4	38	AGND
DAp1	5	39	DAm1
DAp2	6	40	DAm2
DAp3	7	41	DAm3
DAp0	8	42	DAm0
DAp18	9	43	DAm18
DAp4	10	44	DAm4
DAp17	1	45	DAm17
DAp5	12	46	DAm5
DAp6	13	47	DAm6
DAp16	14	48	DAm16
DAp7	15	49	DAm7
DAp8	16	50	DAm8
DAp9	17	51	DAm9
dIO11	18	52	dIO10
DAp10	19	53	DAm10
+3.3VD	20	54	+3.3VD
GND	21	55	GND
CLKINp	22	56	CLKINm
CLKOUTp	23	57	CLKOUTm
dIO9	24	58	dIO8
DAp11	25	59	DAm11
dIO7	26	60	dIO6
dIO5	27	61	dIO4
dIO2	28	62	dIO3
dIO0	29	63	dIO1
DAp12	30	64	DAm12
DAp13	31	65	DAm13
DAp14	32	66	DAm14
DAp15	33	67	DAm15
GND	34	68	GND

Table C.1: Pin specification of LVDS interface adapter (female); HAGEN usage.

C.2 Pattern Handling

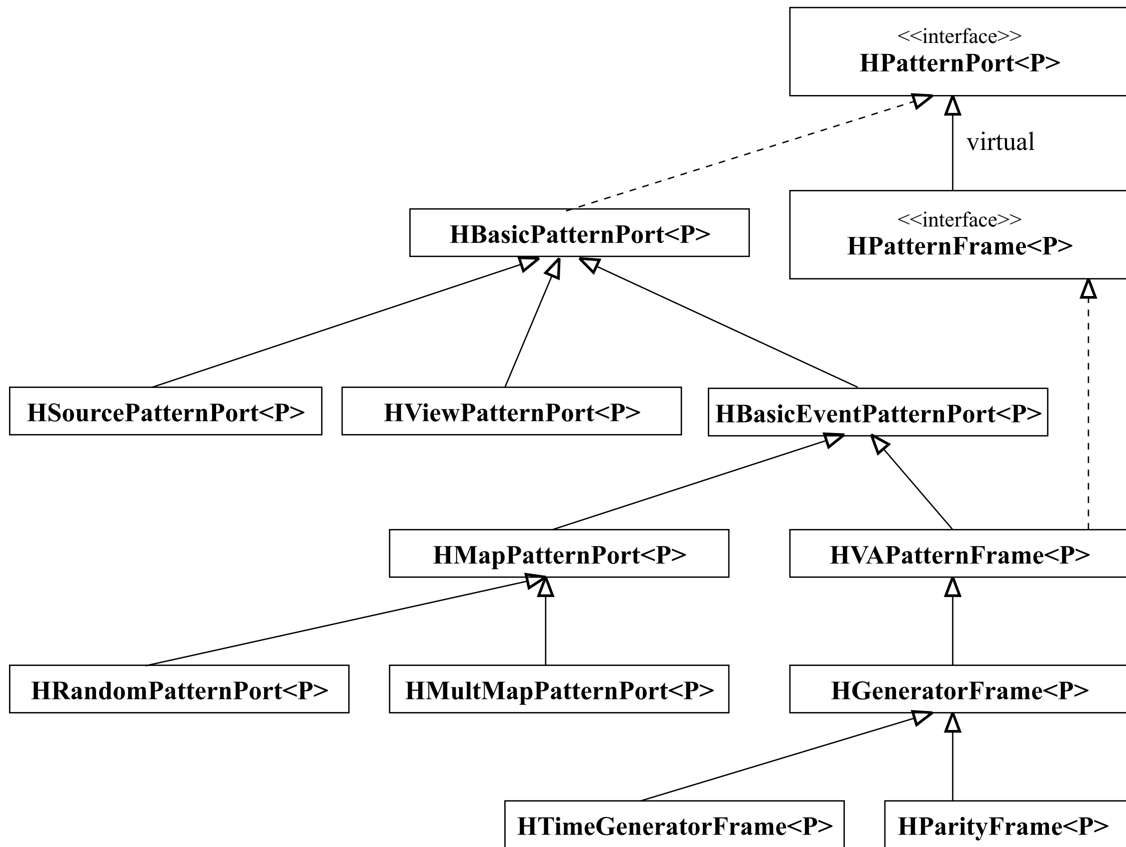


Figure C.1: Simplified class hierarchy of pattern ports.

class	short description
VAPattern	Implementation of a pattern class that can be used as the template parameter class in all pattern handling classes. It allows the conversion of multi-bit values to the respective bit representation and vice versa.
HPatternHandler<P>	Abstract interface class of a pattern handler. It declares all necessary methods to manage chains of pattern ports.
HSimplePatternHandler<P>	Implementation of the HPatternHandler<P> interface that allows to read previously used pattern files for backward compatibility. This handler automatically generates two port chains, one for training, and one for generalization.
HGeneralPatternHandler<P>	Implementation of the HPatternHandler<P> interface with full functionality. It allows to manage arbitrary numbers of pattern ports and stores them in XML.
HPatternPort<P>	Abstract interface class of all pattern ports. It declares all necessary methods to chain ports and to manage the access to patterns.
HPatternFrame<P>	Abstract interface class that extends the HPatternPort<P> interface by method declarations such as add, remove, change, etc., which are necessary for a port that actually holds patterns itself.
HBasicPatternPort<P>	Implementation of the HPatternPort<P> interface that defines the chaining methods and a generic pattern iterator. This is the simplest type of port user defined ports should be derived from.
HSourcePatternPort<P>	Descendant to HBasicPatternPort<P> which should be inherited from if a pattern port is to be implemented that provides actual pattern data of some kind, e.g. a data base. Such a port may be used as a base port.
HViewPatternPort<P>	Descendant to HBasicPatternPort<P> which should be inherited from if a pattern port is to be implemented that provides some kind of view to an adapted pattern port yet does not hold pattern data itself. Such a port must not be used as a base port. If the view port is to react to individual pattern changes of underlying ports, it should be derived from HBasicEventPatternPort<P>.
HBasicEventPatternPort<P>	Descendant to HBasicPatternPort<P> that defines an event mechanism required to signal pattern changes through a chain of ports.

Table C.2: Description of the most important base classes concerned with the pattern handling.

HSimplePatternPort<P>	Class derived from HSourcePatternPort<P> that internally uses an HSimplePatternHandler<P> to make previously used pattern files accessible with the more general HGeneralPatternHandler<P>.
HVAPatternFrame<P>	Class inheriting from HBasicEventPatternPort<P> that furthermore implements the HPatternFrame<P> interface. It uses an STL value array to store the pattern data. This port can be the base port of a port chain.
HGeneratorFrame<P>	Descendant to HVAPatternFrame<P> that simplifies the generation of specific test patterns which simultaneously are stored in its value array. A useful descendant is the HParityFrame<P> that provides n-bit parity patterns.
HTimeGeneratorFrame<P>	Descendant to HGeneratorFrame<P> that simplifies the generation of pattern time series. From this class several pattern ports for the liquid computing experiments are derived, e.g. HPoissonSpikeFrame<P> or HRateFrame<P>.
HMapPatternPort<P>	View port class that allows to reorder and leave out patterns of an adapted pattern port. It does not hold pattern data itself and thus cannot be a base port. It is derived from HBasicEventPatternPort<P>.
HMultMapPatternPort<P>	Extension to HMapPatternPort<P> that allows to change the multiplicity of patterns of an adapted pattern port.
HRandomPatternPort<P>	Extension to HMapPatternPort<P> that provides functionality to randomly reorder the pattern access to an adapted pattern port.
HSievePort<P>	Extension to HViewPatternPort<P> that allows to multiply pattern crumbs of an adapted pattern port. This port is used to provide identical pattern copies to several network blocks.

Table C.3: Description of the most important pattern port classes.

Appendix D

Variable Network Resources

Interfacing multi-valued signals to binary inputs and McCulloch-Pitts output neurons can principally be realized by pooling several synapses and neurons as described in Sec. 1.2.2 using precalculated weights. These results shown here have in part been published in [194].

In order to use precalculated weights in conjunction with the HAGEN prototype, a compensation of the on-chip DAC and neuron offsets should be performed. The technical details concerning the effects have been described in Sec. 3.3.3 and Sec. 3.3.4. A procedure to infer these offsets has been developed in [86] and shortly been described in Sec. 4.4.2. The measurement results of the HAGEN prototype used for the following experiments are presented in Sec. B.5.

While the on-chip DAC and neuron offsets are part of the design as well as their compensation is part of the regular operation of the HAGEN prototype, the synapses (c.f. Sec. 3.3.2) can be used without any fixed-pattern offset correction. The measurements in [86] support that their fixed-pattern variation is of similar magnitude as the temporal variations and below 2.5 LSB of the 10 bit weight amplitude resolution. If one further considers the additional bit for the sign of the synapse, this corresponds to an individual realizable synapse accuracy of 9 to 10 bit for a repetitive programming. This slightly lower accuracy than the nominal one is likely to be caused by cross-coupling from neighboring synapses and imperfect timing of the controlling FPGA (c.f. Sec. 4.2.2).

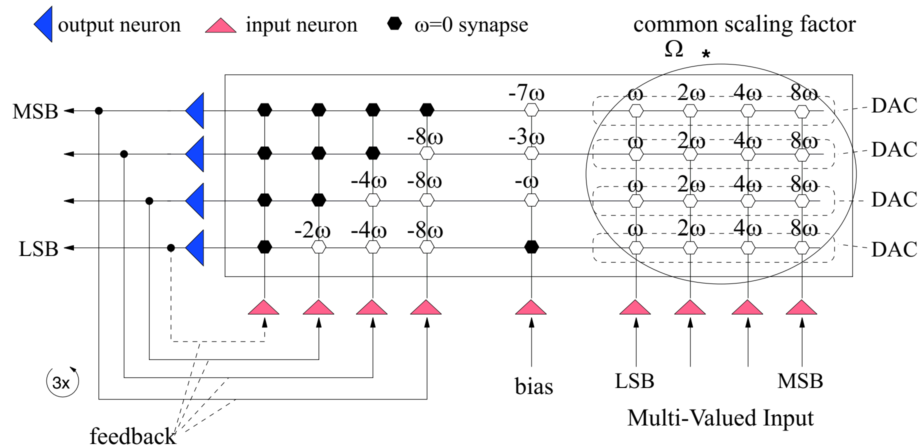


Figure D.1: Network configuration of a 4-bit input being connected to a 4-bit ADC by a 4-to-4-bit synapse of weight Ω. This figure is identical to Fig. 1.5 in Sec. 1.2.2.

Especially, for a highly iterative training with evolutionary algorithms [86] it is desirable not to have each synapse weight be corrected by an additive offset. Simultaneously, multi-bit inputs are often required to adapt real-world problems. In the following, therefore, the HAGEN prototype is used with precalculated weights only having the DAC and neuron offsets compensated. Due to the pooling of several synapses of a neuron in this setup not the individual synapse accuracy is the limit but the relative offsets. This can be assessed with multi-valued inputs according to the setup given in Fig. D.1 (which is repeated here for clarity) and the measurement algorithm described in Sec. 4.4.2. By increasing the number of bits to interface and maintaining the same maximum induced activity, the respective LSB difference in weight is reduced until it is ultimately smaller than the offsets between several synapses of the same neuron.

The actual weights needed to realize an n-bit multi-valued input according to Eq. 1.5 translate with the discrete 10-bit weights of the HAGEN prototype (c.f. Sec. 3.4) to:

$$\hat{\omega}_i = 2^i \hat{\omega} \quad \text{for} \quad 0 \leq i < n, \quad \text{and} \quad \hat{\omega}_i, \hat{\omega} \in [0, 1023] \tag{D.1}$$

and a sign. In the following, the LSB weight quanta $\hat{\omega}$ is chosen not to exceed the dynamic range of a single synapse for the maximum input code, i.e., $\sum_{i=0}^{n-1} \hat{\omega}_i \leq 1023$. As introduced in Sec. 1.2.2, a common scaling factor $\Omega \in [-1, 1]$ can be used to flip the sign or decrease the effective dynamic range. By Eq. 3.3 a weight can equivalently be given as a current. The formulated range limitation for the multi-valued inputs therefore reads:

$$\sum_{i=0}^{n-1} \frac{\hat{\omega}_i}{1023} I_{\text{syn}}^{\text{max}} \leq I_{\text{syn}}^{\text{max}}. \tag{D.2}$$

For all experiments $I_{\text{syn}}^{\text{max}}$ is set to 22 μA; other parameters are V_{cas} = 2.9 V, V_{bias} = 0.91 V, V_{casdac} = 2.9 V; the HAGEN interface speed is 88 MHz. The clock pattern of Sec. B.4 is used.

An n-bit multi-valued input essentially is a binary weighted DAC[1]. Therefore, common performance measures for DACs can be used to assess the quality. The realized linearity of the induced activity in dependence on the input code is measured by the *differential non-linearity* (DNL) [215]. For each two successive input codes, it describes the deviation of the analog output change from the ideal step size:

$$\text{DNL} = \frac{y(k) - y(k-1)}{y_{\text{LSB}}} - 1, \quad 0 < k < 2^n. \tag{D.3}$$

The DNL is calculated after a gain and offset correction. For this purpose a linear least-squares fit is performed on the resulting values for all input codes. The slope of the fit yields y_{LSB}.

In the network block architecture, the analog values $y(k)$ are measured indirectly by sweeping a synapse until it compensates the induced activity by the multi-valued input. As described in Sec. 4.4.2, the measurement yields the input code of the 10-bit on-chip DAC that compensates the activity. With the help of Eq. 3.3, this value can be given in terms of a current, yet it is not an absolute measurement of the weight current.

Fig. D.2 shows a plot of the DNL versus the input code for a measured 5-bit, 6-bit, and 7-bit multi-valued input (top to bottom). The data for the different resolutions was acquired in successive runs. Yet, the location of the multi-valued inputs was chosen to use the same synapses for the more significant bits. The error of determining the DNL from two individual measurements is given by the $\Delta\text{DNL(k)} = \sqrt{\Delta y(k)^2 + \Delta y(k-1)^2}$. The error for the individual measurements is the standard deviation of the mean resulting from 10 measurements. For clarity, in Fig. D.2 the maximum DNL error is plotted, yet not for all input codes.

[1] Beware not to confuse this with the 10-bit on-chip DACs for the actual weight generation (c.f. Sec. 3.4). The latter will always be referred to as *on-chip DACs* or *weight DACs* in this section.

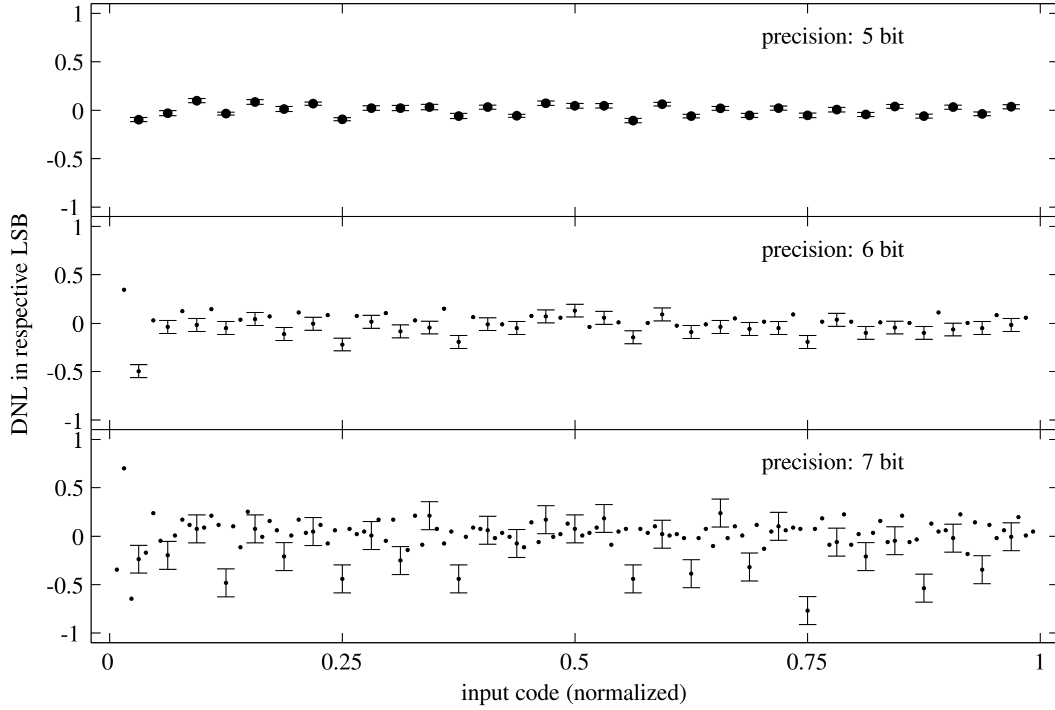


Figure D.2: DNL versus the input code for a measured 5-bit, 6-bit, and 7-bit multi-valued input (top to bottom). The DNL is given in the respective LSB values.

The full scale activity is set to be $0.9 I_{\mathrm{syn}}^{\mathrm{max}}$, i.e. about 20 μA ($\Omega = 0.9$). Accordingly, the LSB currents are about 0.65 μA for the 5-bit input, 0.32 μA for the 6-bit input, and about 0.16 μA for the 7-bit input. The LSB of the weight DAC—in increments of which the activity is measured—is about 20 nA.

As can be seen from the errorbars, the analog values can be reliably reproduced and measured which is in agreement with [86]; since all three shown multi-valued inputs exhibit a DNL < 1 for all input codes, it can be concluded that they are monotonic which is most important for a multi-valued input. Yet, the deviations from the ideal DNL of 0 are systematic and exhibit a typical regularity as observed in binary weighted DACs. If the MSB bit is considered to be the 1st bit, it can be seen that the 3rd bit (responsible for $2^2\hat{\omega}$ in the case of the 5-bit multi-valued input, $2^4\hat{\omega}$ in the 7-bit case respectively) systematically causes small DNL values. The explanation for this is likely to be a larger relative offset of this synapse compared to its neighbors. This is supported by the observation for shifting the DAC location in the network block one column to the left or right: A shift to the left causes the synapse to be responsible for the 4th bit, which changes the regularity appropriately (not shown).

While monotonicity can be concluded immediately from the DNL, it does not give information on the overall linearity of the multi-bit input since it only considers successive codes. A measure that covers the actual shape of the DAC is the *integral non-linearity* (INL):

$$\mathrm{INL} = \frac{y(k)}{y_{\mathrm{LSB}}} - k, \quad 0 \leq k < 2^n. \tag{D.4}$$

Here, the deviation of a measured value from a straight line drawn through the measured endpoints is used; accordingly, y_{LSB} denotes the slope.

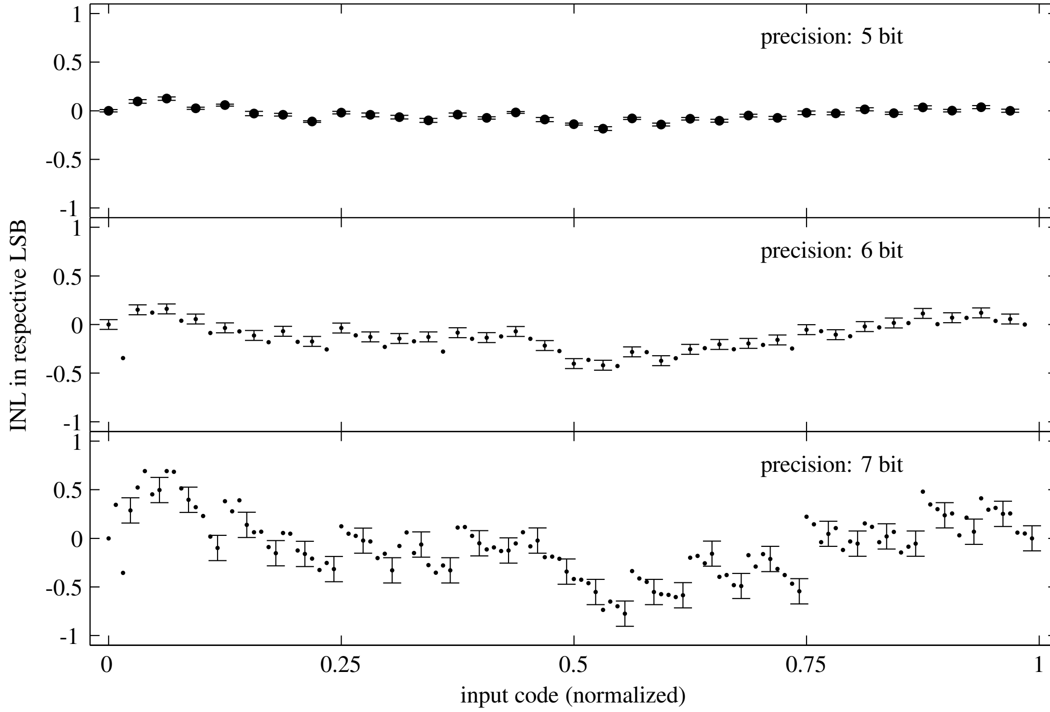


Figure D.3: INL versus the input code for the same measured 5-bit, 6-bit, and 7-bit multi-valued inputs (top to bottom) of Fig. D.2. The INL is given in the respective LSB values.

Fig. D.3 shows the INL versus the input code for the same 5-bit, 6-bit, and 7-bit multi-valued inputs measured above. The errorbars depict the standard deviation of the mean for the repeated measurements at each input code. Again, the maximum error is plotted yet not for all input codes. The systematic deviation from the ideal zero-INL exhibit a regularity that is in agreement with the DNL measurement: In the 7-bit the mismatch of the 3rd bit ($2^4\hat{\omega}$) can be seen which is additionally sub-partitioned by a repetitive mismatch occurring at every 8th input code (4th bit, $2^3\hat{\omega}$).

If several multi-bit inputs are to be connected, the relative accuracy of DACs placed in the same synapse row of the network block is important. Fig. D.4 shows a histogram of the measured DNL values for all input codes of 7-bit DACs successively shifted horizontally in the array, i.e., the histogram is comprised of the DNL values of all input codes for a DAC placed with its MSB in column one, placed in the second column and so on. In this measurement columns 1 to 117 are covered. Of the remaining columns a few have been used for the neuron offset compensation and the actual measurement. The two neurons A and B (left, right respectively) are programmed by different on-chip DACs and as well at different times: Neuron A being the 8th output neuron is the last value to be converted by the on-chip DAC; neuron B being the 9th neuron is the first to be converted by its respective on-chip DAC.

It can be seen that the majority of DNL values (more than 95 %) is below the respective 1 LSB and thus likely to be monotone. All 110 DACs per neuron have furthermore been tested for monotonicity by evaluating successive input codes according to

$$y(k) - y(k-1) > \sqrt{(\Delta y(k))^2 + (\Delta y(k-1))^2}, \quad 0 < k < 2^n \tag{D.5}$$

and said to be monotone if for all input codes this relation holds. This measure yields that in fact 97 % of the DACs connected to neuron A are monotone and 100 % of neuron B.

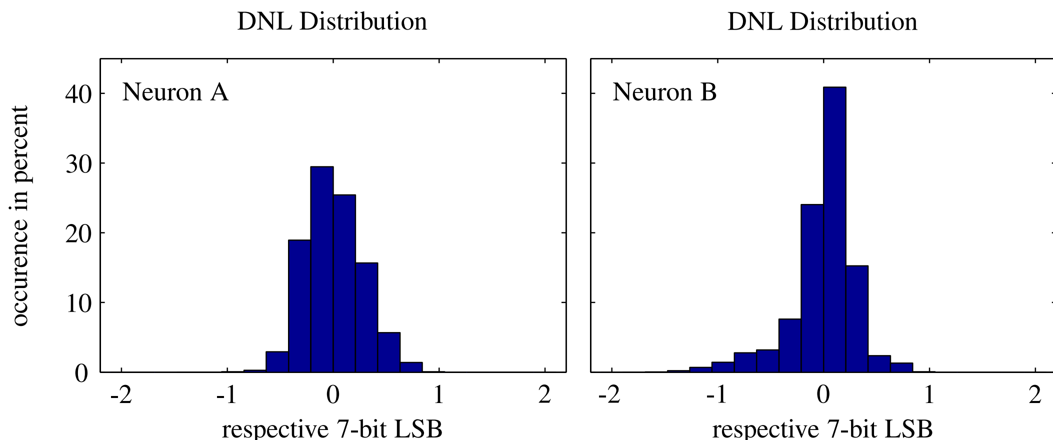


Figure D.4: Histogram of the DNL values measured for all possible input codes and 110 7-bit DACs connected to the same neurons. Neuron A (left) is the 8th output neuron in the network block and programmed last by its on-chip DAC. Neuron B (right) is the 9th output neuron and programmed first by a different on-chip DAC.

Tab. D.1 lists other performance figures for the two analyzed neurons as well as for a complete network block. The measurements have been performed for 5, 6, and 7 bit with 5 repetitive measurements per DAC. Shown are the mean values of the DAC performance figures and the standard deviation of the individual measurement. The gain is normalized to one and the given limit encloses 2/3 of the measured gains. Besides slight differences concerning the offset and the mean of the INL between neuron A and B (especially visible in the 7 bit measurement) both neurons show a very similar performance independent of the on-chip DAC used. Even the distributions measured for all 64 output neurons show good homogeneity. According to the measure of Eq. D.5, 100 % of the 5-bit DACs, 99 % of the 6-bit DACs, and 88 % of the 7-bit DACs are monotone.

	$\overline{\text{DNL}}$	DNL σ_{ind}	$\overline{\text{INL}}$	INL σ_{ind}	$\overline{\text{offset}}$	offset σ_{ind}	$\overline{\text{gain}}$	gain err
5-bit, neuron A	0.01	0.1	0.02	0.1	-0.29	0.03	1	0.003
5-bit, neuron B	0	0.1	-0.15	0.1	-0.03	0.01	1	0.004
whole block	0	0.15	0	0.2	-0.2	0.3	1	0.03
6-bit, neuron A	0.01	0.1	0.05	0.2	-0.6	0.05	1	0.003
6-bit, neuron B	0	0.2	-0.3	0.3	-0.08	0.03	1	0.004
whole block	0	0.3	0	0.4	-0.4	0.7	1	0.03
7-bit, neuron A	0	0.27	0.1	0.5	-1.18	0.14	1	0.003
7-bit, neuron B	0	0.31	-0.6	0.5	-0.2	0.08	1	0.003
whole block	0	0.5	0	0.8	-0.8	1.4	1	0.03

Table D.1: Performance of n-bit DACs connecting to the same neuron (A, B) and for a complete network block. Given are the mean and the standard deviation of the individual measurements.

An application where the homogeneity of multi-bit inputs to several neurons is of importance, are the multi-valued output neurons introduced in Sec. 1.2.2. The setup shown in Fig. D.1 is used to program a 5, 6, and 7 bit ADC comprised of several synapses and neurons. The respective number of multi-bit inputs is used to induce the network activity which is to be measured by the successive approximation setup in several network cycles. Fig. D.5 shows the mean integral non-linearity for each input code. According to Eq. D.4, for this setup it is simply given by the input

code subtracted from the respective mean output value of the ADC. 30 repetitive measurements have been performed and the errorbars indicate the standard deviation of the mean.

The top plot of Fig. D.5 shows the 5-bit setup which yields a perfect performance. The 6-bit performance in the middle plot for most of the input codes is below 1 LSB. Only for input code transitions, where a large bit (MSB or second most important bit) switches on while the lower bits have to switch off, larger deviations can be observed. Furthermore, a network activity which causes only the MSB and the LSB to switch on seems especially susceptible for small offset variations. This may be corrected by adjusting the threshold weights of the ADC (c.f. Fig. D.5). The slightly increased errorbars for larger input codes are likely to be caused by the gain error of the multi-bit inputs. A 7 bit accuracy for the ADC finally is not perfectly realized. Adjusting the threshold may reduce the deviations at the critical input code transitions. Yet, the main error source are probably the seven 7-bit inputs that are required to induce an identical network activity to seven output neurons. While these inputs each may be monotone, the earlier observed non-linearities can account for a deviation in activity from one output neuron to the next and thus affect the successive approximation scheme.

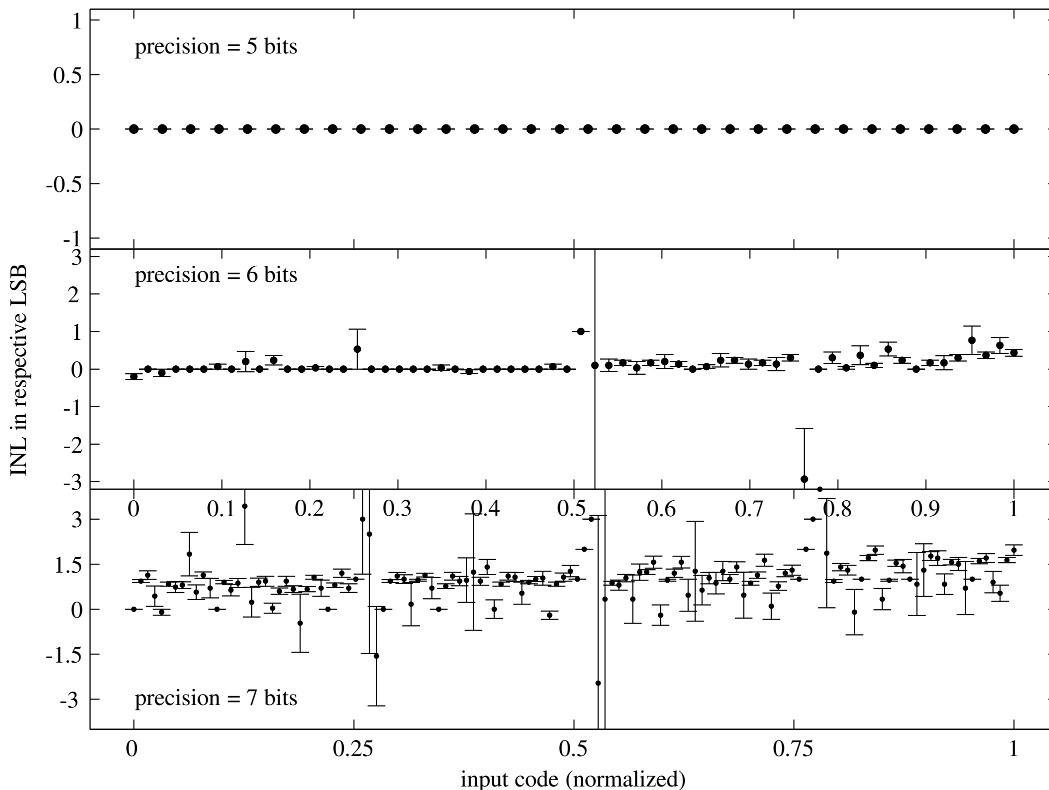


Figure D.5: The mean integral non-linearity versus the input code of a 5 (top), 6, and 7-bit (bottom) ADC built from several synapses and neurons according to Fig. D.1. The network activity is generated by an appropriate number of respective n-bit inputs. Each input code is measured 30 times; the indicated errorbars are the standard deviation of the mean.

The conclusion to be drawn from these experiments is that the HAGEN prototype allows a close to 7 bit accuracy without having to care about individual synapse offsets. Repetitive measurements of the multi-bit inputs show that the limiting effects are fixed-pattern offsets which may be compensated by an individual synapse calibration. This is in agreement with the results of the individual synapse accuracy measured in [86] to be between 9 and 10 bit.

Appendix E

Supplementary Liquid Computing Experiments

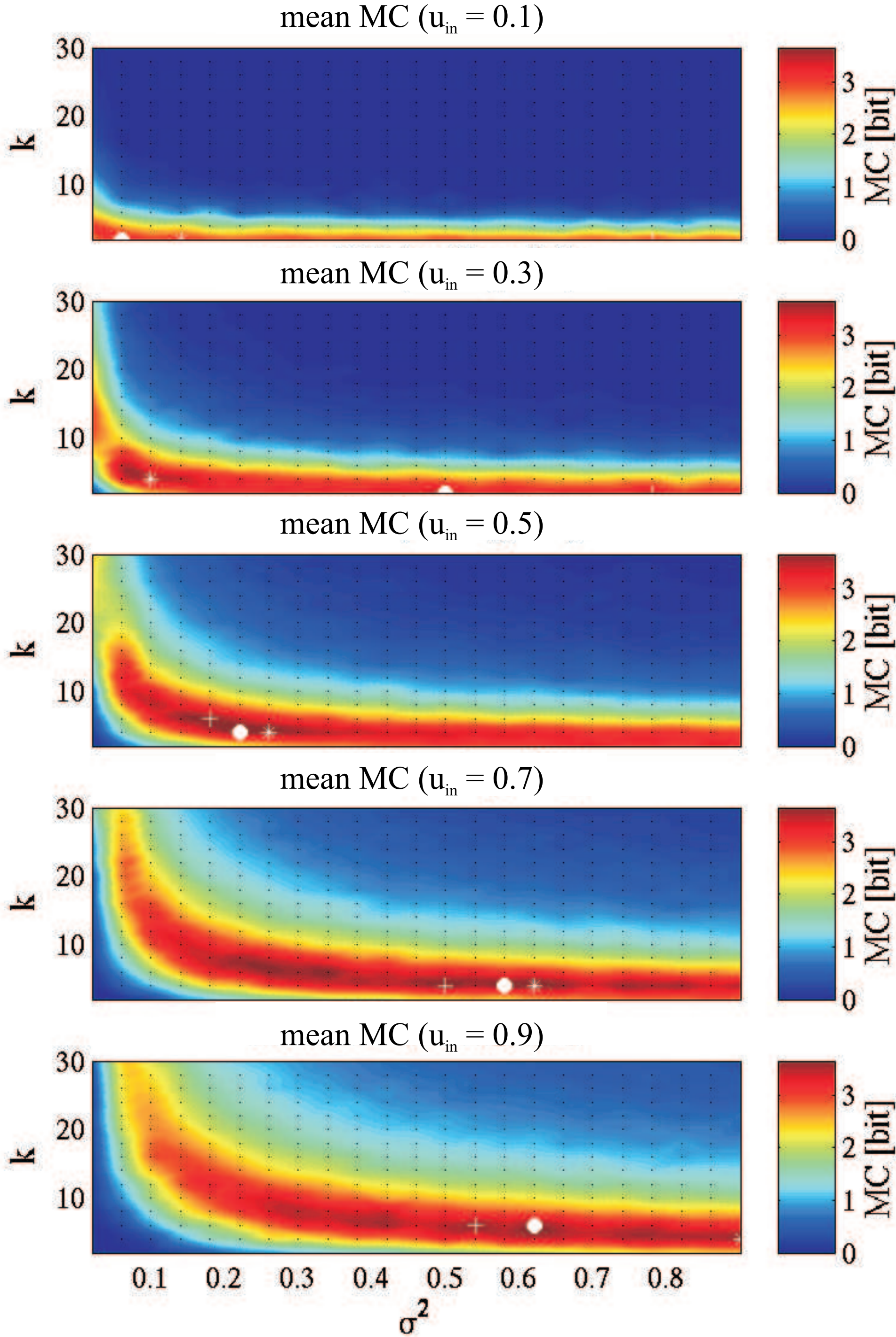


Figure E.1: Evaluation of the input weight scaling using a $\{-1,1\}$-input. Shown are k-σ^2 parameter sweeps varying the weight of the driving input u_{in} from 0.1 (top) to 0.9 (bottom). The color code is the obtained mean memory capacity of readouts for 15 different liquids evaluated per parameter set (small black dots). The respective readouts have been trained to extract the 3-bit time-delayed parity task. Intermediate values are interpolated. The scaling value of 0.5 is used throughout the experiments presented in Ch. 5.

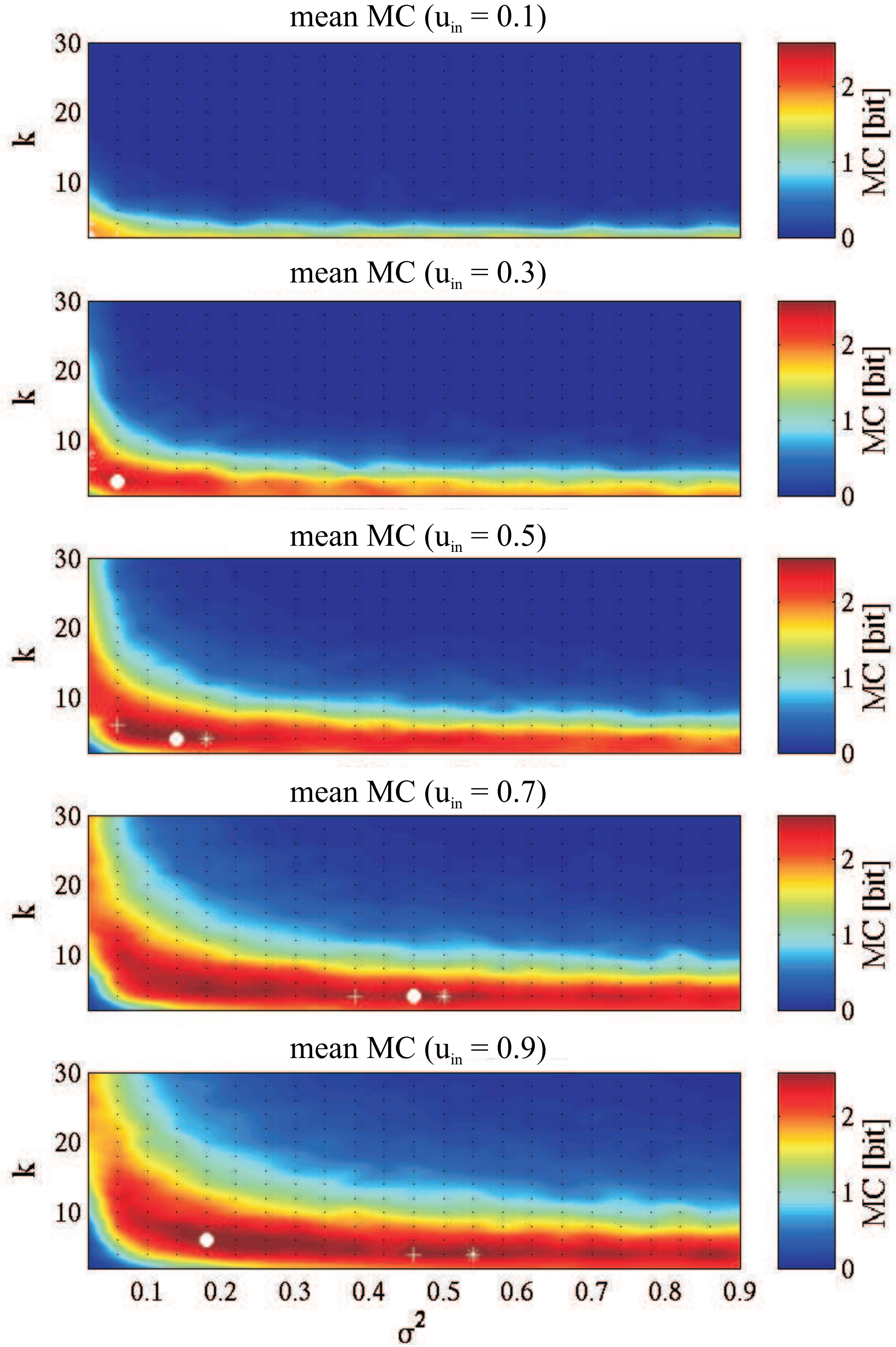


Figure E.2: Evaluation of the input weight scaling using a $\{0,1\}$-input. Shown are k-σ^2 parameter sweeps varying the weight of the driving input u_{in} from 0.1 (top) to 0.9 (bottom), yet. The color code is the obtained mean memory capacity of readouts for 15 different liquids evaluated per parameter set (small black dots). The respective readouts have been trained to extract the 3-bit time-delayed parity task. Intermediate values are interpolated. Due to the overall decrease performance compared to Fig. E.1, complementary inputs are used as described in Sec. 2.3.1.

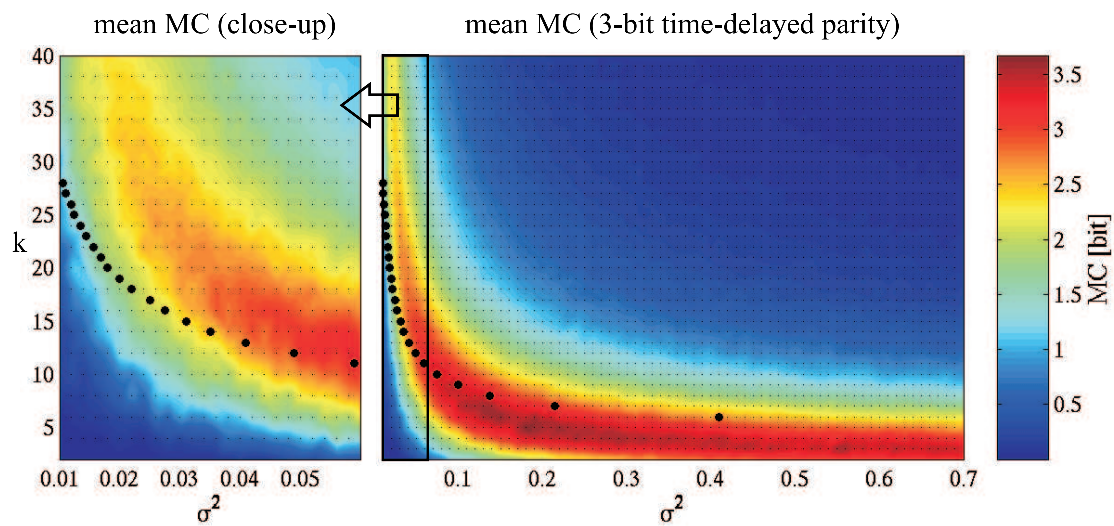


Figure E.3: Parameter sweep of k and σ^2 with a close-up for small σ^2 (left). The right hand side has been shown in the main text in Fig. 5.4. In color code is the mean memory capacity for readouts extracting the 3-bit time-delayed parity task from 30 liquids per measured parameter set (small black dots). The large black dots indicated the predicted critical dynamics by the mean-field theory. It can be seen that the overall performance for LSMs along the critical line with small σ^2 liquids is slightly decreased compared to LSMs with large σ^2 liquids. See main text Sec. 5.2.2.

PSfrag replacements

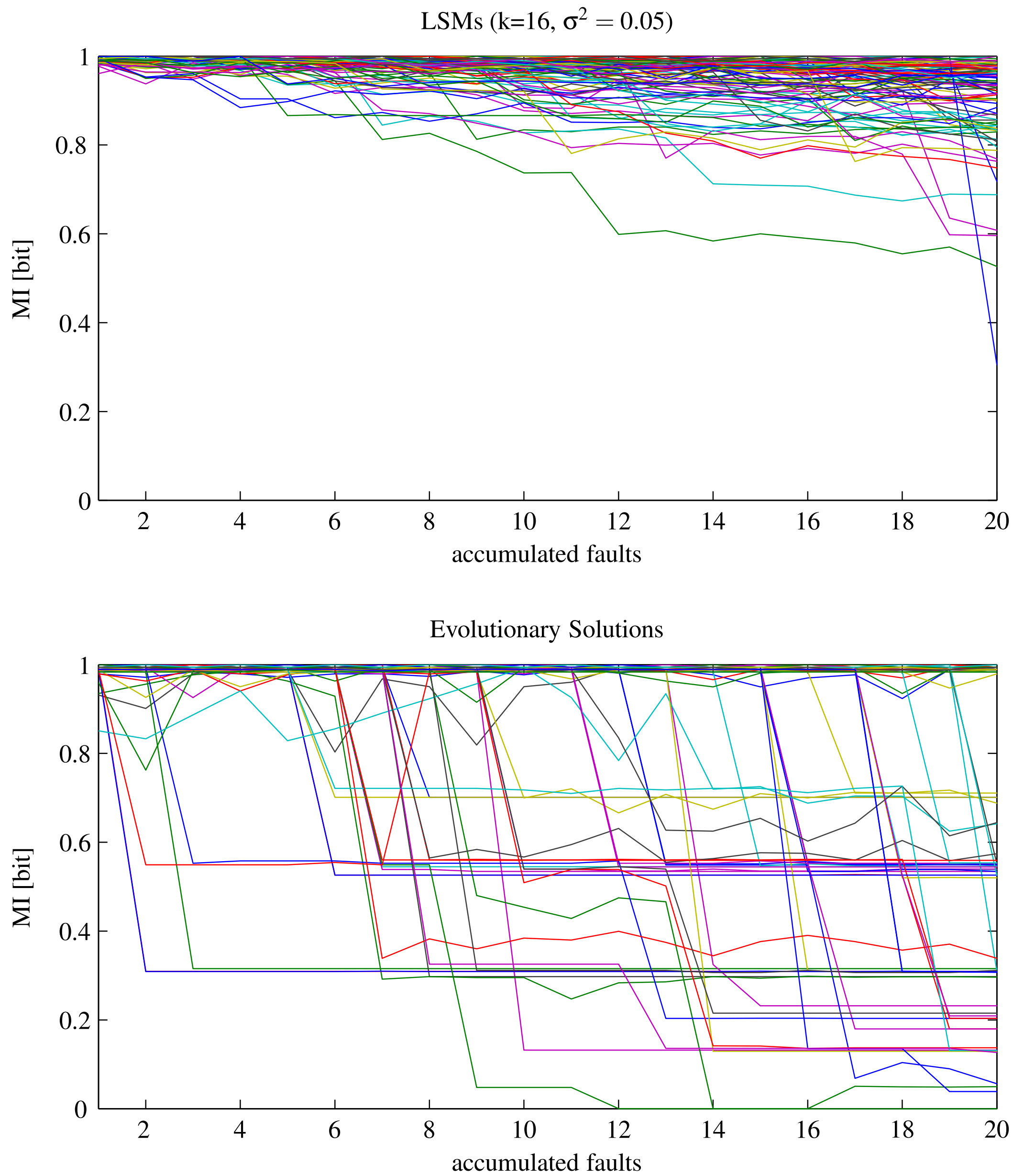


Figure E.4: Detailed version of Fig. 5.20. Shown are 100 LSMs (top) and 100 evolutionary trained feed-forward networks (bottom) tested for a random series of 20 synapse faults each. For the histograms in Fig. 5.21 and Fig. E.5 each of the 100 networks has been tested for 200 different fault series. The LSMs are drawn from $k = 16$, $\sigma^2 = 0.15$ and the evolutionary trained networks are not redundant.

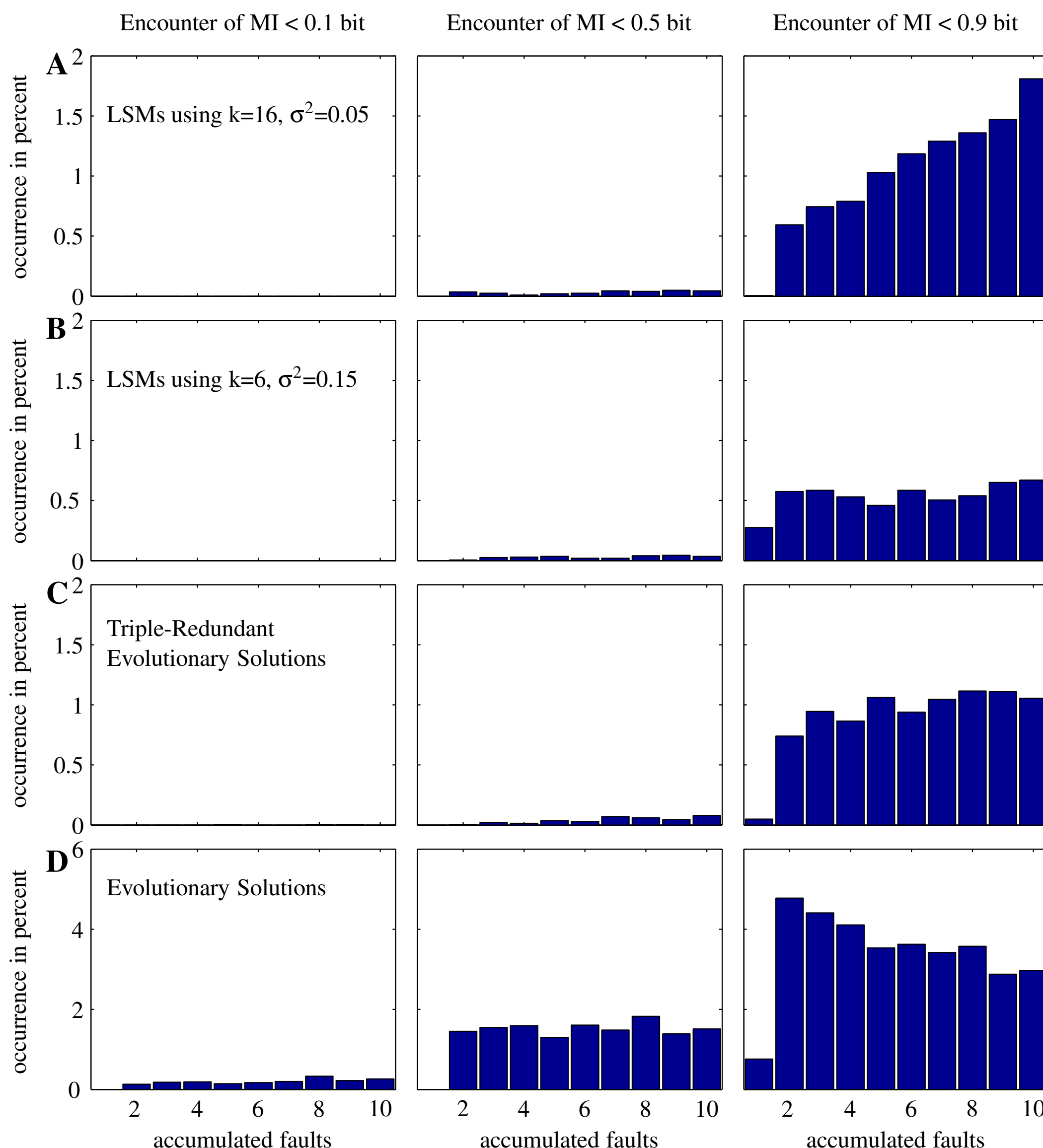


Figure E.5: Encounter of a mutual information below 0.1 (left column), 0.5 (middle column), 0.9 bit (right column) binned for the number of accumulated synapse faults. From top to bottom four different types of networks trained on the 3-bit time parity problem are presented. The rows A and B are LSMs with $k = 16$, $\sigma^2 = 0.05$, $k = 6$, $\sigma^2 = 0.15$ respectively. The row C shows the result of triple-redundant feed-forward networks with a tapped-delay line memory trained by an evolutionary algorithm. Row D yields the results of a non-redundant evolutionary trained networks. Of each type 100 networks have been tested for 200 different series of accumulated faults. The resulting 20000 performance series each are binned. See Sec. 5.3.3 for details.

Acronyms

ADC .... Analog-to-Digital Converter
ANN .... Artificial Neural Network
ASIC .... Application Specific Integrated Circuit
AWG .... Arbitrary Waveform Generator
BGA .... Ball Grid Array
CMC .... Common Mezzanine Card
CMOS .... Complementary Metal Oxide Semiconductor
CPS .... Connections Per Second
CPU .... Central Processing Unit
CUPS .... Connection Updates Per Second
DAC .... Digtial-to-Analog Converter
DMA .... Direct Memory Access
DNL .... Differential Non-Linearity
DSP .... Digital Signal Processor
FPGA .... Field-Programmable Gate Array
GUI .... Graphical User Interface
HAGEN .... Heidelberg Analog Evolvable Neural Network
HANNEE .... Heidelberg Analog Neural Network Evolution Environment
HASTE .... HAGEN Spike Translation Environment
HDL .... Hardware Description Language
INL .... Integral Non-Linearity
LSB .... Least Significant Bit
LSM .... Liquid State Machine
LVDS .... Low Voltage Differential Signalling
LVTTL .... Low Voltage Transistor-Transistor Logic
MC .... Memory Capacity
MGT .... Multi-Gigabit Transceiver
MI .... Mutual Information
MIMD .... Multiple Instruction Multiple Data
MNOS/CCD ...... Metal Nitride Oxide Semiconductor/Charge Coupled Device
MSB .... Most Significant Bit
PCB .... Printed Circuit Board
PCI .... Peripheral Component Interface
PGA .... Pin Grid Array
PLCC .... Plastic Leaded Chip Carrier
SIMD .... Single Instruction Multiple Data
STL .... Standard Template Library
SVM .... Support Vector Machine

VLSI .... Very Large Scale Integration
XML .... Extensible Markup Language

Bibliography

[1] Y. Abu-Mostafa and J. St. Jacques. Information capacity of the hopfield model. *IEEE Transactions on Information Theory*, 31(4):461–464, 1985.

[2] Agilent Technologies, Paolo Alto. *Help Volume: System: Agilent 16700A/B Logic Analysis System*, 2002.

[3] J. Alspector, A. Jayakumar, and S. Luna. Experimental evaluation of leaning in a neural microsystem. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Proceedings of NIPS 1991, Advances in Neural Information Processing Systems 4*, pages 871–878. Morgan Kaufmann Publishers, San Mateo, 1992.

[4] G.M. Amdahl. Validity of the single processor approach to achieve large scale computing capabilities. In *Proceedings of the AFIPS Conference*, volume 30, pages 483–485, Reston,, 1967. AFIPS Press.

[5] E. Anderson and Z. Bai. *LAPACK User's Guide*. SIAM, Philadelphia, 3rd edition, 1999.

[6] J.A. Anderson and E. Rosenfeld. *Neurocomputing - Foundations of Research*. The MIT Press, Cambridge, 1988.

[7] D. Anguita, S. Ridella, and S. Rovetta. Worst case analysis of weight inaccuracy effects in multilayer perceptrons. *IEEE Transactions on Neural Networks*, 10(2):415–418, 1999.

[8] D. Anguita and M. Valle. Perspectives on dedicated hardware implementations. In M. Verleysen, editor, *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*, pages 45–53, Evere, 2001. D-Facto.

[9] A.J. Annema and H. Wallings. Analog weight adaptation hardware. *Neural Processing Letters*, 2(3):1–4, 1995.

[10] ANSI/TIA/EIA-644. *Electrical Characteristics of Low Voltage Differential Signalling (LVDS)*, March 1996.

[11] K. Asanoviċ and N. Morgan. Experimental determination of precision requirements for back-propagation training of artificial neural networks. In U. Ramacher, U. Rückert, and J.A. Nossek, editors, *Proceedings of the Second International Conference MicroNeuro'91*, pages 9–15, München, 1991.

[12] P. Auer, H. Burgsteiner, and W. Maass. Reducing communication for distributed learning in neural networks. In J.R. Dorronsoro, editor, *Proceedings of the International Conference on Artificial Neural Networks*, pages 123–128, Heidelberg, 2002. Springer.

[13] P. Auer, H. Burgsteiner, and W. Maass. A learning rule for very simple universal approximators consisting of a single layer of perceptrons. Submitted for publication, February 2005.

[14] Austria Mikro Systeme International AG, Unterpremstätten. *0.35 µm CMOS Design Rules*, 2.0 edition, July 2000.

[15] Austria Mikro Systeme International AG, Unterpremstätten. *0.35 µm CMOS Process Parameters*, 2.0 edition, January 2001.

[16] I. Aybay, S. Cetinkaya, and U. Halici. Classification of neural network hardware. *Neural Network World, IDG Co.*, 6(1):11–29, 1996.

[17] M. Bazes. Two novel fully complementary self-biased CMOS differential amplifiers. *IEEE Journal of Solid-State Circuits*, 26(2):165–168, February 1991.

[18] J. Becker. Ein FPGA-basiertes Testsystem für gemischt analog/digitale ASICs. Diploma Thesis (German), Ruprecht-Karls-Universität Heidelberg, 2001.

[19] V. Beiu. Digital integrated circuit implementations. In E. Fiesler and R. Beale, editors, *The Handbook of Neural Computation*, chapter E1.4. Institute of Physics Publishing and Oxford University Publishing, New York, January 1997.

[20] V. Beiu. VLSI implementations of threshold logic - a comprehensive survey. *IEEE Transactions on Neural Networks*, 14(5):1217–1243, 2003.

[21] N. Bertschinger and T. Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413 – 1436, July 2004.

[22] T. Blickle and L. Thiele. A comparison of selection schemes used in genetic algorithms. Technical Report 11, Swiss Federal Institute of Technology (ETH), Zürich, 1995.

[23] H.D. Block. The perceptron: a model for brain functioning. I. *Reviews of Modern Physics*, 34:123–135, 1962.

[24] B.E. Boser, I.M. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[25] B.E. Boser, E. Säckinger, J. Bromley, J. LeCun, and L.D. Jackel. An analog neural network processor with programmable topology. *IEEE Journal of Solid-State Circuits*, 26(12):2017–2025, 1991.

[26] S. Boyd and L.O. Chua. Fading memory and the problem of approximating nonlinear operators with volterra series. *IEEE Transactions on Circuits and Systems*, 32(11):1150–1161, 1985.

[27] A.E. Brain. The simulation of neural elements by electrical networks based on multi-aperture magnetic cores. In *Proceedings of the IRE*, pages 49–52, 1961.

[28] A. Breidenassel, K. Meier, and J. Schemmel. A flexible scheme for adaptive integration time control. In *Proceedings of the IEEE Sensors 2004, Vienna*, To be published.

[29] I.N. Bronstein, K.A. Semendjajew, and G. Musiol. *Taschenbuch der Mathematik*. Harri Deutsch, 2000.

[30] D. Brooks. *Signal Integrity Issues and Printed Circuit Board Design*. Prentice Hall PTR, Indianapolis, June 2003.

[31] D. Brüderle. Implementing spike-based computation on a hardware perceptron. Diploma Thesis, Ruprecht-Karls-Universität Heidelberg, October 2004.

[32] A.E. Bryson and Y.-C. Ho. *Applied Optimal Control*. Ginn, Waltham, 1969.

[33] H.C. Card, D.K. McNeill, C.R. Schneider, and R.S. Schneider. The impact of VLSI fabrication on neural learning. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 985–988, 1995.

[34] G. Cauwenberghs. Learning on silicon: A survey. In G. Cauwenberghs and M.A. Bayoumi, editors, *Learning on Silicon: Adaptive VLSI Neural Systems*, chapter 1, pages 1–29. Kluwer Academic Publisher, Norwell, 1999.

[35] E. Chicca, D. Badoni, V. Dante, M. D'Andreagiovanni, G. Salina, L. Carota, S. Fusi, and P. Del Giudice. A VLSI recurrent network of integrate-and-fire neurons connected by plastic synapses with long-term storage. *IEEE Transactions on Neural Networks*, 14(5):1297–1307, 2003.

[36] N. Christianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, 2000.

[37] L.O. Chua, G. Gulak, E. Pierzchall, and A. Rodriguez-Vazquez, editors. *Analog Integrated Circuits and Signal Processing - Special Issue on Cellular Neural Networks and Analog VLSI*, 15(3), Boston, March 1998. Kluwer Academic Publishers.

[38] L.O. Chua and L. Yang. Cellular neural networks: Theory. *IEEE Transactions on Circuits And Systems*, 35(10):1257–1272, 1988.

[39] HyperTransport Technology Consortium. *HyperTransport I/O Link Specification*. Sunnyvale, 1.03 edition, 2001.

[40] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with application in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14:326–334, 1965.

[41] H.D. Crane. A high-speed logic system using magnetic elements and connecting wire only. In *Proceedings of the IRE*, pages 63–73, 1959.

[42] P. Cusinato, M. Bruccoleri, D.D. Caviglia, and M. Valle. Analysis of the behavior of a dynamic latch comparator. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 45(3):294–298, March 1998.

[43] D.J. Deleganes, M. Barany, D. Chow, T.D. Fletcher, G.L. Geannopoulos, K. Kreitzer, A.P. Singh, and S.B. Wijeratne. LVS technology for the intel pentium 4 processor on 90nm technology. *Intel Technology Journal*, 08(01):43–54, February 2004.

[44] B. Denby, P. Garda, B. Dranado, C. Kiesling, J.-C. Prévotet, and A. Wassatsch. Fast triggering in high-energy physics experiments using hardware neural networks. *IEEE Transactions on Neural Networks*, 14(5):1010–1026, 2003.

[45] B. Derrida and Y. Pomeau. Random networks of automata: A simple annealed approximation. *Europhysics Letters*, 1(2):45–49, January 1986.

[46] Design Automation Standards Committee of the IEEE Computer Society, New York. *VHDL Language Reference Manual, IEEE Std. 1076.1*, 1997.

[47] C. Diorio, P. Hasler, B.A. Minch, and C.A. Mead. A single-transistor silicon synapse. *IEEE Transactions on Electron Devices*, 43(11):1972–1996, 1996.

[48] C. Diorio, D. Hsu, and M. Figueroa. Adaptive CMOS: From biological inspiration to system-on-a-chip. *Proceedings of the IEEE*, 90(3):345–357, 2002.

[49] B.K. Dolenko and H.C. Card. Tolerance to analog hardware of on-chip learning in backpropagation networks. *IEEE Transactions on Neural Networks*, 6(5):1045–1052, 1995.

[50] J.J. Dongarra, J. DuCroz, I.S. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16:1–17, 1990.

[51] J.J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson. An extended set of fortran basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–32, 1988.

[52] S. Draghici. Guest editorial - new trends in neural network implementations. *International Journal of Neural Systems*, 10(3), 2000.

[53] C.E. Elger, A.D. Friederici, C. Koch, H. Luhmann, C. von der Malsburg, R. Menzel, H. Monyer, F. Rösler, G. Roth, H. Scheich, and W. Singer. Das Manifest - Elf führende Neurowissenschaftler über Gegenwart und Zukunft der Hirnforschung. *Gehirn und Geist*, 6:30, 2004.

[54] D. Erdogmus, D. Rende, J.C. Principe, and T.F. Wong. Nonlinear channel equalization using multilayer perceptrons with information-theoretic criterion. In *Proceedings of the IEEE Signal Processing Society Workshop*, pages 443–451, 2001.

[55] N.H. Farhat, D. Psaltis, A. Prata, and E. Paek. Optical implementation of the hopfield model. *Applied Optics*, 24:1469–1475, 1985.

[56] E. Fielser, A. Choudry, and H.J. Caulfield. A weight discretization paradigm for optical neural networks. In *Proceedings of the International Congress on Optical Science and Engineering (SPIE vol. 1281)*, pages 164–173, Bellingham, 1990. SPIE.

[57] J. Fieres, A. Grübl, S. Philipp, K. Meier, J. Schemmel, and F. Schürmann. A platform for parallel operation of VLSI neural networks. In L.S. Smith, A. Hussain, and I. Aleksander, editors, *Proceedings of the 2004 Brain Inspired Cognitive Systems Conference (BICS2004)*, pages NC4.3 1–7, University of Stirling, 2004. Published on CD-ROM.

[58] E. Fiesler, A. Choudry, and H.J. Caulfield. Weight discretization in backward error propagation neural networks. *Neural Networks*, 1(supplement 1):380, 1988.

[59] M. Figueroa, S. Bridges, and C. Diorio. On-chip compensation of device-mismatch effects in analog VLSI neural networks. In L.K. Saul, Y. Weiss, and L. Bottou, editors, *Proceedings of NIPS 2004, Advances in Neural Information Processing Systems 17*. The MIT Press, 2005. To appear.

[60] M.J. Flynn. Some computer organization and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960, 1972.

[61] P. Földesy. Trends in design of massively parallel coprocessors implemented in digital ASICs. In *Proceedings of the International Joint Conference on Neural Networks*, pages 3131–3136, 2004.

[62] Free Software Foundation. GNU general public license. http://www.gnu.org/copyleft/gpl.html, June 1991. Version 2.

[63] Free Software Foundation, Boston. *GNU Compiler Collection*, 3.3.5 edition, 2003.

[64] K. Fukushima, S. Miyake, and T. Ito. Neocognitron: a neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:826–834, 1983.

[65] S. Fuse, P. Del Giudice, and D.J. Amit. Neurophysiology of a VLSI spiking neural network: LANN21. *Proceedings of the International Joint Conference on Neural Networks*, 3:121–126, 2000.

[66] M. Galassi et al. *GNU Scientific Library Reference Manual*, 2nd edition. http://www.gnu.org/software/gsl.

[67] R.L. Geiger, P.E. Allen, and N.R. Strader. *VLSI - Design Techniques for Analog and Digital Circuits*. Materials Science and Engineering. McGraw-Hill, Inc., New York, 1990.

[68] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6:721–741, 1984.

[69] R. Genov. Kerneltron: Support vector “machine” in silicon. *IEEE Transactions on Neural Networks*, 14(5):1426–1434, 2003.

[70] R. Genov and G. Cauwenberghs. Massively parallel inner-product array processor. In *Proceedings of the International Joint Conference on Neural Networks*, pages 183–188, 2001.

[71] G.G.E. Gielen and R.A. Rutenbar. Computer-aided design of analog and mixed-signal integrated circuits. *Proceedings of the IEEE*, 88(12):1825–1852, 2000.

[72] M. Glesner and M. Pöchmüller. *Neurocomputers*. Neural Computing Series. Chapman & Hall, London, 1994.

[73] D.H. Goldberg, G. Cauwenberghs, and A.G. Andreou. Analog VLSI spiking neural network with address domain probabilistic synapses. *The IEEE International Symposium on Circuits and Systems*, pages 241–244, 2001.

[74] M.J. Goossens, C.J.M. Verhoeven, and A.H.M. Roermund. Concepts for ultra low-power and very-high-density single-electron neural networks. In *Proceedings of the International Symposium on Nonlinear Theory and Its Applications (NOLTA)*, pages 679–682, 1995.

[75] P.R. Gray, P.J. Hurst, S.H. Lewis, and R.G. Meyer. *Analysis and Design of Analog Integrated Circuits*. John Wiley & Sons, 4th edition, 2001.

[76] LVDS Group. *LVDS Owner's Manual*. National Semiconductor, Santa Clara, 3rd edition, Spring 2004.

[77] A. Grübl. Eine FPGA-basierte Plattform für Neuronale Netze. Diploma Thesis (German), Ruprecht-Karls-Universität Heidelberg, January 2004.

[78] G. Han and E. Sánchez-Sinencio. CMOS transconductance multipliers: A tutorial. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(12):1550–1563, 1998.

[79] D. Hanselman and B. Littlefield. *MATLAB Version 6 - User's Guide*. The MathWorks, Inc., Natick, R13 edition, 2002.

[80] J.C. Hay, F.C. Martin, and C.W. Wightman. The MARK I perceptron - design and performance. *IRE National Convention Record*, 2:78–87, 1960.

[81] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, 1999.

[82] R. Hecht-Nielsen. *Neurocomputing*. Addison-Wesley, 1989.

[83] J. Heemskerk. Overview of neural hardware. In: Neurocomputers for Brain-Style Processing. Design, Implementation and Application *PhD thesis*, 1995.

[84] Hewlett-Packard, Böblingen. *HP 8130A Programmable Pulse Generator Operating and Programming Manual*, 1st edition, 1989.

[85] H. Hinsch. *Elektronik - Ein Werkzeug für Naturwissenschaftler*. Springer-Verlag, Berlin, 1996.

[86] S.G. Hohmann. *Stepwise Evolutionary Training Strategies for Hardware Neural Networks*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, Heidelberg, 2005.

[87] S.G. Hohmann, J. Fieres, K. Meier, J. Schemmel, T. Schmittz, and F. Schürmann. Training fast mixed-signal neural networks for data classification. In *Proceedings of the 2004 International Joint Conference on Neural Networks (IJCNN'04)*, pages 2647–2652. IEEE Press, 2004.

[88] S.G. Hohmann, J. Schemmel, F. Schürmann, and K. Meier. Exploring the parameter space of a genetic algorithm for training an analog neural network. In W.B. Langdon et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2002*, pages 375–382. Morgan Kaufmann Publishers, July 2002.

[89] S.G. Hohmann, J. Schemmel, F. Schürmann, and K. Meier. Predicting protein cellular localization sites with a hardware analog neural network. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'03)*, pages 381–386. IEEE Press, July 2003.

[90] M. Holler. VLSI implementation of learning and memory systems: A review. In *Proceedings of NIPS 1987, Advances in Neural Information Processing Systems 3*. Lippmann, R.P., 1991.

[91] M. Holler, S. Tam, H. Castro, and R. Benson. An electronically trainable artificial neural network (ETANN) with 10240 “floating gate synapses”. In *Proceedings of the International Joint Conference on Neural Networks*, pages 191–196, 1989.

[92] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, N. Reading, 1980.

[93] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982.

[94] J.J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81:3088–3092, 1984.

[95] J.J. Hopfield and C. Brody. The mus silicium (sonoram desert sand mouse). http://neuron.princeton.edu/∼moment/Organism/, 2000.

[96] J.J. Hopfield and C.D. Brody. What is a moment? transient synchrony as a collective mechanism for spatiotemporal integration. *Proceedings of the National Academy of Science*, 98(3):1282–1287, 2001.

[97] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[98] D.A. Huffmann. A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40:1098–1101, 1952.

[99] IBM, 590 Madison Avenue, New York. *Electronic Data-Processing Machines - Type 704*, -1 edition, January 1956.

[100] P. Ienne. Architectures for neuro-computers: review and performance evaluation. Technical Report 21/93, Swiss Federal Institute of Technology, Lausanne, 1993.

[101] Intel Corporation. *Intel C++ Compiler for Linux Systems - User's Guide (Version 8.0)*, 2003. http://www.intel.com/software/products/compilers/clin.

[102] Intel Corporation. *Intel Math Kernel Library Reference Manual*, 6004 edition, 2003. http://www.intel.com/software/products/mkl.

[103] ITRS. The international technology roadmap for semiconductors. http://public.itrs.net, July 2003.

[104] ITRS. The international technology roadmap for semiconductors - emerging research devices. http://public.itrs.net, July 2003.

[105] H. Jaeger. The “echo state” approach to analysing and training recurrent neural networks. GMD Report 148, German National Research Center for Information Technology, 2001.

[106] H. Jaeger. A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the “echo state network” approach. GMD Report 159, German National Research Center for Information Technology, 2002.

[107] H. Jaeger and H. Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication. *Science*, 304:78–80, April 2004.

[108] N. M. Josuttis. *The C++ Standard Library*. Addison-Wesley, Reading, 1999.

[109] Jungo Ltd., Netanya. *Windriver 6 User's Manual*, 2003.

[110] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[111] J. Kivinen. Online learning of linear classifiers. In *Advanced lectures on machine learning*, pages 235–257. Springer-Verlag, New York, 2003.

[112] S.C. Kleene. Representations of events in nerve nets and finite automata. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, Princeton, 1956.

[113] K. Kloukinas, F. Faccio, A. Marchioro, and P. Moreira. Development of a radiation tolerant 2.0 v standard cell library using a commercial deep submicron CMOS technology for the LHC experiments. *Nuclear Physics B - Proceedings Supplements*, 78(1-3):625–634, 1999.

[114] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[115] A.H. Kramer. Array-based analog computation. *IEEE Micro*, 16(5):20–29, 1996.

[116] K.R. Laker and W.M.C. Sansen. *Design of Analog Integrated Circuits and Systems*. McGraw-Hill, New York, 1994.

[117] J. Langeheine, K. Meier, and J. Schemmel. Intrinsic evolution of analog electronic circuits using a CMOS FPTA chip. In G. Bugeda, J.-A. Désidéri, J. Périaux, M. Schoenauer, and G. Winter, editors, *Proceedings of the Conference on Evolutionary Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems (EUROGEN 2003)*, Barcelona, September 2003. CIMNE. Published on CD-ROM.

[118] C.G. Langton. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D*, 42:12–37, 1990.

[119] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software*, 5:308–325, 1979.

[120] Y. LeCun. Une procédure d'apprentissage pour réseau á seuil asymetrique. *Cognitiva*, 85:599–604, 1985.

[121] C.G. Lee and D.J. DeVries. Initial results on the performance and cost of vector microprocessors. *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 171–182, 1997.

[122] P.H.W. Leong and M.A Jabri. A low-power VLSI arrhythmia classifier. *IEEE Transactions on Neural Networks*, 6(6):1435–1445, 1995.

[123] C.S. Leung and R. Setiono. Efficient neural network training algorithm for the Cray YM-P. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1943–1946, Piscataway, 1993. IEEE.

[124] J. Liang, T. McConaghy, A. Kochlan, T. Pham, and G. Hertz. Intelligent systems for analog circuit design automation: A survey. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, 2001. IIIS (International Institute of Informatics and Systemics).

[125] B. Linares-Barranco, A.G. Andreou, G. Indiveri, and T. Shibata, editors. *IEEE Transactions on Neural Networks - Special Issue on neural networks hardware implementations*, 14(5), Boston, September 2003. IEEE Press.

[126] C. Lindsey and T. Lindblad. Survey of neural network hardware. In S.K. Rogers and D.W. Ruck, editors, *Proceedings of SPIE 1995*, volume 2492, pages 1194–1205, April 1995.

[127] S.-C. Liu, J. Kramer, G. Indiveri, T. Delbrück, and R. Douglas. *Analog VLSI: Circuits and Principles*. The MIT Press, Cambridge, 2002.

[128] R.F. Lyon and C.A. Mead. An analog electronic cochlea. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36:1119–1134, 1988.

[129] W. Maass, R.A. Legenstein, and N. Bertschinger. Methods for estimating the computational power and generalization capability of neural microcircuits. In L.K. Saul, Y. Weiss, and L. Bottou, editors, *Proceedings of NIPS 2004, Advances in Neural Information Processing Systems 17*, Cambridge, 2005. The MIT Press. To appear.

[130] W. Maass, T. Natschläger, and H. Markram. A fresh look at real-time computation in generic recurrent microcircuits. Submitted for publication in 2002 (online available at http://www.lsm.tugraz.at), 2002.

[131] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbation. *Neural Computation*, 14(11):2531–2560, 2002.

[132] W. Maass, T. Natschläger, and H. Markram. On the computational power of circuits of spiking neurons. *Journal of Physiology*, 2004. In press.

[133] S. McBader, P. Lee, and A. Sartori. The impact of modern FPGA architectures on neural hardware: A case study of the TOTEM neural processor. In *Proceedings of the International Joint Conference on Neural Networks*, pages 3149–3154, 2004.

[134] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin on Mathematical Biophysics*, 5:115–133, 1943.

[135] C.A. Mead. *Analog VLSI and Neural Systems*. Addison Wesley, Reading, MA, 1989.

[136] J. A. Mears. *Transmission Line RAPIDDESIGNER Operation and Applications Guide*. National Semiconductor, Santa Clara, May 1996.

[137] K. Meier et al. FACETS - fast analog computing with emergent transient states (15879). EU FP6-2004-IST-FETPI, 2005.

[138] M. Minsky. *Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain Model Problem*. PhD thesis, Princeton University, 1954.

[139] M. Minsky. *Computation: Finite and infinite machines*. Prentice-Hall, 1967.

[140] M. Minsky and S. Papert. *Perceptrons*. The MIT Press, Boston, 1967.

[141] Model Technology Inc., Portland. *ModelSim SE/EE Tutorial Version 5.4a*, April 2000.

[142] P. Moerland and E. Fiesler. Neural network adaptations to hardware implementations. In E. Fiesler and R. Beale, editors, *The Handbook of Neural Computation*, chapter E1.2. Institute of Physics Publishing and Oxford University Publishing, New York, January 1997.

[143] D. Moore. Draft standard for a common mezzanine card family: CMC. Technical report, IEEE Standards Department, 1995.

[144] T. Morie. Analog VLSI implementation of self-learning neural networks. In G. Cauwenberghs and M.A. Bayoumi, editors, *Learning on Silicon: Adaptive VLSI Neural Systems*, chapter 10, pages 213–242. Kluwer Academic Publisher, Norwell, 1999.

[145] M.L. Mumford, D.K Andes, and L.R. Kern. The Mod2 neurocomputer system design. *IEEE Transactions on Neural Networks*, 3(3):423–433, 1992.

[146] A.F. Murray. Silicon implementations of neural networks. *IEE Proceedings F*, 138(1):3–12, February 1991.

[147] S.K. Nair and J. Moon. Data storage channel equalization using neural networks. *IEEE Transactions on Neural Networks*, 8(5):1037–1048, 1997.

[148] T. Natschläger. Neuro micro circuits portal web site. http://www.lsm.tugraz.at, 2003.

[149] T. Natschläger and N. Bertschinger. A mathematica notebook with an implementation of the mean-field theory for randomly connected recurrent networks of threshold gates. http://www.igi.tugraz.at/tnatschl/edge-of-chaos/mean-field-notebook.nb, 2004.

[150] T. Natschläger and N. Bertschinger. Supplementary information to the mean-field theory for randomly connected recurrent networks of threshold gates. http://www.igi.tugraz.at/tnatschl/edge-of-chaos/mean-field-supplement.pdf, 2004.

[151] T. Natschläger, N. Bertschinger, and R. Legenstein. At the edge of chaos: Real-time computations and self-organized criticality in recurrent neural networks. In L.K. Saul, Y. Weiss, and L. Bottou, editors, *Proceedings of NIPS 2004, Advances in Neural Information Processing Systems 17*, Cambridge, 2005. The MIT Press. To appear.

[152] T. Natschläger and W. Maass. Spiking neurons and the induction of finite state machines. *Theoretical Computer Science: Special Issue on Natural Computing*, 287:251–265, 2002.

[153] T. Natschläger and W. Maass. Information dynamics and emergent computation in recurrent circuits of spiking neurons. In S. Thrun, L.K. Saul, and B. Schölkopf, editors, *Proceedings of NIPS 2003, Advances in Neural Information Processing Systems 16*, pages 1255–1262, Cambridge, 2004. The MIT Press.

[154] T. Natschläger, W. Maass, and H. Markram. The "liquid computer": A novel strategy for real-time computing on time series. *Telematik*, 8(1):39–43, 2002.

[155] T. Natschläger, H. Markram, and W. Maass. Computer models and analysis tools for neural microcircuits. In R. Kötter, editor, *A Practical Guide to Neuroscience Databases and Associated Tools*, chapter 9. Kluver Academic Publishers, Boston, 2002. In press.

[156] T. Nordström and B. Svensson. Using and designing massively parallel computers for artificial neural networks. *Journal of Parallel and Distributed Computing*, 14(3):260, 285 1991.

[157] A.R. Omondi. Neurocomputers: A dead end? *International Journal of Neural Systems*, 10(6):475–481, 2000.

[158] N. Packard. Adaptation towards the edge of chaos. In J.A.S. Mandell and M.F. Shlesinger, editors, *Dynamic Patterns in Complex Systems*, pages 293–301. World Scientific, 1988.

[159] D.B. Parker. Learning logic. Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, 1985.

[160] E. Pasero and M. Perri. Hw-sw codesign of a flexible neural controller through a FPGA-based neural network programmed in VHDL. In *Proceedings of the International Joint Conference on Neural Networks*, pages 3161–3166, 2004.

[161] J.C. Patra, N.P. Ranendra, B. Rameswar, and G. Panda. Nonlinear channel equalization for QAM signal constellation using artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 29(2):262–271, 1999.

[162] G. O. Penokie. *Working Draft SCSI Parallel Interface-2 (SPI-2)*. American National Standard of Accredited Standards Committee NCITS, Washington, DC, 20b edition, April 1998.

[163] PLX Technology, Inc., Sunnyvale. *PLX 9054 Data Book*, 2.1 edition, January 2000.

[164] C. Pohl, M. Franzmeier, M. Porrmann, and U. Rückert. gNBX - reconfigurable hardware acceleration of self-organizing maps. In *Proceedings of the International Conference on Field Programmable Technology*, 6 - 8 December 2004.

[165] Polar Instruments Ltd., Guernsey. *Si600b Controlled Impedance Field Solver*, 2001.

[166] L. Prechelt. Proben1 — A set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, 38 pages, Fakultät für Informatik, Universität Karlsruhe, 1994.

[167] W.H. Press, S.A. Teukolsky, W.T. Vetterlling, and B.P. Flannery. *Numerical Recipes in C++*. Cambridge University Press, Cambridge, 2nd edition, 2002.

[168] A. Prieto and A. Andreou, editors. *Analog Integrated Circuits and Signal Processing - Special Issue on Microelectronics for Bio-Inspired Systems*, 30(2), Boston, February 2002. Kluwer Academic Publishers.

[169] U. Ramacher. SYNAPSE-X - a general-purpose neurocomputer architecture. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 3, pages 2168–2176. IEEE Press, 1991.

[170] L.M. Reyneri. Unification of neural and wavelet networks and fuzzy systems. *IEEE Transactions on Neural Networks*, 10(4):801–814, July 1999.

[171] L.M. Reyneri. Implementation issues of neuro-fuzzy hardware: Going toward HW/SW codesign. *IEEE Transactions on Neural Networks*, 14(1):176–194, January 2003.

[172] L.M. Reyneri, M. Chiaberge, L. Lavagno, B. Pino, and E. Miranda. Simulink-based HW/SW codesign of embedded neuro-fuzzy systems. *International Journal of Neural Systems*, 10(3):211–226, 2000.

[173] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[174] F. Rosenblatt. Perceptron simulation experiments. In *Proceedings of the IRE*, pages 301–309, 1960.

[175] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, New York, 1962.

[176] T. Roska and L.O. Chua. The CNN universal machine: an analogic array computer. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 40(3):163–173, 1993.

[177] U. Rückert. Microelectronic implementation of neural networks. In H. Hüning, S. Neuhauser, M. Raus, and W. Ritschel, editors, *Proceedings of the Workshop on Neural Networks at RTWH Aachen*, pages 77–86, July 1993.

[178] D.E. Rumelhart, G.E. Hinton, and R.J Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing*, chapter I, pages 318–362. The MIT Press, Cambridge, 1986.

[179] D.E. Rumelhart and J.L. McClelland, editors. *Parallel Distributed Processing*. The MIT Press, Cambridge, 1986.

[180] J.P. Sage, K. Thompson, and R.S. Withers. An artificial neural network integrated circuit based upon MNOS/CCD principles. In J.S. Denker, editor, *Neural Networks for Computing*, volume 151 of *AIP Conference Proceedings*, pages 381–385. American Institute for Physics, New York, 1986.

[181] R. Sarpeshkar. Analog versus digital: Extrapolating from electronics to neurobiology. *Neural Computation*, 10(7):1601–1638, 1998.

[182] S. Satyanarayana, P. Tsividis, and H.P. Graf. A reconfigurable VLSI neural network. *IEEE Journal of Solid-State Circuits*, 27(1):67–81, 1992.

[183] J. Schemmel. *An Integrated Analog Network for Image Processing*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 1999.

[184] J. Schemmel, S. Hohmann, K. Meier, and F. Schürmann. A mixed-mode analog neural network using current-steering synapses. *Analog Integrated Circuits and Signal Processing*, 38(2-3):233–244, 2004.

[185] J. Schemmel, K. Meier, and E. Mueller. A new VLSI model of neural microcircuits including spike time dependent plasticity. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'04)*, pages 1711–1716, 2004.

[186] J. Schemmel, K. Meier, and F. Schürmann. A VLSI implementation of an analog neural network suited for genetic algorithms. In *Proceedings of the International Conference on Evolvable Systems ICES 2001*, pages 50–61. Springer Verlag, 2001.

[187] J. Schemmel, F. Schürmann, S. Hohmann, and K. Meier. An integrated mixed-mode neural network architecture for megasynapse ANNs. In *Proceedings of the 2002 International Joint Conference on Neural Networks IJCNN'02*, pages 2704–2710. IEEE Press, 2002.

[188] D.C. Schmidt. C++ network programming with patterns, frameworks, and ACE. http://www.cs.wustl.edu/ schmidt/ACE.html.

[189] T. Schmitz, S. Hohmann, K. Meier, J. Schemmel, and F. Schürmann. Speeding up Hardware Evolution: A Coprocessor for Evolutionary Algorithms. In Andy M. Tyrrell, Pauline C. Haddow, and Jim Torresen, editors, *Proceedings of the 5th International Conference on Evolvable Systems ICES 2003*, pages 274–285. Springer Verlag, 2003.

[190] T. Schoenauer, S. Atasoy, N. Mehrtash, and H. Klar. Neuropipe-chip: A digital neuro-processor for spiking neural networks. *IEEE Transactions on Neural Networks*, 13(1):205–213, 2002.

[191] T. Schoenauer, A. Jahnke, U. Roth, and H. Klar. Digital neurohardware: Principles and perspectives. In *Proceedings of Neuronale Netze in der Anwendung NN'98*, pages 101–106, Magdeburg, 1998.

[192] R. Schüffny, A. Graupner, and J. Schreiter. Hardware for neural networks. *Proceedings of the 4th International Workshop on Neural Networks in Applications*, 1999.

[193] F. Schürmann, S. Hohmann, J. Schemmel, and K. Meier. Towards an Artificial Neural Network Framework. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R.S. Zebulum, editors, *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 266–273. IEEE Computer Society, 2002.

[194] F. Schürmann, S. G. Hohmann, K. Meier, and J. Schemmel. Interfacing binary networks to multi-valued signals. In *Supplementary Proceedings of the Joint International Conference ICANN/ICONIP*, pages 430–433. IEEE Press, 2003.

[195] F. Schürmann, K. Meier, and J. Schemmel. Edge of chaos computation in mixed mode VLSI - "a hard liquid". In L.K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, Cambride, 2005. The MIT Press.

[196] K.D. Schwartzel Jr. Summing amplifier. U.S.Patent 2401779, June 11 1946.

[197] D.J. Sebald. Support vector machine techniques for nonlinear equalization. *IEEE Transactions on Signal Processing*, 48(11):3217–3226, 2000.

[198] C.E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:623–656, October 1948.

[199] H. Shawkey, H. Elsimary, H. Ragaai, and H. Haddara. A low power VLSI neural network based arrhythmia classifier. In *Proceedings of the IEEE Symposium on Computer-Based Medical Systems*, pages 282–288, 1998.

[200] M. Shelley, D. McLaughlin, R. Shapley, and J. Wielaard. States of high conductance in a large-scale model of the visual cortex. *Journal of Computational Neuroscience*, 13:93–109, 2002.

[201] T. Shibata and T. Ohmi. An intelligent MOS transistor featuring gate-level weighted sum and threshold operations. *IEEE Journal of Solid-State Circuits*, 39:1444–1455, 1992.

[202] H.T. Siegelmann and E.D. Sonntag. Turing computability with neural nets. *Applied Mathematics Letters*, 4:77–80, 1991.

[203] A. Singer. Exploiting the inherent parallelism of artificial neural networks to achieve 1300 million interconnections per second. In B. Angeniol and B. Widrow, editors, *Proceedings of the International Neural Networks Conference (INNC)*, volume 2, pages 656–660, Dordrecht, 1990. Kluwer.

[204] S. Singhal and L. Wu. Training feed-forward networks with the extended Kalman filter. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 1187–1190, 1989.

[205] A. Sinsel. Linuxportierung auf einen eingebetteten PowerPC 405 zur Steuerung eines neuronalen Netzwerkes. Diploma Thesis (German), Ruprecht-Karls-Universität Heidelberg, 2003.

[206] M.A. Sivilotti, M.A. Mahowald, and C.A. Mead. Real-time visual computations using analog CMOS processing arrays. In P. Losleben, editor, *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*, pages 295–312, Cambridge, 1987. The MIT Press.

[207] S.H. Strogatz. *Nonlinear Dynamics and Chaos - With Applications to Physics, Biology, Chemistry, and Engineering*. Addison-Wesley, Reading, 1994.

[208] B. Stroustrup. *The C++ Programming Language*. Addison Wesley, Reading, August 1997.

[209] Synopsys, Inc., Mountain View. *FPGA Compiler II*, 2001.08-fc3.6 edition, June 2001.

[210] Tektronix, Inc., Wilsonville. *TDS 500D, TDS 600B & TDS700D Digitizing Oscilloscopes - User Manual*, 1998. Document 071-0130-03.

[211] Tektronix, Inc., Beaverton. *P7330 3.5 GHz Differential Probe*, 2003. Document 071-0758-04.

[212] Trolltech AS. The Qt ® application development framework. http://www.trolltech.com/products/qt/.

[213] UMC. *0.18um Mixed Mode/RFCMOS Technology 1.8V/3.3V 1P6M Electrical Design Rule*, 1.1 edition, October 2000.

[214] M. Valle. Analog VLSI implementation of artificial neural networks with supervised on-chip learning. *Analog Integrated Circuits and Signal Processing*, 33:263–287, 2002.

[215] R. van de Plassche. *Integrated Analog-to-Digital and Digital-to-Analog Converters*. Kluwer Academic Publishers, Dordrecht, 1994.

[216] V.N. Vapnik. *Statistical Learning Theory*. John Wiley, New York, 1998.

[217] M. Vertregt, H. Tuinhout, and M. Pelgrom. Matching of MOS transistors. Electronic Laboratories Advanced Engineering Course on Transistor-Level Analog IC Design, June 2004.

[218] E.A. Vittoz. Analog VLSI signal processing: Why, where and how? *Analog Integrated Circuits and Signal Processing*, 8(1):27–44, 1994.

[219] E.A. Vittoz. Analog VLSI implementation of neural networks. In E. Fiesler and R. Beale, editors, *The Handbook of Neural Computation*, chapter E1.3. Institute of Physics Publishing and Oxford University Publishing, New York, January 1997.

[220] E.A. Vittoz. Analog layout techniques. Electronics Laboratories Advanced Course on CMOS & BiCMOS IC Design, August-September 2000.

[221] L.G. Vuurpijl and Th.E. Schouten. Suitability of transputers for neural network simulations. In W. Joosen and E. Milgrom, editors, *Parallel Computing: From Theory to Sound Practise*, pages 528–537. IOS Press, Amsterdam, 1992.

[222] P. Werbos. *Beyond Regression*. PhD thesis, Harvard University, 1974.

[223] P. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[224] B. Widrow. Generalization and information storage in networks of adeline 'neurons'. In M.C. Yovitz, G.T. Jacobi, and G.D. Goldstein, editors, *Self-Organizing Systems*, pages 435–461. Spartan Books, Washington, DC, 1962.

[225] B. Widrow and M.E. Hoff. Adaptive switching circuits. *IRE WESCON Convention Record*, pages 96–104, 1960.

[226] J.J. Wikner. *Studies on CMOS Digital-to-Analog Converters*. PhD thesis, Linköpings Universitet, 2001.

[227] R.J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computations*, 1:270–280, 1989.

[228] R. Windner. Single-stage logic. *Proceedings of the AIEE General Meeting*, 1960.

[229] W. Wolf. A decade of hardware/software co-design. *IEEE Computer*, pages 38–43, 2003.

[230] C. Wolff, G. Hartmann, and U. Rückert. ParSPIKE - a parallel DSP-accelerator for dynamic simulation of large spiking neural networks. In *Proceedings of the 7th International Conference on Microelectronics for Neural, Fuzzy, and Bio-Inspired Systems (MicroNeuro99)*, pages 324–331, Granada, 1999.

[231] Würth Elektronik GmbH. *Design-Regeln für MicroVia- und TWINflex-Leiterplatten*, 2.7 edition, February 2000. http://www.we-online.de.

[232] Xilinx, Inc., San Jose. *Virtex-II Pro Platform FPGA Handbook*, 1.0 edition, January 2002.

[233] Xilinx, Inc., San Jose. *VirtexE 1.8V Field Programmable Gate Array*, 2.3 edition, July 2002.

[234] Xilinx, Inc., San Jose. *ISE Quick Start Tutorial 6.x*, 2003.

[235] Xilinx, Inc., San Jose. *XC9536XL High Performance CPLD*, September 2004.

[236] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[237] J. Zhu and P. Sutton. FPGA implementations of neural networks - a survey of a decade of progress. In P.Y.K. Cheung, G.A. Constantinides, and J.T. de Sousa, editors, *Proceedings of the International Conference on Field Programmable Logic and Applications*, volume 2778 of *Lecture Notes in Computer Science*, pages 1062–1066. Springer, 2003.

[238] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.

Acknowledgements

An dieser Stelle möchte ich mich bei den Menschen bedanken, die mich während der vergangenen Jahre in meiner Arbeit begleitet und zu dem Gelingen dieses Unterfangens in der einen oder anderen Weise beigetragen haben. Zunächst möchte ich meinen Dank an die Kollegen und Mitarbeiter des Kirchhoff Instituts für Physik richten und dabei insbesondere erwähnen:

- Herrn Prof. Dr. Karlheinz Meier für die offene und nette Aufnahme in seiner Arbeitsgruppe Electronic Vision(s), die bei meinem Eintritt noch *Elektronisches Sehen* hieß, aber zu jeder Zeit visionär war. Die Arbeit hat meinen Horizont in jeder Hinsicht erweitert.

- Herrn Prof. Dr. Norbert Herrmann für die freundliche Übernahme der Zweitkorrektur meiner Arbeit und die Grundsteinlegung, sich mit Physik auf Englisch beschäftigen zu können.

- Dr. Johannes Schemmel für sehr viele Dinge, aber insbesondere dafür, daß er mich an seinen Visionen hat teilhaben lassen und mir die Hardwareseite des modernen Computings gezeigt hat. Seine Kritik habe ich zu jeder Zeit als besonders konstruktiv geschätzt, aber aus unserer gemeinsamen Bürozeit nicht nur fachlich profitiert. Ich bin mir sicher, daß uns trotz leichter Gramm- und Zentimetermeinungsverschiedenheiten ein tiefer Spaß an coolen Gadgets verbindet.

- Steffen Hohmann für die jahrelange konstruktive Zusammenarbeit an allen Aspekten des HAGEN- und des Vorgängerprojektes. Für einen 'aus dem Softwarezimmer' hat er wirklich großartige Lötkünste gezeigt und nicht zuletzt dadurch zum Gelingen beigetragen. Insbesondere aber möchte ich mich für seine Arbeit an HANNEE bedanken und die theoretisch fundierten Antworten zu allen möglichen Fragen.

- Eilif Mueller für das Einbringen der Maass'schen Ideen in meine Weltsicht der neuronalen Netzwerke und seine Geduld, mein Amerikanisch über sich ergehen zu lassen – in Sprache wie Schrift. Glücklicherweise hat das seinem Deutschlernen keinen Abbruch getan.

- Tillmann Schmitz für seine gewissenhafte Arbeit im Zwielicht von Hard- und Software.

- Andy Grübl für seine produktive Diplomandenzeit und den Sinn für das Handfeste in allen Belangen der Hardwarebeschaffung, -bestückung, und des Toolniederringens.

- Stefan Philipp für rettende Diskussion in kniffligen Computerbelangen und die Pflege der Windowswelt in unserer Arbeitsgruppe.

- Michael Reuß und Daniel Brüderle für ihr Interesse an meinen *liquids* und die damit verbundene konstruktive Kritik.

- Johannes Fieres für wirklich coolen Code in HANNEE und einmalige Kommentare im CVS.

- Andreas Breidenassel, Jörg Langeheine und Dr. Thorsten Maucher für ihre jederzeitige Hilfsbereitschaft und langjährige verläßliche Kollegenschaft.
- Martin Trefzer insbesondere dafür, daß ich mit ihm einen gewissenhaften Nachfolger für die Dekanatsbetreuung gefunden habe.
- Ralf Achenbach für seine aufopferungsvolle Hilfe in allen Laborfragen, ohne die ein solches Hardwareprojekt niemals möglich wäre.
- Markus Dorn für seine Pflege der Design-Tools, die den schrecklichen Alltag dieser Software erträglich gemacht hat.
- Den Mitarbeitern der Elektronikwerkstatt für ihre Unterstützung in vielen Belangen.
- Frau Ellen-Maria Schmidt für ihre unbändige Geduld, auch noch den kleinsten Kondensator in das SAP Bestellwesen einzutragen und damit meine Arbeit und die der restlichen Visions unglaublich zu erleichtern.
- Dr. Robert Weis für seine durchdachte Netzwerkinfrastruktur im KIP und seine immerwährende Bereitwilligkeit, bei Linux-, Workaround-, und sogar Dekanatsfragen zu helfen.
- Kathrin Weber für ihre unglaublichen Künste, Literatur besorgen zu können, die jeder zitiert, aber kaum einer gelesen zu haben scheint.
- Nils Bertschinger, der zwar nicht am Kirchhoff Institut ist, dem ich aber für seine Unterstützung in *liquid*-Fragen und die Erhellung seines Mathematica-Codes danken möchte.

Ohne einige andere Menschen wäre diese Arbeit tatsächlich auch nicht möglich gewesen. Das reicht von den Anfängen bis hin zu den Kleinigkeiten und Wundern des Alltags, und deswegen möchte ich erwähnen:

- Christian für langjährige treue Nachbarschaft und Hilfe in allen Lebenslagen, von Funknetzwerken bis zu roten Zwiebeln.
- Meine liebste Julia, die mein Leben verschönert und viele Dinge in mir weckt, welche zu erkennen ich vorher nicht vermochte.
- Meine lieben Eltern, die immer – ausnahmslos – zu mir gehalten haben, wofür ich ihnen gar nicht genug danken kann.