Eine digitale Emulationsplattform für pulscodierte neuronale Netze mit adaptiven Synapsengewichten

vorgelegt von

Diplom-Ingenieur

Heik Heinrich Hellmich

aus Berlin-Pankow

von der Fakultät IV

- Elektrotechnik und Informatik -

der Technischen Universität Berlin

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

Dr. - Ing.

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. rer. nat. Klaus Obermayer, Technische Universität Berlin

1. Gutachter: Prof. Dr.-Ing. Heinrich Klar, Technische Universität Berlin

2. Gutachter: Prof. Dr.-Ing. Ulrich Kleine, Otto-von-Guericke-Universität Magdeburg

Tag der wissenschaftlichen Aussprache: 26. März 2008

Berlin 2008

D83

〝オパ、 約束　を　守りました。〞

Diese Arbeit ist meinem Großvater Tsuneo Ikemi (池見恒夫) gewidmet.

Kurzfassung

Die Simulation von großen pulscodierten neuronalen Netzen für die Evaluation einer biologisch motivierten Bildverarbeitung ist auf Einprozessor-Systemen, wie PCs oder Workstations, immer noch sehr zeitineffizient. Entsprechend den Beobachtungen in der Biologie verarbeiten pulscodierte neuronale Netze Informationen anhand von Pulsen und zeichnen sich durch die Fähigkeit aus, sich während der Verarbeitung äußeren Reizen anzupassen. Diese Anpassungsfähigkeit gilt als Lernprozess und setzt voraus, dass sich die Gewichte von Netzwerkverbindungen (Synapsen) zwischen Netzwerkelementen (Neuronen) dynamisch verändern können. Den größten Flaschenhals während der Simulation stellt der sequentielle Zugriff auf den Speicher von dynamischen Netzwerkparametern (Membranpotentiale und adaptive Synapsengewichte) dar, der für die Berechnung der Neuronenzustände notwendig ist.

In dieser Arbeit wird ein neuartiges Architekturkonzept und die Entwicklung einer digitalen Emulationsplattform für die Simulationsbeschleunigung von pulscodierten neuronalen Netzen vorgestellt. Diese Emulationsplattform repräsentiert die erste Experimentierplattform für ein pulscodiertes Neuronenmodell mit adaptiven Synapsengewichten für den ganzheitlichen Bildverarbeitungsprozess. Bei dieser Plattform stehen die Evaluation der Netzwerkdynamik und die Bereitstellung einer hohen Flexibilität bei der Implementierung im Vordergrund. Die Architektur der Emulationsplattform realisiert ein effizientes Kommunikationsprotokoll zwischen der parallelen Berechnung von Netzwerktopologien und Neuronenzuständen und basiert auf einem noch nicht existierenden Hardware/Software-System. Dieses Hardware/Software-System nutzt zum einen die Datenparallelität durch die Bereitstellung mehrerer Prozessor-Elemente und zum anderen die funktionale Parallelität durch die Realisierung eines Multiprozessor-Systems aus. Die Besonderheit des verwendeten Neuronenmodells ist, dass die Detektion von Bildmerkmalen, wie Flecken oder Kanten, auf der Basis von adaptiven Synapsengewichten erfolgt. Durch die kontinuierlich durchzuführende Adaption der Synapsengewichte, verschärfen sich die Flaschenhals-Probleme der limitierten Speicherbandbreite und der durchzuführenden Rechenoperationen während der Simulation. Diese Flaschenhals-Probleme werden durch eine verteilte Speicherarchitektur und eine parallele Implementierung des notwendigen numerischen Integrationsverfahrens für die Neuronenzustandsberechnung gelöst.

In dieser Arbeit wird gezeigt, dass für die Anforderungen an eine digitale Emulation von pulscodierten neuronalen Netzen programmierbare applikationsspezifische Schaltkreise in Form von FPGAs am besten geeignet sind. Die Vorteile heutiger FPGA-Technologien bestehen neben der Kostenwirksamkeit für Prototypen-Entwürfe und der Flexibilität aufgrund der Rekonfigurierbarkeit vor allem in der hohen Integrationsdichte, der ausreichend hohen Performanz bezüglich der realisierbaren Operationsfrequenzen und der hinreichend nutzbaren Speicherbandbreite. Im Rahmen dieser Arbeit wurde eine Prototypen-Platine für die digitale Emulationsplattform entworfen, die auf drei FPGAs der Typen Virtex-II und Virtex-II Pro von Xilinx basiert. Die für die Emulation parallel nutzbaren Speicher auf der Prototypen-Platine bestehen aus sechs SDRAM-Bausteinen und zwölf SRAM-Bausteinen, die insgesamt eine Speicherbandbreite von 2,6 GB/s zur Verfügung stellen.

Die implementierte Architektur der Emulationsplattform ermöglicht bei einer Taktfrequenz von 50 MHz einen maximalen Beschleunigungsfaktor von 30 für eine spärliche Vernetzungsstruktur und einen maximalen Beschleunigungsfaktor von 8 für eine dichte Vernetzungsstruktur im Vergleich zu durchgeführten Simulationen auf einem Einprozessor-System in Form eines Linux-PCs (2,4 GHz Pentium-IV-Prozessor von Intel mit 1 GB RAM Arbeitsspeicher).

Schlüsselwörter: Digitale Emulation, Pulscodierte Neuronale Netze, Adaptive Synapsengewichte, Simulationsbeschleunigung, Hardware/Software-System, FPGA

Abstract

The simulation of pulse-coded neural networks for the evaluation of biology-oriented image processing performed on general-purpose computers, e. g. PCs or workstations, is still very time-consuming and inefficient. According to observations in the biology pulse-coded neural networks process information with pulses and are characterised by the ability to adapt themselves to outside stimuli during processing. This ability for adaptation is known as learning process. For the purpose of learning it is required that weights of network connections (synapses) between network elements (neurons) are able to alter dynamically. The main bottle-neck during the simulation is the sequential access to the memory of dynamic network parameters (membrane potential and adative synaptic weights). This memory access is necessary for the computation and update of the neuron states.

In this thesis a novel architectural concept and the development of a digital emulation platform are introduced that is intended for the simulation acceleration of pulse-coded neural networks. This emulation platform represents the first experimental platform of a pulse-coded neuron model with adaptive synaptic weights for the integral image processing. The platform is distinguished by the evaluation of various network dynamics and by the provision of high flexibility for the implementation. The architecture of the emulation platform realises an efficient communication between the parallel computation of network topologies and neuron states and is based on an unreported hardware/software-system. This hardware/software-system exploits on the one hand the data parallelism by providing multiple processing elements and on the other hand the functional parallelism by the realisation of a multi-processor system. The specific characteristic of the neuron model is that the detection of image features, e. g. spots and edges, is performed on the basis of adaptive synaptic weights. Because of the continuous adaptation of synaptic weights that has to be carried out during the simulation the bottle-necks of limited memory bandwidth and of necessary calculation steps are giving rise to a more negative impact on the simulation performance. These bottle-neck problems are solved by a distributed memory architecture and a parallel implementation of the numerical integration method required for the computation of neuron states.

The work in this thesis reveals that the requirements on a digital emulation platform for pulse-coded neural networks are best met with programmable application specific integrated circuits in the form of FPGAs. The advantages of FPGA technolgies are cost effectiveness for prototype designs and flexibility due to reconfigurability. However, the main benefits offered by today's FPGA technologies are the capability for very large scale integration circuits, adequate operation speed performance and the realisation of sufficiently usable memory bandwidth. In the context of this thesis a prototyping printed circuit board for the emualtion platform was manufactured that incorporates three FPGA devices of the type Virtex-II and Virtex-II Pro from Xilinx. The parallel accessible memory for the digital emulation on the printed circuit board consists of six SDRAM devices and twelve SRAM devices that provide an overall memory bandwidth of 2.6 GB/s.

The implemented architecture of the emulation platform operates at a frequency of 50 MHz. It is evaluated that the emulation platform achieves a maximum acceleration factor of 30 for a sparsely connected network and a maximum acceleration factor of 8 for a dense connected network compared to performed software simulations on a stand-alone Linux-PC (2.4 GHz Pentium-IV-Processor from Intel and 1 GB RAM of main memory).

Keywords: Digital Emulation, Pulse-Coded Neural Networks, Adaptive Synaptic Weights, Simulation Acceleration, Hardware/Software-System, FPGA

Glossar

A

ALU	Arithmetic Logic Unit
AHB	Advanced High-Performance Bus
ASIC	Application Specific Integrated Circuit

B

BRAM	Block Random Access Memory
BSI	Bulirsch-Stoer Integrator

C

CLB	Configurable Logic Block
CPS	Connections Per Second
CPU	Central Processing Unit
CUPS	Connection Updates Per Second

D

DEL	Dynamic Event List
DIMM	Dual Inline Memory Module
DSP	Digital Signal Processor

E

ETF	Excitation Tag Field

F

FEL	Fire Event List
FTF	Fire Tag Field
FPGA	Field Programmable Gate Array

G

GUI	Graphical User Interface

H

HW	Hardware

I

IC	Integrated Circuit
IO	Input/Output

L

LC	Logic Cell
LUT	Look-Up-Table

M

MIMD	Multiple Instruction Multiple Data
MMID	Modified Midpoint (Integration)

N

NIB	Neuron Information Block
NSC	Neuron State Computation
NTC	Network Topology Computation

P

PC	Personal Computer
PCB	Printed Circuit Board
PCNN	Pulse Coded/Coupled Neural Network
PE	Processing Element
PLB	Processor Local Bus
PPC	PowerPC
PROM	Programmable Read Only Memory
PSP	Postsynaptic Potential
PZEXTR	Polynomial Extrapolation

R

RF	Receptive Field
RISC	Reduced Instruction Set Computer

S

SDRAM	Static Dynamic Random Access Memory
SEE	Spiking Neural Network Emulation Engine
SIMD	Single Instruction Multiple Data
SRAM	Static Random Access Memory
SW	Software

T

TCU	Tag Computation Unit
TPC	Topology Cache
TPM	Topology Memory
TVU	Topology Vector Unit

U

UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

Inhaltsverzeichnis

Kurzfassung i

Abstract iii

Glossar v

KAPITEL 1 Einleitung **1**

1.1 Motivation. 1
1.2 Zielsetzung. 4
1.3 Forschungsbeitrag 5
1.4 Kapitelübersicht 6

KAPITEL 2 Bildverarbeitung mit pulscodierten Neuronen **7**

2.1 Das biologische Neuron. 7
2.1.1 Hodgkin-Huxley Modell 9
2.1.2 Elemente neuronaler Dynamik. 11
2.1.2.1 Synapsen 11
2.1.2.2 Postsynaptische Potentiale. 12
2.1.2.3 Ionenkanäle 13
2.1.3 Neuronale Pulscodierung. 14
2.1.3.1 Ratencodierung 14
2.1.3.2 Pulscodierung. 15
2.2 Pulscodierte Neuronenmodelle 15
2.2.1 Integrate-and-Fire Modell. 16
2.2.2 Spike-Response Modell 17
2.2.3 Marburger Modell. 18
2.2.4 Neuronenmodell mit adaptiven Synapsengewichten 19
2.2.4.1 Puls-Korrelations-Detektor 20
2.2.4.2 Puls-Dekorrelations-Detektor 21
2.3 Das menschliche visuelle System 22
2.3.1 Neuronale Pulsverarbeitung mit rezeptiven Feldern 25
2.3.1.1 Retinale rezeptive Felder 25
2.3.1.2 Cortikale rezeptive Felder. 26
2.4 Biologisch motivierte Bildverarbeitung. 27
2.4.1 Modelle zur visuellen Objektwahrnehmung 30

KAPITEL 3 Simulation pulscodierter neuronaler Netze **33**

3.1 Grundlegender Simulationsablauf 33
3.1.1 Simulationsverfahren 34
3.2 Flaschenhals-Probleme bei der Simulation 36
3.2.1 Rechenoperationen 36
3.2.2 Kommunikationsaufwand. 37
3.2.3 Lastverteilung. 37
3.2.4 Speicherkapazität. 38
3.2.5 Speicherbandbreite 38
3.3 Konzepte zur Simulationsbeschleunigung. 39
3.3.1 Listenverwaltung 39
3.3.2 Parallelisierung. 41

3.3.3 Reduzierung der notwendigen Speicherbandbreite . . . 42
3.3.4 Nutzung von Cache-Speichern. . . . 42
3.4 Digitale Simulationsplattformen . . . 44
3.4.1 Software Simulation . . . 46
3.4.2 DSP basierte Emulation . . . 47
3.4.3 Voll-Kunden-Entwurf basierte Emulation . . . 48
3.4.3.1 Neuronen-Feld . . . 48
3.4.3.2 Systolische Zellenfelder . . . 48
3.4.3.3 Sendebus . . . 49
3.4.3.4 Coprozessor . . . 49
3.4.4 Halb-Kunden-Entwurf basierte Emulation . . . 50
3.4.4.1 NESPINN . . . 50
3.4.4.2 MASPINN . . . 51
3.4.4.3 SP²INN . . . 52
3.4.5 FPGA basierte Emulation . . . 53
3.4.5.1 SPIKE128K . . . 54
3.4.5.2 RAPTOR2000 . . . 55
3.4.5.3 BEE . . . 56
3.4.6 Effizienzbetrachtung digitaler Simulationsplattformen . . . 57

KAPITEL 4 SEE: eine FPGA basierte Emulationsplattform . . . 61

4.1 Ansatz für eine FPGA basierte Emulationsplattform . . . 61
4.1.1 Vorteile heutiger FPGA-Technologien . . . 62
4.1.2 Genauigkeitsanforderung an Festkomma-Arithmetik . . . 64
4.1.3 Eigenschaften eines effizienten Emulationsverfahrens . . . 67
4.1.4 Architekturkonzept der Emulationsplattform . . . 67
4.2 Maßnahmen zur Beschleunigung der Emulation . . . 69
4.2.1 Klassifizierung von pulscodierten Neuronen . . . 69
4.2.2 Parallelverarbeitung des Pulsereignis-basierten Emulationsablaufs . . . 70
4.2.3 Optimale Datenkanalauslastung einer verteilten Speicherarchitektur . . . 71
4.2.4 Autarkie der Emulationsplattform . . . 72
4.2.5 Parallelisierung des notwendigen Integrationsverfahrens . . . 72
4.3 Numerisches Integrationsverfahren nach Bulirsch-Stoer . . . 72
4.3.1 Modified Midpoint Integration . . . 74
4.3.2 Polynomextrapolation . . . 78
4.3.3 Nullstellensuchverfahren nach Newton . . . 81
4.3.4 Einsatz bei der Neuronenzustandsberechnung . . . 83
4.4 Netzwerkarchitekturen für das Neuronenmodell mit adaptiven Synapsengewichten . . . 85
4.4.1 Kaskaden-Architektur . . . 85
4.4.2 Gabor-Architektur . . . 86

KAPITEL 5 Implementierung von SEE . . . 91

5.1 Beschreibung des Hardware-Entwurfs. . . . 91
5.1.1 Emulationsablaufkontrolle . . . 91
5.1.1.1 Verwaltung der Listen pulscodierter Neuronen . . . 94
5.1.1.2 Adressraum der Emulationsplattform . . . 95
5.1.2 Neuronenzustandsberechnung . . . 96
5.1.2.1 Bulirsch-Stoer Hardware Integrator . . . 98
5.1.2.2 Prozessor-Elemente für unterschiedliche Vernetzungsdichten . . . 100
5.1.3 Netzwerktopologieberechnung . . . 102
5.1.3.1 Berechnung von Topologie-Vektoren . . . 103

5.1.3.2 Berechnung von postsynaptischen Neuronen . . . 106
5.1.4 Listenzugriffe von erregten pulscodierten Neuronen . . . 108
5.1.5 Hardware-Ressourcen . . . 110
5.2 Beschreibung des Software-Entwurfs . . . 112
5.2.1 Steuerungsprogramme für den Emulationsablauf . . . 112
5.2.2 Software-Ressourcen . . . 114
5.3 Darstellung des Hardware/Software-Entwurfsfluss . . . 114
5.4 Realisierung der Prototypen-Platine . . . 117
5.4.1 Verlustleistungsabschätzung . . . 118
5.4.2 Eigenschaften . . . 120
5.5 Durchführung der Emulation von pulscodierten neuronalen Netzen . . . 121
5.5.1 Netzwerkbeschreibung . . . 122
5.5.2 Netzwerkkonfiguration . . . 123
5.5.3 Inbetriebnahme der Emulationsplattform . . . 125
5.5.4 Visualisierung von Pulsereignissen . . . 125

KAPITEL 6 Leistungsbewertung von SEE . . . 127

6.1 Simulationen auf einem Einprozessor-System . . . 127
6.1.1 Emulationszeitabschätzungen von SEE . . . 132
6.1.2 Funktionalitätsüberprüfung von SEE . . . 135
6.2 Leistungsgrenzen von SEE . . . 135
6.3 Leistungsbewertung anderer Emulationsplattformen . . . 137
6.3.1 Halb-Kunden-Entwurf basierte Plattformen . . . 137
6.3.2 FPGA basierte Plattformen . . . 138
6.4 Bewertung der Implementierung . . . 140
6.4.1 Effizienzvergleich . . . 140
6.4.2 Flexibilitätsbereitstellung . . . 142
6.5 Evaluation des Neuronenmodells mit adaptiven Synapsengewichten . . . 143

KAPITEL 7 Zusammenfassung . . . 149

7.1 Ergebnisse . . . 149
7.2 Ausblick . . . 154

Literaturverzeichnis . . . 157

Anhang . . . 167

Danksagung . . . 187

Lebenslauf . . . 189

KAPITEL 1

Einleitung

Prozessorarchitekturen

Heutige komplexe Digitalsysteme sind durch 3 praktisch relevante Prozessorarchitekturen gekennzeichnet, deren Klassifizierung auf [Flyn66] zurückgeht. SISD-Architekturen (*Single Instruction Single Data*) sind Einprozessor-Systeme. Man differenziert hierbei zwischen der von-Neumann-Architektur, bei der in einem zentralen Speicher Daten und Befehle abgelegt sind, und der leistungsfähigeren Harvard-Architektur, die für Daten und Befehle jeweils einen separaten Speicher vorsieht und somit den parallelen Zugriff von Daten und Befehlen ermöglicht. Bei den SIMD-Architekturen (*Single Instruction Multiple Data*), die mit einem Befehl mehrere Datensätze bearbeiten, unterscheidet man zwischen der eindimensionalen Anordnung (Pipeline-Prozessor) und der feldförmigen Anordnung (systolisches Array) von Prozessor-Elementen (PE = *Processing Element*). Bei der feldförmigen Anordnung sind die PEs jeweils mit ihren unmittelbaren Nachbarn verbunden. MIMD-Architekturen (*Multiple Instruction Multiple Data*) bezeichnen Multiprozessor-Systeme, die zum einen mit einem gemeinsamen Speicher (*Shared Memory Architecture*) oder mit einem verteiltem Speicher (*Distributed Memory Architecture*) ausgestattet sind. In diesen Systemen arbeiten die Prozessoren parallel unabhängig voneinander. Sie unterscheiden sich somit zum einen von einem Mehrprozessor-System, wo eine zentrale Recheneinheit von einer Anzahl von Coprozessoren zur Durchführung von speziellen Aufgaben unterstützt wird, und zum anderen von einem systolischen Array, bei dem die Prozessoren nicht autonom agierende arithmetische Funktionseinheiten darstellen [Gilo93].

1.1 Motivation

Heterogene SIMD/MIMD Architektur

Die Wahl der Prozessorarchitektur für ein Digitalsystem hängt vom Anwendungsgebiet und den dazugehörigen Algorithmen ab. Algorithmen in der Bildverarbeitung beispielsweise sind durch ihre Datenintensität und ihre 2 unterschiedlichen Arten von Parallelität charakterisiert. Das Komplexitätsspektrum der Algorithmen reicht von pixel-basierten Low-Level Operationen bis hin zu objektbasierten High-Level Operationen. Bei Low-Level Algorithmen in der Bildvorverarbeitung, wie der Filterung zur Glättung oder der Kontrastverbesserung, liegt aufgrund der Lokalität von notwendigen Operanden eine Datenparallelität vor. Folglich eignen sich solche Algorithmen für die Abbildung auf mehrere arithmetische Funktionseinheiten (SIMD-Architektur), welche diese Lokalität optimal ausnutzen können. Bei High-Level Algorithmen, wie der Erkennung und Verfolgung von Objekten oder der Interpretation von Bildern und Szenen, liegt eine funktionale Parallelität vor. Dadurch ist eine Abbildung auf unabhängige Prozessoren vorteilhaft (MIMD-Architektur), die verschiedene Teilaufgaben (*Multitasking*) übernehmen [Visi01].

Leistungskluft

Datenintensive Algorithmen benötigen für die meisten Befehle oder Operationen eine Vielzahl von Daten, was zu einem ständigen Zugriff auf den Datenspeicher führt. Die Ausführung datenintensiver Algorithmen auf einem Einprozessor-System wird aufgrund der steigenden Leistungskluft (*Performance Gap*) zwischen Prozessor und Speicher zunehmend ineffizient. Nach [Henn03] hat sich bis zum Jahr 1986 die Prozessor-Leistung jährlich um 35 % erhöht, und seit 1987 steigt diese jedes Jahr um 55 % an. Im Vergleich dazu erhöht sich die Leistung von Speichern jedes Jahr kontinuierlich nur um 7 %. Diese Leistungskluft ist in Abb. 1-1 dargestellt.

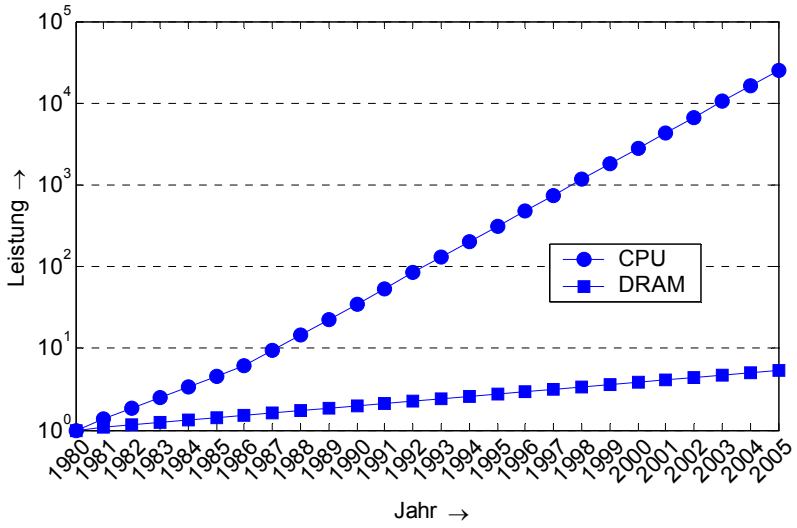


Abb. 1-1: Leistungskluft zwischen Prozessor und Speicher (basierend auf [Henn03])

Die benötigten Daten können nicht vollständig in Speichern mit kurzen Latenzzeiten, wie Registern oder Cache-Speichern, abgelegt werden, sondern müssen von Speichern mit längeren Latenzzeiten, wie Arbeitsspeichern in Form von DRAM-Speichern (DRAM = *Dynamic Random Access Memory*) oder Massenspeichern, gelesen werden. Prozessoren, die für die digitale Signalverarbeitung optimiert sind (DSP = *Digital Signal Processor*), versuchen dieses Problem durch einen großen Cache-Speicher etwas zu entschärfen. Aufgrund der dargestellten Leistungskluft bringen die hohen Operationsfrequenzen von Prozessoren im Falle von datenintensiven Anwendungen aber keine wesentlichen Vorteile. Die Ausführungszeit einer zentralen Recheneinheit (CPU = *Central Processing Unit*) in einem Einprozessor-System ist durch die Summe der Takte pro Befehl (CPI = *Cycles Per Instruction*) für die eigentliche Befehlsausführung (*Execution*) und für die eventuelle Speicherverzögerung (*Memory Stall*) bestimmt und lässt sich wie folgt formulieren [Henn03],

$$T_{CPU} = M \cdot (CPI_{Execution} + CPI_{MemoryStall}) \cdot T_{CLK} \qquad \textbf{(1-1)}$$

wobei *M* die Anzahl der Befehle und T_{CLK} die Taktperiode darstellen. Die Taktanzahl für die Speicherverzögerung ist in Gl. 1-2 dargestellt.

$$CPI_{MemoryStall} = MissRate \cdot MissPenalty \qquad \textbf{(1-2)}$$

Die Anzahl der Takte für eine Speicherverzögerung setzt sich aus dem Produkt von Fehltreffer im Cache (*Miss Rate*) und der notwendigen Takte, diesen Fehltreffer zu kompensieren (*Miss Penalty*), zusammen. Für komplexe und datenintensive Algorithmen in der heutigen digitalen Signalverarbeitung ist die Fehltrefferrate im Cache sehr hoch. Die Anzahl der Takte für die Speicherverzögerung ($CPI_{MemoryStall}$) erweist sich als der dominante Faktor für die Gesamtausführungszeit der CPU und somit als Flaschenhals der Anwendung (siehe Gl. 1-1).

Parallelisierbarkeit

Eine Beschleunigung eines Algorithmus lässt sich durch Parallelisierung erzielen, indem man mehrere PEs vorsieht, die Teile des Algorithmus parallel verarbeiten. Dabei skaliert der Beschleunigungsfaktor *F* nicht linear mit der Anzahl der PEs, sondern ist abhängig von der Parallelisierbarkeit des Algorithmus [Amda67]. Der Wirkungsgrad η einer Beschleunigung ist somit von der Anzahl *N* der PEs und des Parallelitätsgrades ρ des zu beschleunigenden Algorithmus abhängig (siehe Gl. 1-3).

$$F = \left[(1-\rho) + \frac{\rho}{N}\right]^{-1} = N \cdot \eta$$
$$\eta = \frac{1}{N \cdot (1-\rho) + \rho} \qquad \textbf{(1-3)}$$

In Abb. 1-2 ist der Wirkungsgrad in Abhängigkeit des Parallelitätsgrades für unterschiedliche Anzahlen von PEs aufgetragen. Man sieht, dass eine hohe Anzahl von PEs (N = 100) nur sinnvoll ist, wenn der Algorithmus einen hohen Parallelitätsgrad aufweist (> 0,9), da sich erst dann der Wirkungsgrad sichtlich erhöht.

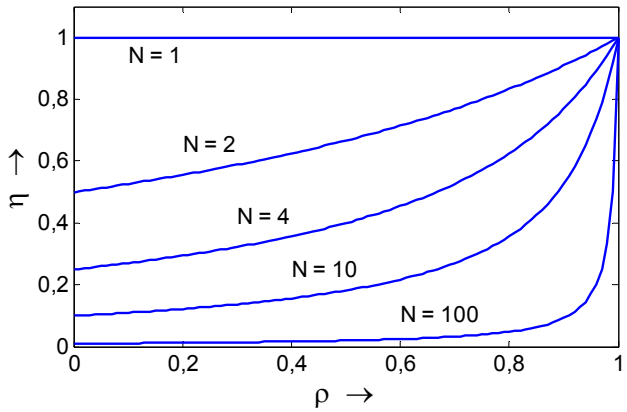


Abb. 1-2: Das Amdahlsche Gesetz: Wirkungsgrad η als Funktion des Parallelitätsgrades ρ mit unterschiedlichen Anzahlen *N* von Prozessor-Elementen (basierend auf [Gilo93])

PCNN Simulation

Die Simulation von pulscodierten neuronalen Netzen, kurz PCNN (*Pulse Coded Neural Network*) Simulation, wird primär aus 2 Gründen durchgeführt. Zum einen möchte man die Pulsverarbeitung der 100 Milliarden Neuronen (10^{11}) und das Adaptionsverhalten der 100 Billionen Synapsen (10^{14}) im menschlichen Gehirn reproduzieren und verstehen [Kand00], und zum anderen möchte man die dadurch erlangten Erkenntnisse verwenden, ausgefeilte technische Systeme für unterschiedliche Anwendungsgebiete, wie Bildverarbeitung, Spracherkennung [Merc02], Robotersteuerung [Rogg03] oder Regelungsaufgaben [Mala00], zu realisieren. In dieser Arbeit wird ausschließlich der Einsatz von PCNNs in der Bildverarbeitung betrachtet.

PCNN Größe

Die PCNN Simulation, die auf immer leistungsfähigeren Einprozessor-Systemen durchgeführt werden kann, gewinnt zunehmend an Bedeutung für die Modellierung von einzelnen Gehirnarealen, die eine PCNN Größe von 10^4-10^5 Netzwerkelementen (Neuronen) und von 10^7-10^8 Netzwerkverbindungen (Synapsen) darstellen [Reut02]. Bei der PCNN Simulation ergibt sich wie in der Bildverarbeitung eine hohe Datenintensität bei zunehmender Größe und Verbindungsdichte des Netzwerks. Aufgrund der Leistungskluft zwischen Prozessor und Speicher wird die Simulation auf einem Einprozessor-System zunehmend ineffizient und sehr zeitaufwendig. Für eine effiziente Simulationsbeschleunigung ist eine SIMD-Architektur mit einer hohen Anzahl an PEs nicht ausreichend. Hier ist zusätzlich eine Erhöhung des Parallelitätsgrades des PCNN Algorithmus notwendig, um den Wirkungsgrad η zu erhöhen. Dies kann erreicht werden, indem man versucht, den sequenziellen Anteil während der Simulation soweit wie möglich zu reduzieren. Der größte sequentielle Teil bei der PCNN Simulation ist der Datentransfer von den PEs zum Parameter- bzw. Gewichtsspeicher [Reyn03]. Durch eine erhöhte Speicherbandbreite (Datentransfer- bzw. Datenübertragungsrate zu Speicherbausteinen) und eine verteilte Speicherarchitektur, wie bei einer MIMD-Architektur, ist es möglich, mehrere Daten- bzw. Speicherkanäle bereitzustellen. Die Speicherbandbreite beschreibt die Anzahl der Bytes (1 Byte = 8 Bit), die in einer Zeiteinheit transferiert werden können, und wird als Produkt von Taktfrequenz und Bitbreite des Speicherbusses in B/s (Bytes pro Sekunde) angegeben. Bei der Bereitstellung von mehreren Speicherkanälen kann der Datentransfer von den PEs zum Gewichtsspeicher auf diesen Kanälen parallel erfolgen. Dadurch wird der Parallelitätsgrad des PCNN Algorithmus weiter erhöht.

PCNN Emulation

Die Herausforderung für ein effizientes digitales Simulationsbeschleuniger-System besteht darin, die Simulation von pulscodierten neuronalen Netzen auf Einprozessor-Systemen exakt nachzubilden (PCNN Emulation) und durch eine geeignete Balance zwischen SIMD- und MIMD-Architektur eine ausreichende Beschleunigung zu erzielen. Weiterhin muss ein Simulationsbeschleuniger-System über eine ausreichende Speicherkapa-

zität verfügen, damit die Emulation von einzelnen Gehirnarealen möglich ist. Aus diesem Grund muss mindestens eine PCNN Größe von 10^5 Neuronen und 10^8 Synapsen berücksichtigt werden können [Reut02]. Anhand einer PCNN Emulation ist es somit möglich, eine Vielzahl von Experimenten durchzuführen und das Verhalten des Neuronenmodells in Verbindung mit einer komplexeren Netzwerkarchitektur zu evaluieren.

1.2 Zielsetzung

Funktionale Emulation

In dieser Arbeit wird explizit zwischen der PCNN Simulation und der PCNN Emulation unterschieden. Die Simulation versucht anhand eines Neuronenmodells das Verhalten von pulsenden Neuronen zu imitieren (ital. simulare = “so tun als ob”) und läuft meist auf einem Einprozessor-System, wie PC oder Workstation, anhand eines SW-Programms (SW = *Software*) ab. Die Emulation bildet diese Simulation auf einer digitalen Emulationsplattform oder anderen optimierten HW-Architekturen (HW = *Hardware*) nach (ital. emulare = “es gleichtun”). Da eine laufzeiteffiziente PCNN Simulation auf Einprozessor-Systemen nicht durchführbar ist, ist eine effiziente digitale Emulationsplattform notwendig. Die Emulationszeit soll dabei mindestens eine Größenordnung kleiner sein als die Simulationszeit. Dabei ist keine Emulation in Echtzeit notwendig, sondern eine funktionale Emulation der PCNN Simulation. Als Netzwerkzeit wird die Zeit betrachtet, in der das pulscodierte neuronale Netz aufgrund der eingestellten Modellparameter operiert. Eine Echtzeit-Emulation ist gegeben, wenn die Emulationszeit die Netzwerkzeit nicht überschreitet. Bei entsprechend großen Netzwerken oder komplexen Neuronenmodellen ist eine Echtzeit-Emulation mit bestehenden Architekturen digitaler Simulationsbeschleuniger-Systeme nicht möglich.

FPGA-Entwurf

Die zu entwerfende digitale Emulationsplattform soll als sog. “rapider” Prototyp (*Rapid Prototyp*) in Form einer applikationsspezifischen integrierten Schaltung (*ASIC = Application Specific Integrated Circuit*) realisiert werden. Ein “Rapid Prototyp” hat die Eigenschaften, dass sowohl der Entwurf als auch die implementierte Funktion zeitlich schnell erfolgen [Krup00] und Änderungen bezüglich der implementierten Funktion kontinuierlich angepasst werden können. Bei ASICs unterscheidet man 3 Kategorien [Smit97]: Voll-Kunden (*Full Custom*) ASICs, Halb-Kunden (*Semi Custom*) ASICs, die auf Standardzellen oder Gate Arrays basieren, und programmierbare ASICs, wie PLDs (*Programmable Logic Devices*) und FPGAs (*Field Programmable Gate Arrays*).

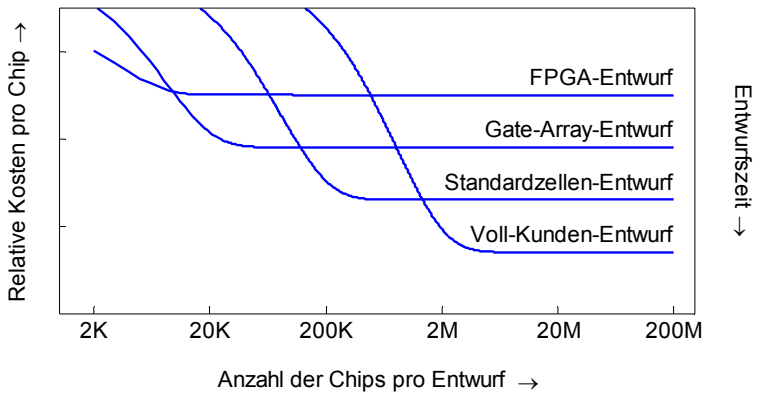


Abb. 1-3: Kostenvergleich unterschiedlicher Entwurfsformen integrierter Digitalschaltungen (basierend auf [Veen00])

Aufgrund des Prototypencharakters (sehr geringe Stückzahl) und der erforderlichen kurzen Entwurfszeit, die notwendig ist, um auf Änderungen des Neuronenmodells reagieren und neue Netzwerkarchitekturen berücksichtigen zu können, ist ein FPGA-Entwurf für die digitale Emulationsplattform im Vergleich zu anderen Entwurfsformen am besten geeignet (siehe Abb. 1-3). Heutige FPGAs sind durch 3 Eigenschaften charakterisiert: Kostenwirksamkeit, hohe Integrationsdichte und ausreichend hohe Performanz bezüglich

der Operationsfrequenzen. Hinzukommt, dass heutige FPGAs in Gehäuseformen verfügbar sind, die mit mehr als 1000 Signalanschlüssen (*IO Pins*) ausgestattet sind. Solche Gehäuseformen sind für Voll- bzw. Halb-Kunden ASICs bei geringen Stückzahlen unwirtschaftlich, ermöglichen aber eine hohe Speicherbandbreite, die für die Simulationsbeschleunigung von pulscodierten neuronalen Netzen notwendig ist.

Evaluation und Flexibilität

Die digitale Emulationsplattform soll primär als Experimentierplattform für große PCNNs dienen. Hierbei stehen die Evaluation der Netzwerkdynamik und die Bereitstellung einer hohen Flexibilität bei der Implementierung im Vordergrund. Die Evaluation der Netzwerkdynamik dient zur Bestimmung und Optimierung von Modellparametern in einem vielschichtigen PCNN. Hierfür ist sowohl der unmittelbare Zugriff auf beliebige Netzwerk- und Neuronenzustände als auch die Protokollierung sämtlicher Pulsereignisse für eine anschließende Visualisierung notwendig. Hiermit kann das Verhalten unterschiedlicher Neuronenmodellvarianten, wie z. B. unterschiedliche Adaptionsregeln für Synapsengewichte, bestimmt werden. Die Flexibilität der Emulationsplattform wird dadurch gewährleistet, dass durch unterschiedliche Netzwerkbeschreibungen anhand von Software-Programmen, die PCNN Emulation für unterschiedliche Netzwerkarchitekturen konfiguriert werden kann. Zusätzlich bietet die Rekonfigurierbarkeit von FPGAs flexibel nutzbare Implementierungsvarianten für unterschiedliche Modellfunktionen und eine einstellbare Genauigkeit für arithmetische Operationen.

1.3 Forschungsbeitrag

Digitale Emulationsplattform

In dieser Arbeit wird die Konzeption und Implementierung einer Hardware/Software-Emulationsplattform für große pulscodierte neuronale Netze vorgestellt. Die Architektur der Emulationsplattform ist durch eine heterogene SIMD/MIMD-Architektur charakterisiert. Die SIMD-Architektur wird durch die Bereitstellung mehrerer PEs realisiert, deren Anzahl von den verfügbaren Datenkanälen zum Gewichtsspeicher abhängig ist. Die PEs realisieren ein numerisches Integrationsverfahren [Stoe00], welches im Rahmen eines Simulationsbeschleuniger-Systems noch nicht in Hardware realisiert wurde. Die MIMD-Architektur wird durch eingebettete CPUs innerhalb eines FPGAs ermöglicht. Dadurch können voneinander unabhängige Aufgaben, wie die Emulationsablaufkontrolle auf der Plattform und die Kommunikation mit einem Host-Rechner, parallel durchgeführt werden. Weiterhin werden 2 Ereignislisten und 2 Markierungsspeicher verwendet, die pulsende Neuronen und erregte Neuronen verwalten. Der ausreichend dimensionierte Parameter- bzw. Gewichtsspeicher ist mit 3 parallel nutzbaren Datenkanälen ausgelegt. Die notwendige Kommunikation der getrennten Netzwerktopologie- und Neuronenzustandsberechnungen wird anhand von Topologie-Vektoren realisiert, die die Pulsaktivität von Puls-sendenden Neuronen widerspiegeln. Diese Pulsaktivität ist für die Zustandsberechnungen von Puls-empfangenden Neuronen notwendig.

Pulscodiertes Neuronenmodell

In dieser Arbeit wird ein pulscodiertes Neuronenmodell mit adaptiven Synapsengewichten betrachtet [Heit02], das bezüglich der Leistungsfähigkeit für den Einsatz in der Bildverarbeitung evaluiert werden soll. Für dieses pulscodierte Neuronenmodell existiert noch keine geeignete digitale Emulationsplattform. Anhand von durchgeführten Simulationen auf der Emulationsplattform wird versucht, einige Grenzen der Leistungsfähigkeit des Neuronenmodells mit adaptiven Synapsengewichten aufzuzeigen. Im Vordergrund stehen dabei aber die Leistungsbewertung der konzipierten Emulationsplattform und die Darlegung, dass für einen ausreichenden Beschleunigungsfaktor die Erhöhung des Parallelitätsgrads einer PCNN Simulation unabdingbar ist.

Adaptive Synapsengewichte

Bei vielen technischen Anwendungen, wo neuronale Netze zum Einsatz kommen, unterscheidet man zwischen einer Lern- und einer Verarbeitungsphase. In der Lernphase werden die Synapsengewichte adaptiert, und in der Verarbeitungsphase werden die in der Lernphase eingestellten Gewichtswerte nicht mehr verändert. Während der Verarbeitung hat man somit ein statisches Verhalten der Synapsen. Die Fähigkeit von pulscodierten neuronalen Netzen, sich auch während der Verarbeitung äußeren Reizen anzupassen,

entspricht den Beobachtungen in der Biologie. Die Synapsen haben in diesem Fall während der Verarbeitung ein dynamisches Verhalten und müssen ständig adaptiert werden. Dies hat zur Folge, dass sich der Flaschenhals in Form des Datentransfers zwischen PEs und Gewichtsspeichern bei einer konstanten Speicherbandbreite weiter verschärft. Eine ausreichende Beschleunigung unter Berücksichtigung adaptiver Synapsengewichte wird durch eine hohe Speicherbandbreite bei der konzipierten Emulationsplattform sichergestellt.

Im Rahmen dieser Arbeit sind als Forschungsbeitrag 3 internationale Publikationen [Hell04a] [Hell04b] [Hell05c] (siehe auch Literaturverzeichnis Seite 159) und 5 Diplomarbeiten [Geik07] [Grie07] [Mahr07] [Rafa06] [Sann03] entstanden.

1.4 Kapitelübersicht

Das Kapitel 2 dieser Arbeit beschreibt den aktuellen Stand der Technik in der Bildverarbeitung anhand von pulscodierten neuronalen Netzen. Dabei wird ausgehend vom biologischen Vorbild zuerst ein Überblick über pulscodierte Neuronenmodelle einschließlich des zu evaluierenden Neuronenmodells mit adaptiven Synapsengewichten gegeben. Anschließend wird von der Betrachtung einzelner Neuronenmodelle auf die Funktionsweise des menschlichen visuellen Systems übergeleitet, um die vielversprechenden Vorteile einer biologisch motivierten Bildverarbeitung darzustellen. In Kapitel 3 werden die Flaschenhals-Probleme der digitalen Simulation von pulscodierten neuronalen Netzen diskutiert und bekannte Konzepte skizziert, die der Simulationsbeschleunigung dienen. Darauf folgt die Vorstellung von relevanten Simulationsplattformen und aktuellen Forschungsprojekten, die sich für die Simulationsbeschleunigung von pulscodierten neuronalen Netzen eignen. Die im Rahmen dieser Arbeit konzipierte digitale Emulationsplattform, die abgekürzt mit SEE (*Spiking Neural Network Emulation Engine*) bezeichnet wird, stellt ein PCNN Simulationsbeschleuniger-System dar und wird in Kapitel 4 präsentiert. Hierbei werden die Besonderheiten der Netzwerkarchitekturen des Neuronenmodells mit adaptiven Synapsengewichten herausgestellt, für die SEE primär beabsichtigt ist. Das Kapitel 5 beschreibt die Realisierung von SEE, welche den Entwurf von integrierten Hardware-Modulen, Software-Programmen und der Prototypen-Platine umfasst. Dabei wird der Ablauf des Hardware/Software-Co-Entwurfs von SEE und der Ablauf für die Durchführung einer Netzwerksimulation dargestellt. Im nachfolgenden Kapitel 6 erfolgt eine Leistungsbewertung von SEE anhand von durchgeführten Netzwerksimulationen. Als Vergleich dienen hierfür durchgeführte Netzwerksimulationen auf Einprozessor-Systemen und Simulationszeitabschätzungen anderer bereits existierender Simulationsbeschleuniger-Systeme. Die Arbeit schließt in Kapitel 7 mit einer Zusammenfassung der Ergebnisse und einem Ausblick auf notwendige Forschungsarbeiten in der Zukunft.

Abkürzungen, die in dieser Arbeit häufig verwendet werden, sind zum Nachschlagen alphabetisch im Glossar geordnet, welches sich vor dem Inhaltsverzeichnis auf der Seite v befindet.

KAPITEL

2

Bildverarbeitung mit pulscodierten Neuronen

In diesem Kapitel werden ausgehend vom dynamischen Verhalten biologischer Neuronen 4 pulscodierte Neuronenmodelle vorgestellt. Die beiden bekanntesten pulscodierten Neuronenmodelle sind das Integrate-and-Fire [Stei67] und das Spike-Response Modell [Gers98]. Die weiteren beiden Modelle, das Marburger Modell [Eckh90] und das in dieser Arbeit verwendete Neuronenmodell mit adaptiven Synapsengewichten [Heit02], erheben den Anspruch, speziell für die Bildverarbeitung geeignet zu sein. Nach der Betrachtung dieser Neuronenmodelle werden die Verarbeitungsprozesse im menschlichen visuellen System vorgestellt. Dabei wird die Pulsverarbeitung von Neuronen, die sich aufgrund von unterschiedlichen rezeptiven Feldern unterscheidet, während des Sehprozesses beleuchtet. Schließlich werden die Vor- und Nachteile einer biologisch motivierten Bildverarbeitung mit pulscodierten Neuronen angegeben. Damit soll gezeigt werden, dass eine reine datengesteuerte (*Bottom-Up*) Verarbeitung, die nur eine spärlich vernetzte Netzwerkstruktur mit Vorwärtsverbindungen (*Feed-Forward*) benötigt, für eine effektive Objekterkennung nicht ausreichend ist. Es ist zusätzlich eine konzeptgesteuerte (*Top-Down*) Verarbeitung notwendig, die eine dichtere Vernetzungsstruktur mit Rückführungsverbindungen (*Feed-Back*) und Verbindungen für eine assoziative Speicherung erfordert. Die Simulation solcher pulscodierter neuronaler Netze ist jedoch aufgrund der Vielzahl an Neuronen und Synapsen sehr zeitintensiv.

2.1 Das biologische Neuron

Das biologische Neuron kann man als Zellkörper mit mehreren Eingängen und nur einem Ausgang betrachten, welches im wesentlichen aus 4 Funktionsteilen besteht: Dendriten, Soma, Axon und Synapsen (siehe Abb. 2-1).

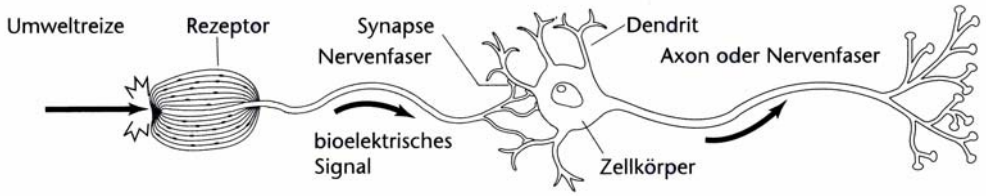


Abb. 2-1: Aufbau eines biolgischen Neurons [Gold02]

Dendriten, Soma, Axon und Synapsen

Die Dendriten übernehmen die Rolle der Eingänge, indem sie ankommende bioelektrische Signale in Form von Pulsen sammeln und an den Zellkörper weiterleiten. Die Dendriten sind durch einen Zellfortsatz mit dem Zellkörper verbunden und bilden einen Dendritenbaum. Man spricht von einem uni- und bipolaren Neuron, wenn es nur einen Dendritenbaum besitzt, und von einem multipolarem Neuron, wenn es wie in Abb. 2-1 mehrere Dendritenbäume hat [Kand00]. Die Eingangssignale eines Neurons können entweder Pulse von anderen Neuronen sein oder aus Umweltreizen resultieren, die von Rezeptoren aufgenommen werden. Der Zellkörper eines Neurons wird als Soma bezeichnet und stellt die “zentrale Recheneinheit” des Neurons dar [Gers02], welche nichtlineare

Verarbeitungsschritte durchführt. Wenn die Summe der eingehenden Eingangssignale eine bestimmte Schwelle überschreitet, wird ein Ausgangspuls generiert. Dieser Puls wird durch den einzigen Ausgang, das Axon, an andere Neuronen weitergeleitet. Das Axon ist von einer Myelinschicht umgeben, die gewährleistet, dass das Axon als perfekter Leiter fungieren kann, um den generierten Puls verlust- und störungsfrei über eine längere Distanz zu übertragen. Bi- und multipolare Neuronen besitzen einen separaten Zellfortsatz für das Axon, wobei unipolare Neuronen nur einen einzigen Zellfortsatz besitzen, den Dendriten und Axon gemeinsam nutzen. Die Verbindungselemente zwischen Neuronen werden als Synapsen bezeichnet. Über eine Synapse verläuft der Puls von einem sendenden (präsynaptischen) Neuron zu einem Puls-empfangenden (postsynaptischen) Neuron.

Ruhepotential

Das Membranpotential eines Neurons ist die Potentialdifferenz zwischen dem Zellinneren und Zelläußeren (Intra- bzw. Extrazellulärraum) [Birb02]. Das Ruhepotential u_0 eines Neurons wird im wesentlichen durch die intra- und extrazelluläre Konzentrationsverteilung von Kalium-Ionen (K^+, *Potassium*), Natrium-Ionen (Na^+, *Sodium*) und Chlorid-Ionen (Cl^-) bestimmt. Im Ruhezustand ist die Konzentration von Kalium-Ionen im Zellinneren $[K^+]_{in}$ am größten ($[K^+]_{in}$ | $[Na^+]_{in}$ | $[Cl^-]_{in}$ = 400 | 50 | 52 mMol/l), während im Extrazellulärraum eine hohe Konzentration von Natrium- und Chlorid-Ionen vorliegt ($[K^+]_{ex}$ | $[Na^+]_{ex}$ | $[Cl^-]_{ex}$ = 20 | 440 | 560 mMol/l) [Kand00]. Die Gleichgewichtspotentiale E_x (*Reversal Potential*) der Ionen lassen sich anhand der Nernst-Gleichung bestimmen [Gers02],

$$E_X = y\left(\frac{k \cdot T}{q} \cdot \ln\frac{[X^y]_{ex}}{[X^y]_{in}}\right) \approx y\left(25\text{mV} \cdot \ln\frac{[X^y]_{ex}}{[X^y]_{in}}\right) \qquad \begin{array}{l} k = 1,38 \cdot 10^{-23}\,\text{J/K} \\ T = 300\text{K} = 27°\text{C} \\ q = 1,6 \cdot 10^{-19}\,\text{C} \end{array} \qquad \textbf{(2-1)}$$

wobei y das Ladungsvorzeichen des Ions und $[X^y]_{ex}$ und $[X^y]_{in}$ die extra- und intrazelluläre Ionenkonzentration darstellen. Somit erhält man die Gleichgewichtspotentiale für K^+-, Na^+- und Cl^--Ionen (E_K | E_{Na} | E_{Cl} = -75 | +55 | -60 mV). Die Permeabilität der Zellmembran ist im Ruhezustand für K^+-Ionen am größten, so dass ein ständiger Ausstrom von K^+-Ionen erfolgt und im Zellinneren eine negative Ladung zurückbleibt. Da die Zellmembran auch im geringen Maße für Na^+-Ionen durchlässig ist, findet ebenfalls ein geringer Einstrom von Na^+-Ionen statt. Somit pendelt sich das Ruhepotential eines Neurons etwas oberhalb von E_K ein und liegt zwischen -60 mV und -70 mV [Kand00].

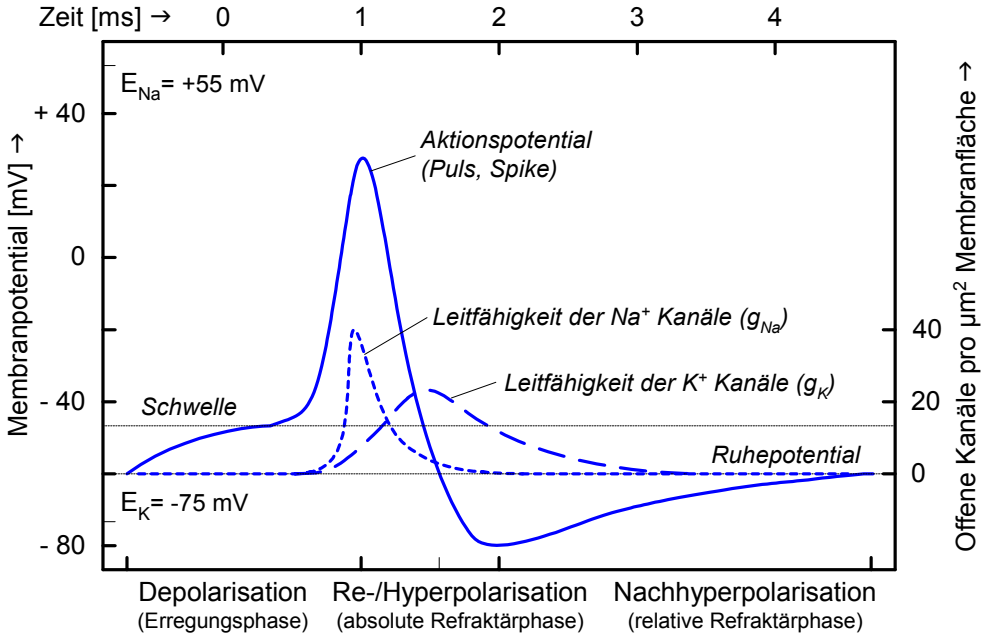


Abb. 2-2: Verlauf eines Aktionspotentials (basierend auf [Kand00] und [Birb02])

Aktionspotential

Der zeitliche Verlauf eines Aktionspotentials, der auch als Puls oder Spike bezeichnet wird, und die 3 unterschiedlichen Phasen, in denen sich ein Neuron befinden kann, sind in Abb. 2-2 dargestellt.

Depolarisation

Während der Depolarisation wird das Neuron erregt, und das Membranpotential steuert bei anhaltender Erregung vom Ruhepotential auf die Schwelle zu. Wird der Schwellwert erreicht, findet eine positive Rückkopplung statt, und das Membranpotential wird kurzzeitig positiv und erreicht einen Maximalwert. Während dieser Rückkopplung öffnen sich die Na^+-Kanäle sehr schnell und erzielen ihre maximale Anzahl an offenen Kanälen. Durch den Einstrom von Na^+-Ionen in den Zellkörper wird ein Spannungsanstieg des Membranpotentials verursacht. Die K^+-Kanäle öffnen sich in dieser Phase nur sehr langsam.

Re- und Hyperpolarisation

Nun beginnt die absolute Refraktärphase, in der die Erregbarkeit (Refraktärität) des Neurons aufgehoben ist. Die Dauer dieser Phase von etwa 1 ms beschränkt zudem die theoretisch maximal mögliche Pulsfrequenz eines Neurons auf 1 kHz. Am Anfang dieser Phase findet eine Repolarisation statt, wobei sich die Na^+-Kanäle schließen, während die K^+-Kanäle sich weiter öffnen und ihre maximale Anzahl an offenen Kanälen erreichen. Es erfolgt nun ein verstärkter Ausstrom von K^+-Ionen. Wenn das Membranpotential den Wert des Ruhepotentials unterschreitet, hat die Anzahl der ausströmenden K^+-Ionen die Anzahl der eingeströmten Na^+-Ionen übertroffen. Die Hyperpolarisation setzt ein und endet beim Erreichen eines Minimalwertes.

Nachhyperpolarisation

In der folgenden Nachhyperpolarisation, die auch als relative Refraktärphase bezeichnet wird, ist die Erregbarkeit bis zum Wiedererreichen des Ruhepotentials vermindert. Die Na^+-Kanäle sind geschlossen und der Austrom von K^+-Ionen vermindert sich weiter bis sich schließlich auch alle K^+-Kanäle schließen. Unter Energieaufwand werden dann die Na^+-Ionen anhand von Ionenpumpen aus dem Zellinneren heraustransportiert, und das Membranpotential erreicht wieder das Ruhepotential.

Dieses Verhalten der Ionenkanäle eines biologischen Neurons, welches für die dargestellte Pulsform in Abb. 2-2 verantwortlich ist, wird durch das Neuronenmodell von Hodgkin und Huxley modelliert [Hodg52]. Auf der Basis dieses Modells sind durch weitere Vereinfachungen die heutigen pulscodierten Neuronenmodelle entstanden.

2.1.1 Hodgkin-Huxley Modell

Das Hodgkin-Huxley Modell beschreibt die elektrischen Eigenschaften einer Zellmembran anhand des in Abb. 2-3 dargestellten Ersatzschaltbildes.

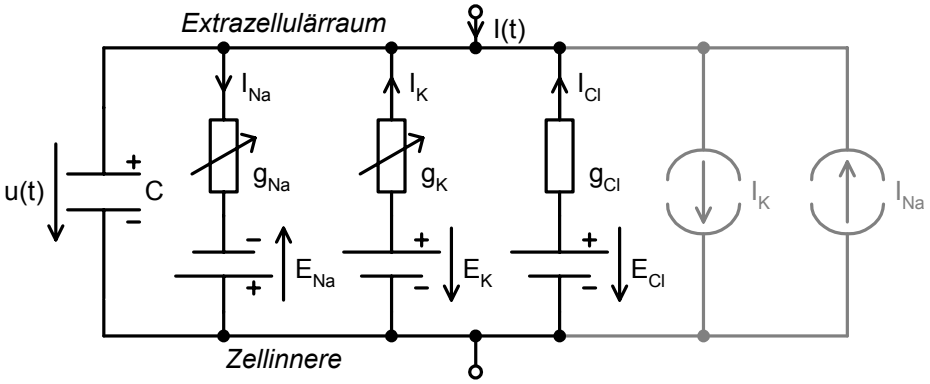


Abb. 2-3: Ersatzschaltbild des Neuronenmodells von Hodgkin und Huxley (basierend auf [Kand00])

Das Membranpotential wird durch die Spannung *u(t)* an der Membrankapazität *C* dargestellt, die zwischen dem Extrazellulärraum und dem Zellinneren abfällt. Die Gleichgewichtspotentiale der 3 Ionenkanäle werden anhand von Batterien repräsentiert, und die spannungsabhängigen Eigenschaften der Na^+- und K^+-Ionenkanäle (siehe Abb. 2-2) werden durch die variablen Leitwerte g_{Na} und g_K modelliert. Die beiden Stromquellen auf der rechten Seite des Ersatzschaltbildes stellen die Ionen-Pumpen dar, die im Ruhezustand ($I(t) = 0$) dem verstärkten Austritt des K^+-Ionenstroms I_K und dem geringen Eintritt des Na^+-Ionenstroms I_{Na} entgegenwirken.

Der Gesamtstrom *I(t)*, der bei Erregung in das Zellinnere gelangt, besteht aus einem kapazitiven Strom und aus der Summe aller durch die Zellmembran passierenden Ionenströme. Die Stromgleichung wird nach dem Hodgkin-Huxley Modell wie folgt beschrieben,

$$C \cdot \frac{du(t)}{dt} = -\{I_{Na}(t) + I_K(t) + I_{Cl}(t)\} + I(t)$$
$$= -\{g_{Na} \cdot m^3 \cdot h \cdot [u(t) - E_{Na}] + g_K \cdot n^4 \cdot [u(t) - E_K] + g_{Cl} \cdot [u(t) - E_{Cl}]\} + I(t) \quad \textbf{(2-2)}$$

3 Steuervariablen: h, m und n

wobei die Wahrscheinlichkeit, dass ein Ionenkanal geöffnet ist, durch die 3 Steuervariablen (*Gating Variables*) *h*, *m* und *n* beschrieben wird. Wie aus Gl. 2-2 ersichtlich, werden die Na^+-Kanäle aufgrund des kombinierten Verhaltens von *m* und *h* gesteuert, und die K^+-Kanäle werden durch *n* kontrolliert. Jede der 3 Steuervariablen wird über eine Differentialgleichung mit den beiden empirischen Funktionen $\alpha(u)$ und $\beta(u)$ beschrieben, die in Tab. 2-1 für die jeweilige Steuervariable aufgelistet sind [Gers02].

x	α_x(u/mV)	β_x(u/mV)
n	$[0,1 - 0,01 \cdot (u - u_r)] / [e^{(1-0,1\cdot(u-u_0))} - 1]$	$0,125 \cdot e^{-(u-u_0)/80}$
m	$[2,5 - 0,1 \cdot (u - u_r)] / [e^{(2,5-0,1\cdot(u-u_0))} - 1]$	$4 \cdot e^{-(u-u_0)/18}$
h	$0,07 \cdot e^{-(u-u_0)/20}$	$1 / [e^{(3-0,1\cdot(u-u_0))} + 1]$

Tab. 2-1: Empirische Funktionen α und β des Hodgkin-Huxley Modells (u_0 = Ruhepotential)

Die jeweilige Differentialgleichung der Steuervariablen *h*, *m* und *n* ist in Gl. 2-3 dargestellt,

$$\frac{dx(t)}{dt} = \alpha_x(u) \cdot (1 - x) - \beta_x(u) \cdot x$$
$$= -\frac{1}{[\alpha_x(u) + \beta_x(u)]^{-1}} \cdot \left[x - \frac{\alpha_x(u)}{\alpha_x(u) + \beta_x(u)}\right] \quad \textbf{(2-3)}$$
$$= -\frac{1}{\tau_x(u)} \cdot [x - x_0(u)]$$

Zeitkonstante und Gleichgewichtsfunktion

wobei *x* der Platzhalter der jeweiligen Steuervariable, $\tau_x(u)$ eine Zeitkonstante und $x_0(u)$ eine Gleichgewichtsfunktion (*Equilibrium Function*) ist. Der spannungsabhängige Verlauf der Zeitkonstanten und der Gleichgewichtsfunktionen ist in Abb. 2-4 wiedergegeben.

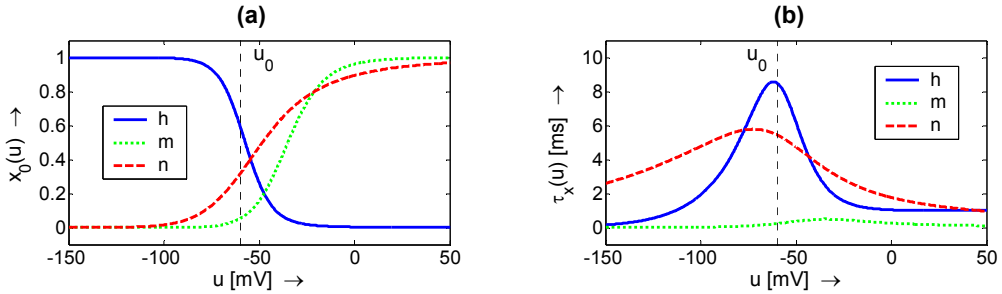


Abb. 2-4: (a) Gleichgewichtsfunktion und (b) Zeitkonstante der 3 Steuervariablen *h*, *m* und *n* des Hodgkin-Huxley Modells (Ruhepotential u_0 = -60 mV) (basierend auf [Gers02])

Aus Abb. 2-4a ist ersichtlich, dass *m* und *n* bei zunehmendem Membranpotential steigen, während *h* abfällt. Wird das Membranpotential durch eine Erregung angehoben, erhöht sich die Leitfähigkeit der Na^+-Kanäle aufgrund des Anstiegs von *m*. Bei hohen Spannungswerten wird die Leitfähigkeit der Na^+-Kanäle durch *h* aufgehoben. In Abb. 2-4b ist dargestellt, dass die Zeitkonstante τ_h stets größer ist als τ_m. Die Steuervariable *h*, die für das Schließen der Na^+-Kanäle zuständig ist, reagiert somit langsamer als die Steuervariable *m*, die das Öffnen der Kanäle modelliert [Gers02]. Der langsame An- und Abstieg der K^+-Ionenleitfähigkeit wird durch die Steuervariable *n* beschrieben.

Der Vorteil des Hodgkin-Huxley Modells liegt in der Feststellung, dass die Ionenleitfähigkeiten von Natrium und Kalium die fundamentalen physikalischen Prozesse für den Aktionspotentialverlauf in Neuronen sind. Die genaue Modellierung des Aktionspotentials anhand von 4 Differentialgleichungen (siehe Gl. 2-2 und Gl. 2-3) ist aber aufgrund der

stets identischen Form für die neuronale Informationsverarbeitung nicht relevant [MacG87]. Hierfür ist eher die neuronale Dynamik von Interesse, deren wesentliche Elemente nun im einzelnen vorgestellt werden.

2.1.2 Elemente neuronaler Dynamik

Die neuronale Dynamik beruht auf Effekten, die ein Puls eines präsynaptischen Neurons auf ein postsynaptisches Neuron bewirkt. Diese Effekte werden durch das Verbindungselement zwischen 2 Neuronen gesteuert, der Synapse. Die Überleitung eines präsynaptischen Pulses wird von einer Synapse mit Hilfe von chemischen Substanzen, die als Neurotransmitter bezeichnet werden, bewerkstelligt. Dabei müssen die Neurotransmitter den synaptischen Spalt überwinden, der jede Synapse von einem postsynaptischen Neuron trennt. Auf der Seite des postsynaptischen Neurons werden die Neurotransmitter schließlich von geeigneten Rezeptoren aufgenommen.

2.1.2.1 Synapsen

Wirksamkeit

Synapsen können eine erregende oder eine hemmende Wirksamkeit (*Efficacy*) auf ein postsynaptisches Neuron ausüben. Synapsen, die einen Membranpotentialanstieg am postsynaptischen Neuron bewirken und somit eine erregende Wirksamkeit auf das Neuron haben, werden als exzitatorisch bezeichnet (90 % aller Synapsen eines Neurons sind erregend [Abel91]). Exzitatorische Synapsen verwenden Glutamat als Neurotransmitter und reagieren auf 2 Haupttypen von Rezeptoren: NMDA (*N-Methyl-D-Aspartate*) und AMPA (*Alpha-amino-3-hydroxy-5-Methylisoxazole-4-Proprionic Acid*). Synapsen, die eine hemmende Wirksamkeit auf ein postsynaptisches Neuron ausüben, da sie einen Membranpotentialabfall am postsynaptischen Neuron verursachen, werden als inhibitorisch bezeichnet (10 % aller Synapsen eines Neurons sind hemmend [Abel91]). Inhibitorische Synapsen verwenden GABA (*Gamma-Amino-Butyric Acid*) als Neurotransmitter und reagieren auf die Rezeptoren $GABA_A$ und $GABA_B$ [Kand00].

Plastizität

Das dynamische Verhalten von Synapsen, welches die Erhöhung und Erniedrigung der Wirksamkeit umfasst, wird als synaptische Plastizität bezeichnet. Wenn der Effekt der erhöhten bzw. erniedrigten Wirksamkeit von Synapsen über einen längeren Zeitraum anhält spricht man von LTP (*Long-Term Potentiation*) bzw. LTD (*Long-Term Depression*). Ist der Effekt von erhöhter bzw. erniedrigter Wirksamkeit von Synapsen allerdings nur von kurzer Dauer handelt es sich um STP (*Short-Term Potentiation*) bzw. STD (*Short-Term Depression*).

Hebb'sche Lernregel

Die Modellierung von Synapsen erfolgt anhand von Gewichtswerten, wobei man zwischen dynamischen und statischen Synapsen unterscheidet. Bei dynamischen Synapsen werden die Gewichtsparameter und -werte ständig adaptiert, wo hingegen bei statischen Synapsen die Gewichtswerte fest eingestellt sind. Der Prozess der Adaption von Gewichtsparametern und -werten wird als Lernen bezeichnet, und die Prozedur, nach der die Justierung der Gewichtswerte erfolgt, nennt man Lernregel. Die Hebb'sche Lernregel [Hebb49] besagt, dass die Wirksamkeit der synaptischen Verbindungen zwischen Neuron N_A und N_B erhöht wird, wenn ein Puls von N_A das Neuron N_B zum feuern anregt. Dadurch ist N_A in der Lage, durch einen Puls N_B früher zum feuern zu veranlassen. Diese Lernregel entspricht den Beobachtungen in der Biologie und wird durch 2 Eigenschaften charakterisiert [Gers02]: Lokalität (*Locality*) und Mitwirkung (*Cooperativity*). Lokalität bedeutet, dass die Modifikation der synaptischen Verbindung nur anhand von lokalen Variablen bestimmt wird, die von einer Synapse aus betrachtet zugänglich sind. Dies sind die Zustände des prä- und postsynaptischen Neurons sowie die momentane synaptische Wirksamkeit. Mitwirkung bedeutet, dass eine Modifikation der synaptischen Verbindung nur erfolgen kann, wenn prä- und postsynaptisches Neuron zur gleichen Zeit aktiv sind. Ein präsynaptisches Neuron ist aktiv, wenn es einen Puls sendet. Aktivität bei einem postsynaptischen Neuron kann unterschiedlich interpretiert werden. Das postsynaptische Neuron kann beispielsweise als aktiv betrachtet werden, wenn es innerhalb eines definierten Zeitfensters nach

dem präsynaptischen Puls ebenfalls einen Puls generiert oder sich das Membranpotential zum Zeitpunkt des präsynaptischen Pulses über einem bestimmten Wert befindet.

2.1.2.2 Postsynaptische Potentiale

Die Spannungsantwort eines postsynaptischen Neurons auf einen präsynaptischen Puls wird als PSP (postsynaptisches Potential) bezeichnet. Erregende Synapsen verursachen ein EPSP (exzitatorisches PSP) und hemmende Synapsen ein IPSP (inhibitorisches PSP).

Depolarisationsgrad

Je nach Depolarisationsgrad des postsynaptischen Neurons können 2 Effekte auftreten [Gers02]: eine sättigende Erregung (*Saturation Excitation*) und eine kurzschließende Hemmung (*Shunting Inhibition*). Exzitatorische Synapsen haben ein Gleichgewichtspotential, welches oberhalb des Schwellwertes liegt ($E_{SYN} \approx 0$ mV), und inhibitorische Synapsen verfügen über ein Gleichgewichtspotential, welches unterhalb des Ruhepotentials liegt und dem Gleichgewichtspotential von Kalium-Ionen entspricht ($E_{SYN} \approx E_K$) [Gers02]. Die Amplitude des PSPs ist abhängig vom postsynaptischen Ionenstrom I_{PSC} (*PSC = Postsynaptic Current*), der sich proportional zur Spannungsdifferenz zwischen dem postsynaptischen Membranpotential und dem Gleichgewichtspotential des entsprechenden Synapsentyps E_{SYN} verhält (siehe Gl. 2-4).

$$I_{PSC} \sim [u(t) - E_{SYN}] \qquad \textbf{(2-4)}$$

Diese Proportionalität wird auch im Hodgkin-Huxley Modell in Gl. 2-2 auf Seite 10 beschrieben. Wenn sich das Membranpotential des postsynaptischen Neurons nah am Schwellwert befindet, nimmt die Amplitude eines EPSPs ab, da die Spannungsdifferenz in Gl. 2-4 kleiner wird. Es tritt eine Sättigung der Erregung des postsynaptischen Neurons ein. Die Amplitude eines IPSPs dagegen nimmt mit steigendem Membranpotential zu. Befindet sich das Membranpotential geringfügig unterhalb der Schwelle, hat ein einziges IPSP aufgrund der sehr hohen Amplitude eine kurzschließende Hemmwirkung und das Membranpotential wird auf das Ruhepotential heruntergezogen.

Pulszeitpunkte

Um ein postsynaptisches Neuron zum Feuern anzuregen, sind zusätzlich die Zeitpunkte der präsynaptischen Pulse von Bedeutung, die bei einer erregenden Verbindung für die Generierung der EPSPs verantwortlich sind. Nach dem Erreichen der maximalen Amplitude klingen die PSPs exponentiell ab. Aufgrund der nicht-linearen Interaktion zwischen verschiedenen präsynaptischen Pulsen ist es notwendig, dass die Pulse in kurzen Zeitabständen aufeinanderfolgend eintreffen. Somit können sich die EPSPs entsprechend aufsummieren, und das Membranpotential ist in der Lage den Schwellwert zu erreichen.

Räumliche Anordnung von Dendriten

Im Gegensatz zum Axon eines Neurons, welches aufgrund der Myelinschicht verlustfrei Pulse übertragen kann, sind die Eingänge eines Neurons in Form des Dendriten verlustbehaftet. Aufgrund dieser Tatsache spielt die räumliche Anordnung einzelner Dendriten an einem Dendritenbaum bei der neuronalen Dynamik eine wichtige Rolle. Die PSPs, die über einen Dendriten übertragen werden, der weit vom Soma entfernt ist, gelangen mit einer geringeren Amplitude zum Neuron. Hingegen können PSPs, die über einen Dendriten verlaufen, der nah am Soma angebracht ist, das Neuron stärker stimulieren und zum Feuern bringen ($u(t) \geq \theta$).

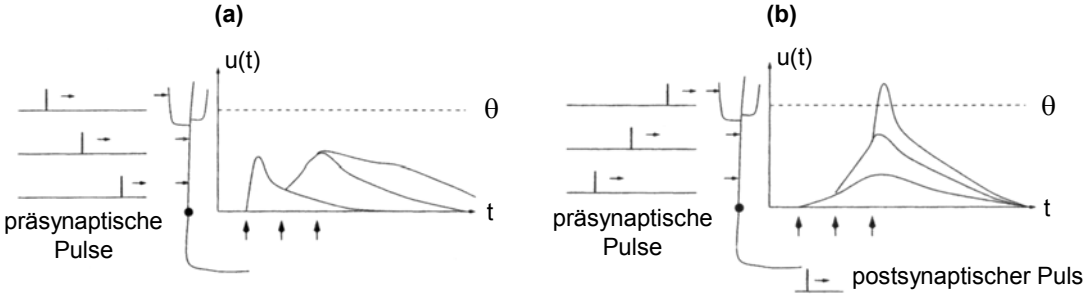


Abb. 2-5: (a) Präsynaptischer Puls nah am Soma stimuliert zuerst das Neuron, (b) Präsynaptischer Puls an der Spitze des Dendritenbaums stimuliert zuerst das Neuron (modifiziert von [Gers02])

Aus Abb. 2-5 wird diese Dynamik ersichtlich, und man erkennt die strategisch günstige Position, wenn PSPs nur eine kurze Strecke über den Dendriten zum Neuron zurücklegen müssen. An der Spitze eines Dendritenbaums befinden sich zum größten Teil exzitatorische Synapsen, während die meisten inhibitorischen Synapsen aufgrund der größeren Einflussnahme nah am Soma angebracht sind [Kand00].

Mehrere Dendritenbäume

Bei multipolaren Neuronen, die mehrere Dendritenbäume besitzen, ist zusätzlich ausschlaggebend, ob IPSPs und EPSPs über denselben Dendritenbaum übertragen werden. Bei der Übertragung eines IPSPs kommt es zu einer Änderung der Leitfähigkeit am Dendritenbaum, und kurzzeitig nachfolgende EPSPs am selben Dendritenbaum können das Neuron nicht zur Pulsauslösung veranlassen. Dieser Effekt wird als stumme Hemmung (*Silent Inhibition*) bezeichnet. Erfolgen die EPSPs aber über einen anderen Dendritenbaum, der kein vorausgegangenes IPSP erfahren hat, wird das Neuron dazu gebracht, einen Puls auszulösen [Gers02].

2.1.2.3 Ionenkanäle

Neben den spannungsaktivierenden (*Voltage-Activated*) Ionenkanälen Na^+ und K^+, die vom Hodgkin-Huxley Modell berücksichtigt werden, gibt es noch 2 weitere Haupttypen von Ionenkanälen, die zur neuronalen Dynamik beitragen. Zum einen die Calzium-aktivierenden und zum anderen die Transmitter-aktivierenden Ionenkanäle [Gers02].

Calzium-aktivierend

Eine interessante Eigenschaft bei Calzium-Ionenkanälen (Ca^{2+}) ist, dass sie nach einem hemmenden Reiz einen Puls am postsynaptischen Neuron generieren können. Dieses Phänomen wird als “Rücksprung nach einem inhibitorischen Reiz” (*Postinhibitory Rebound*) bezeichnet [Gers02]. Dieser Effekt tritt auf, wenn das Neuron von der Hyperpolarisation nach der Inhibition wieder in den Ruhezustand gelangt. Ca^{2+}-Ionen verhalten sich wie Na^+-Ionen und haben ebenfalls 2 Steuervariablen h und m, die das Schließen und Öffnen des Ionenkanals beschreiben. Allerdings sind die spannungsabhängigen Verläufe der beiden Steuervariablen im Vergleich zu Na^+-Kanälen (siehe Abb. 2-4a auf Seite 10) in Richtung eines hyperpolarisierenden Membranpotentials (nach links) verschoben. Dies bedeutet, dass beim Ruhepotential bereits die Bedingungen $h = 0$ und $m > 0$ gelten, und während der Hyperpolarisation, wie bei den Na^+-Kanälen, $h > 0$ und $m = 0$ ist. Beim Übergang von der Hyperpolarisation zum Ruhepotenzial gibt es nun eine kurze “Rücksprung”-Periode, in der beide Steuervariablen ungleich Null sind, und somit ein kurzzeitiger Ionenstrom erzeugt wird, der das Neuron in den Pulszustand versetzen kann.

Transmitter-aktivierend

Transmitter-aktivierende Ionenkanäle beschreiben die vielfältigen Dynamiken, die Neurotransmitter verursachen, um anhand von Rezeptoren unterschiedliche Ionenkanäle am postsynaptischen Neuron zu öffnen. Bei exzitatorischen Synapsen beispielsweise reagieren die AMPA-Rezeptoren sehr schnell auf die Ausschüttung der Neurotransmitter, wobei exzitatorische Synapsen mit NMDA-Rezeptoren ein deutlich langsameres Reaktionsverhalten aufweisen [Kand00]. Dieses langsame Verhalten kommt aufgrund von blockierenden Magnesium-Ionen (Mg^{2+}) zustande. Wenn sich das postsynaptische Neuron im Ruhezustand ($u(t) \approx u_0$) befindet, sind die NMDA-Rezeptoren nicht in der Lage mit Hilfe der ausgeschütteten Neurotransmitter, die Ionenkanäle zu öffnen, da diese durch Mg^{2+}-Ionen blockiert sind. Erst wenn sich das postsynaptische Neuron in einem depolarisierendem Zustand befindet ($u(t) \gg u_0$), können die Mg^{2+}-Ionen entfernt und somit der präsynaptische Puls übertragen werden. Eine Depolarisation kann beispielsweise durch präsynaptische Pulse von exzitatorischen Synapsen mit AMPA-Rezeptoren erfolgen oder aber durch ein Aktionspotential des Neurons selbst. Experimentelle Messungen haben bewiesen, dass sich ein Aktionspotential nicht nur über das Axon fortpflanzt, sondern auch wieder in den Dendritenbaum zurückläuft und dort das Potential kurzzeitig anheben kann [Kuhn01]. Dieses Phänomen nennt man zurücklaufendes Aktionspotential (*Back Propagating Action Potential*). Es ist also bei dieser Art von Transmitter-aktivierendem Ionenkanal eine postsynaptische Aktivität notwendig, wie sie von der Hebb’schen Lernregel gefordert wird.

Nachdem nun ein Eindruck über die Vielfältigkeit der neuronalen Dynamik gegeben wurde, stellt sich nun die Frage, wie die Pulscodierung von Neuronen erfolgt, die eine effiziente Informationsverarbeitung anhand von Pulsen gewährleistet.

2.1.3 Neuronale Pulscodierung

Die neuronale Pulscodierung beruht auf der Annahme, dass relevante Informationen in den generierten Pulsfolgen von Neuronen enthalten sind. Aufgrund von experimentellen Messungen kann man 4 Haupttypen von pulsenden Neuronen unterscheiden [Gers02]: regulär pulsende (*Regular-Spiking*), schnell pulsende (*Fast-Spiking*), explosiv pulsende *(Bursting*) und vereinzelt pulsende (*Single-Spiking*) Neuronen. Bei regulär pulsenden Neuronen findet im Gegensatz zu schnell pulsenden Neuronen eine Adaption statt bis ein stabiler Zustand der Feuerperiode erreicht ist. Explosiv pulsende Neuronen generieren eine Sequenz von schnell aufeinanderfolgenden Pulsen, die periodisch durch längere Intervalle unterbrochen werden. Vereinzelt pulsende Neuronen generieren bei Stimulierung nur einen einzigen Puls.

Einzelzelle und Population

Es gibt nun 2 Betrachtungsweisen, die jeweils 2 unterschiedliche Codierungsstrategien gegenüberstellen, wie man die neuronalen Pulse interpretieren kann. Zum einen differenziert man zwischen Einzelzellen- und Populationscodierung [Gold02]. Die Einzelzellencodierung geht davon aus, dass jeweils ein einziges Neuron für eine einzige Art von Stimulus verantwortlich ist (*Master Neuron*). Dies würde bedeuten, dass bei Defekt oder Ausfall dieses Neurons der zugehörige Stimulus nicht mehr wahrgenommen werden würde. Die realistischere Populationscodierung dagegen beruht auf unterschiedlichen Feuermustern eines Netzwerks von Neuronen, die bei unterschiedlichen Stimuli entstehen können. Weiterhin unterscheidet man zwischen Raten- und Pulscodierung [Gers02]. Die Ratencodierung (*Rate Code*) interpretiert den neuronalen Code aufgrund der Pulsrate und die Pulscodierung (*Spike Code*) misst den Zeitpunkten von Pulsen ihre Bedeutung bei. Die zweite Betrachtungsweise der neuronalen Pulscodierung, die Raten- und Pulscodierung unterscheidet, wird nun etwas näher beleuchtet.

2.1.3.1 Ratencodierung

Anzahl, Dichte, Aktivität

Bei der Ratencodierung unterscheidet man 3 Codierungsstrategien: Pulsanzahlrate, Pulsdichtenrate und Populationsaktivitätsrate. Bei der Pulsanzahlrate wird die Pulsfrequenz eines einzelnen Neurons über eine bestimmte Zeitspanne ermittelt. Der zeitlich veränderbare Stimulus stimuliert das Neuron hierbei nur ein einziges Mal (*Single Neuron, Single Run*). Die Pulsdichtenrate erhält man, indem man die mittlere Pulsfrequenz eines Neurons ermittelt. Dies ist möglich, wenn der gleiche Stimulusverlauf mehrere Male aufeinanderfolgend das Neuron stimuliert (*Single Neuron, Several Runs*).

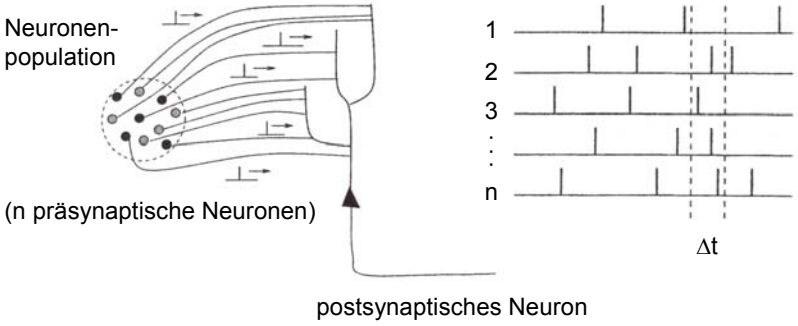


Abb. 2-6: Ratencodierung als Populationsaktivität (modifiziert von [Gers02])

Die Populationsaktivitätsrate betrachtet eine Neuronenpopulation, also eine Gruppe von Neuronen. Die Aktivität dieser Neuronen wird ermittelt, indem man, wie in Abb. 2-6 dargestellt, die Anzahl der Pulse in einem bestimmten Zeitintervall Δt durch die Gesamtanzahl der Neuronen in der Population mittelt (*Several Neurons, Single Run*).

2.1.3.2 Pulscodierung

Erster Puls, Phase, Synchronität, Rekonstruktion

Bei der Pulscodierung werden 4 Codierungsstrategien unterschieden: “Zeitintervall bis zum ersten Puls” (*Time-To-First-Spike*), Phase, Synchronität (Korrelation) und Stimulus Rekonstruktion (reversible Korrelation). Bei dem “Zeitintervall bis zum ersten Puls”-Code wird angenommen, dass nach Anlegen eines Stimulus nur der zuerst auftretende Puls eines Neurons relevante Informationen enthält. Diese Strategie wird durch die Tatsache gestützt, dass visuelle Reize eines neuen Stimulus innerhalb einer sehr kurzen Zeitspanne von 150 ms verarbeitet werden können [Thor96]. Wenn der Stimulus ein oszillierendes Signal ist, könnten Pulsfolgen eines Neurons, die sich in Phase zu dem Stimulus Signal befinden, ebenfalls relevante Informationen enthalten. Betrachtet man nun die Pulse von mehreren Neuronen kann das synchrone Feuern von verschiedenen Neuronenpopulationen, die auf unterschiedliche visuelle Eigenschaften reagieren, bedeuten, dass diese visuellen Eigenschaften von demselben gesehenen Objekt repräsentiert werden. Diese Synchronität bzw. Korrelation von Pulsen zeigt, dass unterschiedliche visuelle Eigenschaften, wie Form, Größe oder Farbe, einem Objekt zugeordnet werden können [Mals81]. Bei der Stimulus Rekonstruktion (siehe Abb. 2-7) wird versucht, die Bedeutung einzelner Pulse zu ermitteln, indem man identisch vorangegangene Stimulusverläufe ermittelt. Mittels dieser Codierungsstrategie ist es möglich einen neuronalen Pulscode zu “lesen” und den zeitabhängigen Stimulus zu rekonstruieren [Gers02].

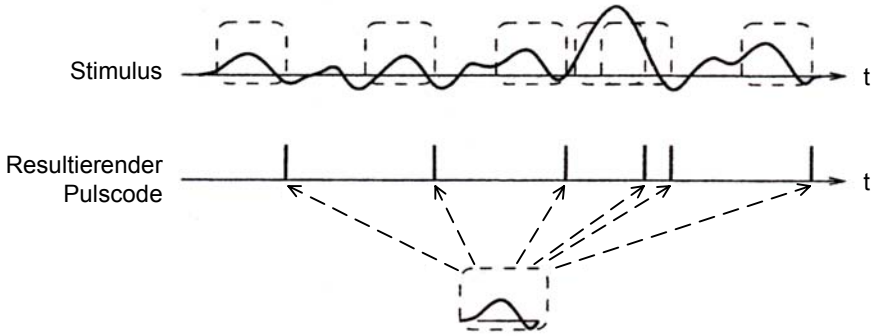


Abb. 2-7: Pulscodierung zur Stimulus Rekonstruktion (modifiziert von [Gers02])

Eine klare Trennung zwischen Raten- und Pulscodierung ist allerdings je nach Interpretation nicht möglich. Der Pulscode der Stimulus Rekonstruktion kann auch als Ratencode basierend auf der Pulsanzahl interpretiert werden. Ebenso gibt der “Time-To-First-Spike”-Pulscode Informationen über die Pulsrate eines Neurons, da das Zeitintervall bis zum ersten Puls bei Neuronen mit niedriger Pulsrate länger ist als bei Neuronen mit einer hohen Pulsrate. Dieser Pulscode kann ebenfalls aufgrund der alleinigen Betrachtung des ersten Pulses als eine unrealistische Einzelzellencodierung verstanden werden. Der Ratencode basierend auf der Populationsaktivität in Abb. 2-6 erscheint zum jetzigen Forschungsstand am wahrscheinlichsten, da er einer realistischen Populationscodierung entspricht (Ausfall oder Defekt einzelner Neuronen können kompensiert werden) und zugleich für eine zeitlich schnelle Informationsverarbeitung (Δt in Abb. 2-6) geeignet ist [Gers02].

2.2 Pulscodierte Neuronenmodelle

Das Hodgkin-Huxley Modell stellt einen erheblichen Berechnungsaufwand für die Modellierung eines einzigen Neurons dar, da insgesamt 4 Differentialgleichungen für das Membranpotential und die dazugehörigen 3 Steuervariablen *m*, *h* und *n* gelöst werden müssen.

Rinzel Modell

Eine Vereinfachung dieses komplexen Neuronenmodells ergibt sich aus 2 Beobachtungen nach Rinzel, die die Modellierung des Membranpotentials mit nur 2 Differentialgleichungen erlaubt [Rinz85]. Als erstes kann man die Steuervariable *m* als unverzögerte

Variable betrachten (siehe Abb. 2-4b auf Seite 10), die sich nur in Abhänigkeit des Membranpotentials ändert (siehe Abb. 2-4a). Aus dieser zeitlichen Unabhängigkeit folgt aus Gl. 2-3 auf Seite 10:

$$\frac{dm(t)}{dt} = -\frac{1}{\tau_m(u)} \cdot [m - m_0(u)] = 0 \Leftrightarrow m = m_0(u) \qquad \textbf{(2-5)}$$

Weiterhin lässt sich die Modellierung der anderen beiden Steuervariablen *h* und *n* zusammenfassen, da nach Abb. 2-4a eine lineare Abhängigkeit zwischen $n_0(u)$ und $h_0(u)$ besteht und für die Zeitkonstanten $\tau_n(u) \approx \tau_h(u)$ gilt. Somit erhält man nach Gl. 2-2 und Gl. 2-3 eine vereinfachte Beschreibung des Membranpotentials,

$$\begin{aligned} C \cdot \frac{du(t)}{dt} &= -\{I_{Na}(t) + I_K(t) + I_{Cl}(t)\} + I(t) \\ &= -\{g_{Na} \cdot m_0{}^3 \cdot (1-w) \cdot [u(t) - E_{Na}] + g_K \cdot w^4 \cdot [u(t) - E_K] + g_{Cl} \cdot [u(t) - E_{Cl}]\} + I(t) \qquad \textbf{(2-6)} \\ \frac{dw(t)}{dt} &= -\frac{1}{\tau_w(u)} \cdot [w - w_0(u)] \end{aligned}$$

wobei *w* die einzige zeitlich abhängige Steuervariable darstellt, die die lineare Abhängigkeit von *n* und *h* widerspiegelt ($h \cong [1 - w]$ und $n \cong w$).

Eine weitere Vereinfachung führt zu den pulscodierten Neuronenmodellen, die nicht mehr als eine Differentialgleichung für die Modellierung des Membranpotentials eines Neurons berücksichtigen.

2.2.1 Integrate-and-Fire Modell

Das bekannteste pulscodierte Neuronenmodell ist das Integrate-and-Fire Modell von Stein [Stei67], dessen Ersatzschaltbild in Abb. 2-8 dargestellt ist.

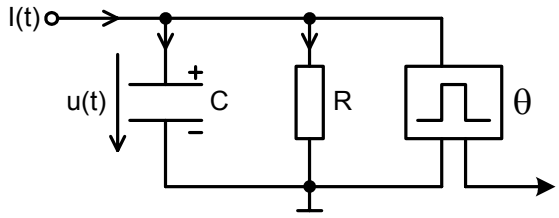


Abb. 2-8: Ersatzschaltbild des Integrate-and-Fire Modells

Die Spannung *u(t)*, die über der Kapazität *C* abfällt, stellt das Membranpotential dar. Der Widerstand *R* wird als Leck-Widerstand bezeichnet, über dessen Strompfad sich die Kapazität langsam entladen kann. Der Eingangsstrom *I(t)*, der das Neuron stimuliert, fließt parallel durch *C* und *R* und setzt sich somit aus einem kapazitiven und einem resistiven Teilstrom wie folgt zusammen,

$$I(t) = \frac{u(t)}{R} + C \cdot \frac{du(t)}{dt} \Leftrightarrow \tau \cdot \frac{du(t)}{dt} = -u(t) + R \cdot I(t) \qquad \textbf{(2-7)}$$

wobei $\tau = R \cdot C$ der Zeitkonstante des Membranpotentials entspricht. Die entstandene Differentialgleichung für *u(t)* in Gl. 2-7 wird auch als Leck-Integrator (*Leaky Integrator*) bezeichnet [Gers02]. Wenn das Membranpotential den Schwellwert θ erreicht hat, fängt das Neuron an zu feuern (*Fire*), und ein Puls wird generiert. Zeitgleich wird der Strompfad auf der rechten Seite in Abb. 2-8 nach Masse hin kurzgeschlossen und dadurch das Membranpotential auf 0 V zurückgesetzt. Mit der Zeitdauer, in der dieser Strompfad durchgeschaltet ist, kann man die absolute Refraktärphase Δ_{abs} eines Neurons modellieren, in der das Membranpotential auf keine Änderungen des Eingangsstroms reagiert.

Konstanter Eingangstrom

Wenn man einen zeitlich konstanten Eingangsstrom $I(t) = I_0$ annimmt, ergibt sich durch Lösung der Differentialgleichung (siehe Anhang B auf Seite 167) in Gl. 2-7 folgender Membranpotentialverlauf:

$$u(t) = R \cdot I_0 \cdot \left(1 - e^{-\frac{t - t_0}{\tau}}\right) \tag{2-8}$$

Das Membranpotential nähert sich in diesem Fall asymptotisch dem Spannungswert $R \cdot I_0$ an. Wenn die Bedingung $R \cdot I_0 < \theta$ gilt, wird das Neuron niemals feuern. Wenn allerdings die Bedingung $R \cdot I_0 > \theta$ erfüllt ist, lässt sich aus Gl. 2-8 die Feuer- bzw. Pulsfrequenz ν des Neurons bestimmen [Gers02],

$$\theta = R \cdot I_0 \cdot \left(1 - e^{-\frac{T}{\tau}}\right) \Leftrightarrow T = \tau \cdot \ln\left[\frac{R \cdot I_0}{R \cdot I_0 - \theta}\right]$$
$$\Rightarrow \nu = \left(\Delta_{abs} + \tau \cdot \ln\left[\frac{R \cdot I_0}{R \cdot I_0 - \theta}\right]\right)^{-1} \tag{2-9}$$

wobei Δ_{abs} die Zeitdauer der absoluten Refraktärphase darstellt.

2.2.2 Spike-Response Modell

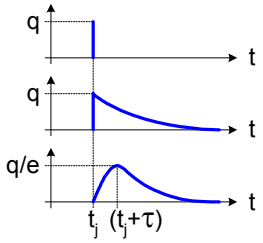
Das Spike-Response Modell, abgekürzt SRM, von Gerstner [Gers98] ist ein pulscodiertes Neuronenmodell, welches im Gegensatz zum Integrate-and-Fire Modell das Membranpotential nicht als Differentialgleichung ausdrückt, sondern als Integral über den vergangenen Membranpotentialverlauf. Zusätzlich sind die verwendeten Modellparameter (*Kernel*) im Spike-Response-Modell abhängig vom Zeitpunkt t_i, an dem das Neuron i seinen letzten Puls generiert hat, und von den Zeitpunkten t_j, an denen die jeweiligen präsynaptischen Neuronen einen Puls generiert haben. Das Membranpotential $u_i(t)$ setzt sich aus 3 Additionstermen zusammen, die jeweils von einem der 3 Modellparameter η, ε und κ abhängig sind:

$$u_i(t) = \eta(t - t_i) + \sum_j w_{ij} \sum_f \varepsilon_{ij}(t - t_i, s) + \int_0^\infty \kappa(t - t_i, s) \cdot I(t - s)ds \quad | \; s = (t - t_j) \tag{2-10}$$

3 Modellparameter: η, ε und κ

Der erste Modellparameter η beschreibt die Form des Aktionspotentials unter Berücksichtigung der Hyperpolarisation und wird dem Membranpotential immer hinzugefügt, wenn $u_i(t)$ den Schwellwert erreicht. Der Einfluss von η endet, wenn die absolute Refraktärphase beendet ist (siehe auch Abb. 2-2 auf Seite 8). Der zweite Modellparameter ε modelliert den Zeitverlauf eines postsynaptischen Potentials, welches durch den Puls eines präsynaptischen Neurons j verursacht wird. Eine Gewichtung der postsynaptischen Potentiale erfolgt anhand der Synapsengewichte w_{ij}, die unter Umständen eine Vielzahl von präsynaptischen Neuronen j mit dem zu betrachtenden postsynaptischen Neuron i verbinden. Der zeitliche Verlauf eines postsynaptischen Potentials $w_{ij} \cdot \varepsilon_{ij}$ wird meistens mit Hilfe einer Alpha-Funktion $\alpha(t - t_j)$ modelliert [Gers02]. In Gl. 2-11 sind 3 Varianten der Alpha-Funktion illustriert,

$$\alpha(t - t_j) = \begin{cases} q \cdot \delta(t - t_j) \\ q \cdot \Theta(t - t_j) \cdot e^{-\frac{t - t_j}{\tau}} \\ q \cdot \Theta(t - t_j) \cdot \frac{t - t_j}{\tau} \cdot e^{-\frac{t - t_j}{\tau}} \end{cases} \tag{2-11}$$



wobei q eine Konstante ist, die die maximale Amplitude des Potentialverlaufs bestimmt und $\Theta(t - t_j)$ eine Schwellenwert-Funktion (*Heaviside Function*) mit den Eigenschaften $\Theta(t < 0) = 0$ und $\Theta(t \geq 0) = 1$ darstellt. Der dritte Modellparameter κ beschreibt das lineare Verhalten des Membranpotentials gegenüber einem Eingangsstrom $I(t)$. Auf einen kurzen Strompuls antwortet κ mit einem schwachen und kurzen Verlauf, wenn das Neuron kurz zuvor einen Puls gesendet hat und sich somit noch in der relativen Refraktärphase befin-

det. Wird das Neuron allerdings von einem kurzen Strompuls stimuliert, wenn es sich wieder im Ruhezustand befindet, reagiert κ mit einem stärkeren und längeren Verlauf.

Refraktärität

Das Spike-Response Modell verwendet zusätzlich einen dynamischen Schwellwert $\theta(t - t_i)$, der erstmals von French und Stein [Fren70] als Erweiterung zum Integrate-and-Fire Modell eingeführt wurde. Dieser dynamische Schwellwert ist wie das Membranpotential vom letzten Feuerzeitpunkt des Neurons abhängig ist. Die Berücksichtigung des letzten Feuerzeitpunkts t_i sowohl für das Membranpotential als auch für den Schwellwert stellt beim Spike-Response Modell die Modellierung der refraktären Eigenschaften eines Neurons in den Vordergrund. Diese Modellierung setzt sich aus dem Aktionspotential mit Hyperpolarisation ($\eta(t - t_i)$), der abgeschwächten Reaktion kurz nach einem präsynaptischen Puls ($\varepsilon(t - t_i)$) oder externen Strompuls ($\kappa(t - t_i)$) und der Erhöhung des ansonsten exponentiell abklingenden Schwellwertes ($\theta(t - t_i)$) zusammen.

2.2.3 Marburger Modell

Das Marburger Modell von Eckhorn [Eckh90] basiert auf Eigenschaften des Integrate-and-Fire und des Spike-Response Modells und ist in Abb. 2-9 dargestellt.

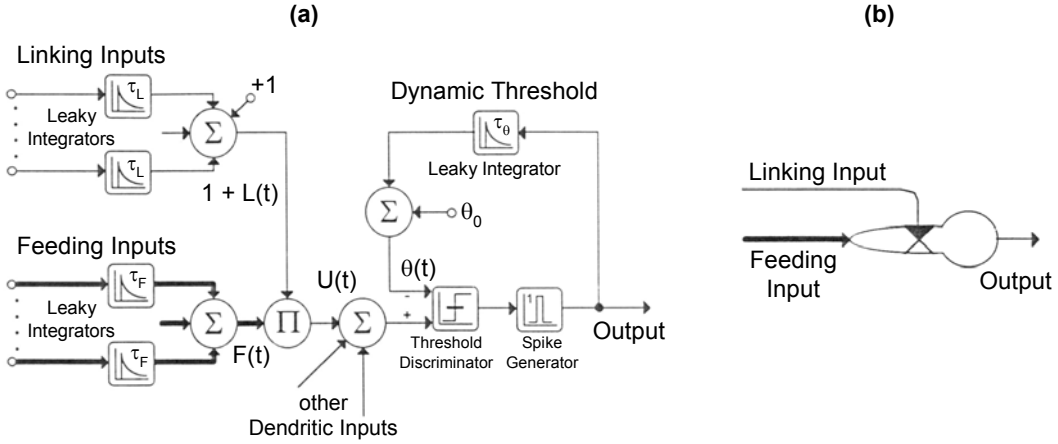


Abb. 2-9: (a) Schaltbild des Marburger Neuronenmodells, (b) Symbol eines Marburger Modellneurons (modifiziert von [Stöc96])

Feeding/Linking Synapsen

Eine Besonderheit beim Marburger Modell ist, dass es 2 Synapsentypen verwendet, die wie das Integrate-and-Fire Modell mit einem Leck-Integrator modelliert werden. Die Feeding-Synapsen dienen in der Regel als Empfangskanal externer Reize oder Pulse von Neuronen einer anderen Verarbeitungsschicht und werden zu einem gemeinsamen Dendritenpotential, dem Feeding-Potential *F(t)*, aufaddiert. Den zweiten Empfangskanal bilden die Linking-Synapsen, die benachbarte Neuronen derselben Verarbeitungsschicht mit dem Neuron verbinden. Die Linking-Synapsen werden ebenfalls zu einem Dendritenpotential aufsummiert, dem Linking-Potential *L(t)*. Die Berechnung des Membranpotentials *U(t)* erfolgt nach Gl. 2-12,

$$U(t) = F(t) \cdot [1 + L(t)] \quad | \; \tau_F > \tau_L \qquad \textbf{(2-12)}$$

in der man leicht erkennen kann, dass das Neuron nur durch ein existentes Feeding-Potential stimuliert werden kann. Das Linking-Potential hat bei Existenz aufgrund der Multiplikation in der Membranpotentialgleichung einen Modulationseffekt.

Synchronisation

Damit dieser Modulationseffekt wirksam wird, muss die in Gl. 2-12 angegebene Bedingung so erfüllt sein, dass *F(t)* aufgrund der größeren Zeitkonstante der Feeding-Synapsen τ_F langsamer als *L(t)* mit τ_L abklingt. Jedes Neuron ist mit benachbarten Neuronen derselben Verarbeitungsschicht, die sich innerhalb eines Linking-Feldes befinden, mit Linking-Synapsen verbunden. Eine oft verwendete Linking-Feld-Größe besteht aus 9×9 Neuronen, woraus sich 80 Linking-Synapsen pro Neuron ergeben [Stöc96] [Stöc97]. Das Auslösen eines Pulses führt zu einer Erhöhung der Linking-Potentiale der Neuronen innerhalb des Linking-Feldes. Aufgrund der kleineren Zeitkonstante der Linking-Synapsen τ_L klingen die Linking-Potentiale schnell ab, und kurz aufeinanderfolgende Pulse führen wieder zu

einer ähnlich starken Erhöhung dieser Potentiale. Werden nun Neuronen eines Linking-Feldes mit ähnlich starken Reizen über die Feeding-Synapsen stimuliert, führt der modulative Einfluss der Linking-Synapsen zu einer gegenseitigen Anhebung der Membranpotentiale und somit zu einem synchronen Feuern dieser Neuronengruppe. Dieser Synchronisationsmechanismus stellt die Fähigkeit dar, Neuronen aufgrund externer Reizsignale (in Form eines Eingangsbildes) zu gruppieren, und ist für die Objekt-Hintergrund-Trennung (*Figure-Ground-Separation*) in der Bildverarbeitung geeignet [Eckh99].

Explosiv pulsende Neuronen

Wie beim Spike-Response Modell verwendet das Marburger Modell eine dynamische Schwelle $\theta(t)$, die mit der Zeitkonstante τ_θ auf den statischen Schwellwert θ_0 abklingt (siehe Abb. 2-9a). Da das Membranpotential des Marburger Modells nicht wie beim Integrate-and-Fire Modell nach einem Puls zurückgesetzt wird, sondern in Abhängigkeit der Feeding- und Linking-Potentiale langsam abklingt, ist es möglich, dass das Neuron durch eine sehr schnell abklingende Schwelle ($\tau_F >> \tau_\theta$) wiederholt feuert. Mit diesem Mechanismus ist das Marburger Modell in der Lage, explosiv pulsende Neuronen (siehe auch Kapitel 2.1.3 auf Seite 14) zu modellieren [John99], solange sich das Membranpotential über dem statischen Schwellwert befindet.

2.2.4 Neuronenmodell mit adaptiven Synapsengewichten

Das Neuronenmodell mit adaptiven Synapsengewichten von Heittmann und Ramacher [Heit02] beschreibt das Membranpotential a_K wie beim Spike-Response Modell mit Hilfe eines Integrals. Die Membranpotentialgleichung ist in Gl. 2-13 angegeben,

$$a_K(t) = a_K(t_0) + \int_{t_0}^{t} \left[i_K + \sum_{i \in N_S} W_{KLi}(t) \right] dt \qquad | \; i_K = I_K \cdot W_0 \qquad \textbf{(2-13)}$$

wobei i_K einen eventuell vorhandenen externen Reizstrom darstellt, der sich aus dem Produkt von normierten Eingangsreizen I_K (Werte zwischen 0 und 1) und einem konstanten Gewichtswert W_0 ergibt. Mit W_{KLi} werden alle Synapsengewichte zu präsynaptischen Neuronen bezeichnet, über die im Zeitraum t_0 bis t ein Puls auf das zu betrachtende postsynaptische Neuron gesendet wird ($i \in N_S$).

Kein Leck-Widerstand

Wenn das Neuron den statischen Schwellwert θ erreicht, wird das Membranpotential wie beim Integrate-and-Fire Modell auf Null zurückgesetzt, und es wird ein Puls generiert. Während der Pulsdauer t_d, die typischerweise 1 ms beträgt, ist das Neuron absolut refraktär. Wie aus Gl. 2-13 ersichtlich bleibt a_K konstant, wenn kein externer Reiz vorliegt und das Neuron keinen Puls empfängt. Es handelt sich bei diesem Neuronenmodell somit um ein Integrate-and-Fire Modell ohne Leck-Widerstand (*Non-Leaky Integrate-and-Fire Model*), welcher sonst das Abklingen des Membranpotentials bewirkt, wenn das Neuron nicht stimuliert wird. Im Vergleich zum Integrate-and-Fire Modell mit Leck-Widerstand, bei dem für die Generierung eines Pulses das Produkt aus konstantem Eingangsstrom und Leck-Widerstand größer als der Schwellwert sein muss (siehe auch Gl. 2-8 auf Seite 17), feuert das Neuron ohne Leckwiderstand immer periodisch, wenn ein konstanter Eingangsreiz anliegt.

2 Quellen

Das Schaltbild des Neuronenmodells mit adaptiven Synapsen ist in Abb. 2-10 dargestellt. Man erkennt, dass 2 Quellen das Membranpotential $a_K(t)$ des postsynaptischen Neurons beeinflussen, solange es nicht selber einen Puls sendet ($X_K = 0$). Die erste Quelle sind Stromgrößen, wie der externe Reiz i_K oder präsynaptische Ströme I_{PRE}, die aus der Multiplikation von konstanten Synapsengewichten W_{KL} und Eingangspulsen entstehen ($X_L = 1$). Die zweite Quelle sind Spannungsgrößen von adaptiven Synapsengewichten $W_{KL}(t)$, die von dem postsynaptischen Membranpotential und dem Schwellwert oder von präsynaptischen Strömen abhängen.

Adaptive Synapsengewichte

Die neuronale Dynamik des Neurons basiert auf der Adaption der Synapsengewichte, die präsynaptische Neuronen mit dem postsynaptischen Neuron verbinden. Die verschiedenen Adaptionsregeln für die Synapsengewichte beeinflussen durch einen Verstärkungsterm den ansonsten exponentiell abklingenden Verlauf des Synapsengewichts,

$$W_{KL}(t) = W_{KL}(t_0) \cdot e^{-\gamma \cdot (t - t_0)} \quad \textbf{(2-14)}$$

wobei γ die Abklingkonstante des Gewichts darstellt. Die Bedingung für die Berücksichtigung des Verstärkungsterms ist, dass sich das präsynaptische Neuron im Sendezustand (X_L = 1) und das postsynaptische Neuron im Empfangszustand (X_K = 0) befinden müssen. Bei einem positiven Verstärkungsterm erfährt das Synapsengewicht einen verstärkenden Effekt (*Potentiative Effect*) und bei einem negativen Verstärkungsterm einen abschwächenden Effekt (*Depressive Effect*).

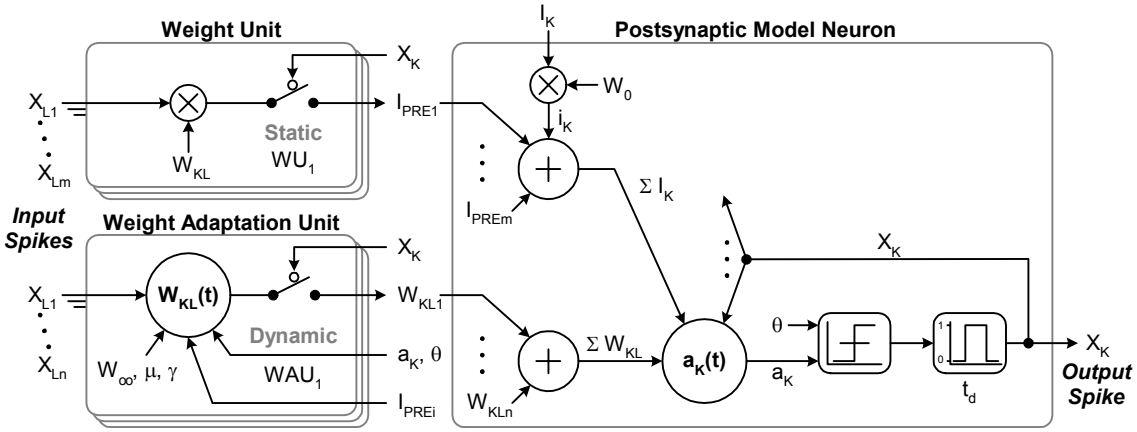


Abb. 2-10: Schaltbild des Neuronenmodells mit adaptiven Synapsengewichten

Fleck-Detektion

Bei der Verwendung folgender Adaptionsregel,

$$W_{KL}(t)' = -\gamma \cdot W_{KL}(t) + \begin{cases} \mu \cdot \left(a_K(t) - \frac{\theta}{2}\right) & X_K = 0 \wedge X_L = 1 \\ 0 & \text{sonst} \end{cases} \quad \textbf{(2-15)}$$

wobei μ den positiven Verstärkungsfaktor darstellt, kann durch eine Vierer-Nachbarschafts-Verbindung und vorhandene externe Eingangsreize ein synchrones Feuern von Neuronengruppen wie bei dem Marburger Modell erreicht werden (γ = 0,1 ms^{-1}; μ = 0,3 ms^{-2}; t_d = 1 ms; θ = 1). Ein verstärkender Effekt des Synapsengewichts findet statt, wenn das Membranpotential a_K des verbundenen postsynaptischen Neurons über dem halben Schwellwert liegt. Anhand dieser Adaptionsregel kann eine Detektion von Flecken durchgeführt werden (*Spot Detection*). Flecken sind Stellen einer visuellen Szene, die aufgrund eines ähnlichen Helligkeitswertes als zusammengehörig empfunden werden. Eine weiterführende Detektion ist die Detektion von Gradienten, die Stellen einer visuellen Szene mit großen Helligkeitsunterschieden offenbaren. Diese Gradienten-Detektion beruht auf Puls-Korrelations- und Puls-Dekorrelations-Detektoren, die im folgenden beschrieben werden.

2.2.4.1 Puls-Korrelations-Detektor

Der Puls-Korrelations-Detektor besteht aus 3 Neuronen (siehe Abb. 2-11a).

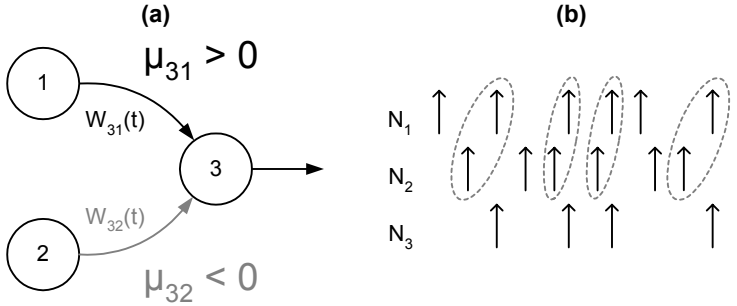


Abb. 2-11: (a) Verbindungstruktur des Puls-Korrelations-Detektors, (b) Pulsfolgen der 3 Neuronen

Die Neuronen N_1 und N_2 sind über adaptive Synapsen nach der Adaptionsregel in Gl. 2-15 mit dem Neuron N_3 verbunden. Das Neuron N_3 detektiert, ob die Pulsfolgen der Neuronen N_1 und N_2 korreliert sind, indem es immer einen Puls sendet, wenn es ein relevantes Puls-Paar erkennt. Als relevantes Puls-Paar werden 2 aufeinanderfolgende Pulse bezeichnet, wobei der erste Puls von N_2 und der zweite Puls von N_1 stammen müssen (siehe Abb. 2-11b). Nachdem N_3 gepulst hat, müssen der erste und alle weiteren Pulse von N_2 das postsynaptische Neuron N_3 in einen hoch erregten Zustand versetzen, der durch die Bedingung $0{,}5\cdot\theta << a_K < \theta$ gegeben ist. Dies kann durch einen negativen Verstärkungsfaktor μ der Adaptionsregel in Gl. 2-15 erreicht werden. Dieser bewirkt, dass die Synapse verstärkt wird, wenn das Membranpotential Null ist.

$$W_{32}(t)' = -\gamma \cdot W_{32}(t) - |\mu| \cdot \left(a_3(t) - \frac{\theta}{2}\right) \qquad \textbf{(2-16)}$$

Die zeitliche Änderung des Membranpotentials $a_3(t)$ von N_3 ist bei einem Puls von N_2 nach Gl. 2-13 nur das Synapsengewicht:

$$a_3(t)' = W_{32}(t) \qquad \textbf{(2-17)}$$

Durch Einsetzen von Gl. 2-17 in Gl. 2-16 erhält man eine Differentialgleichung 2. Ordnung für a_3 und nach entsprechender Lösung (siehe Anhang C auf Seite 168) folgenden Ausdruck für das Membranpotential [Heit03]:

$$a_3(t) = \frac{\theta}{2} \cdot \left(1 - e^{-\frac{\gamma}{2}\cdot t} \cdot \left(\frac{\gamma}{2\cdot\omega_0} \cdot \sin(\omega_0 \cdot t) + \cos(\omega_0 \cdot t)\right)\right) \quad | \ \omega_0 = \sqrt{\mu - \frac{\gamma^2}{4}} \qquad \textbf{(2-18)}$$

Durch eine geeignete Wahl von γ und μ wird durch den ersten Puls von N_2 das Membranpotential nach der Pulsdauer soweit angehoben, dass die Bedingung $0{,}5\cdot\theta < a_3(t_d) < \theta$ erfüllt ist. Weitere Pulse von N_2 haben dann eine unterdrückende Wirkung auf das Synapsengewicht, da durch das negative μ der Verstärkungsterm in Gl. 2-15 insgesamt negativ ist, und können N_3 nicht zum feuern bringen. Ein Puls von N_1 allerdings bringt bei entsprechend großem μ N_3 zum feuern, wenn es sich durch einen oder mehrere vorangegangene Pulse von N_2 in einem hoch erregten Zustand befindet.

2.2.4.2 Puls-Dekorrelations-Detektor

Der Puls-Dekorrelations-Detektor besteht aus 6 Neuronen (siehe Abb. 2-12a). Die Neuronen N_1 und N_2 bilden jeweils mit den Neuronen N_{3+} und N_{3-} 2 Puls-Korrelations-Detektoren. Die Neuronen N_{4+} und N_{4-} detektieren die nicht korrelierten Pulse von N_1 bzw. von N_2. Die Pulse, die von den Korrelations-Neuronen N_{3+} und N_{3-} nicht als Puls-Paar erkannt werden, werden von den Dekorrelations-Neuronen N_{4+} und N_{4-} weiterverarbeitet. Aus Abb. 2-12b wird deutlich, dass für die Detektion von dekorrelierten Pulsen das zu detektierende Neuron eine höhere Pulsfrequenz ν haben muss, damit das entsprechende Dekorrelations-Neuron pulst. Das Neuron N_{4+} kann somit nur pulsen wenn $\nu_{N1} > \nu_{N2}$ gilt, und entsprechend wird für die Pulsaktivität von N_{4-}, vorausgesetzt, dass die Bedingung $\nu_{N1} < \nu_{N2}$ für die Pulsfrequenzen gelten muss.

Die Detektion von unkorrelierten Pulsen erfolgt anhand einer Dendritenstruktur, bei der mehrere präsynaptische Neuronen durch einen präsynaptischen Strom I_{PRE} Einfluss auf das adaptive Synapsengewicht haben. Die Adaptionsregel, die diese Dendritenstruktur berücksichtigt und beim Puls-Dekorrelations-Detektor von den adaptiven Synapsen $W_{41}(t)$ und $W_{42}(t)$ in Abb. 2-12a verwendet wird, lautet [Heit03]:

$$W_{KL}(t)' = -\gamma \cdot (W_{KL}(t) - W_\infty) - \begin{cases} \mu \cdot (I_{PRE,K}(t) - I_\theta) \cdot W_{KL}(t) & X_K = 0 \wedge X_L = 1 \\ 0 & \text{sonst} \end{cases} \qquad \textbf{(2-19)}$$

Die positive Konstante W_∞ stellt einen statischen Gewichtswert dar, gegen den die Synapse konvergiert, und I_θ ist ein positiver Schwellwertstrom. Der präsynaptische Strom $I_{PRE,K}$ setzt sich aus der Summe aller präsynaptischen Pulse zusammen, die jeweils mit einem statischen Synapsenwert gewichtet werden:

$$I_{PRE,K} = \sum_{i \in N_S} I_{PRE,Li} = \sum_{i \in N_S} (W_{KLi} \cdot X_{Li}) \quad | \; I_{PRE,Li} > I_\theta \qquad \textbf{(2-20)}$$

Entsprechend gilt für den Puls-Dekorrelations-Detektor, dass die beiden präsynaptischen Ströme $I_{PRE,4+} = W_{42} \cdot X_2 + W_{43+} \cdot X_{3+}$ und $I_{PRE,4-} = W_{41} \cdot X_1 + W_{43-} \cdot X_{3-}$ sind.

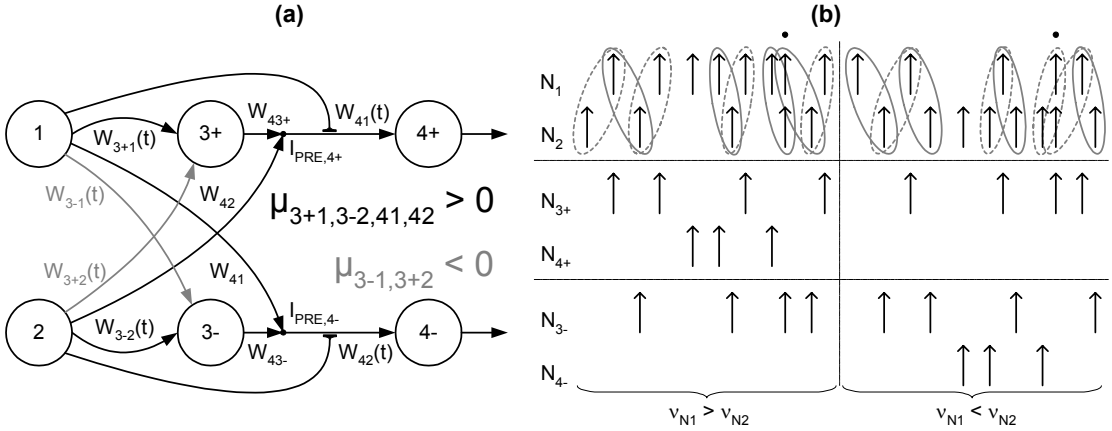


Abb. 2-12: (a) Verbindungstruktur des Puls-Dekorrelations-Detektors, (b) Pulsfolgen der 6 Neuronen

Präsynaptischer Strom

Wenn beispielsweise N_1 pulst und keine präsynaptische Pulsaktivität von N_2 und N_{3+} vorliegt ($I_{PRE,4+} = 0$), ist der präsynaptische Gesamtstrom ($I_{PRE,K} - I_\theta$) in Gl. 2-19 negativ, was einen positiven Verstärkungsterm zur Folge hat. Die Synapse $W_{41}(t)$ wird verstärkt und bringt N_{4+} zum Pulsen. Liegt eine präsynaptische Pulsaktivität von N_2 oder N_{3+} vor ($I_{PRE,4+} > I_\theta$) ist der präsynaptische Gesamtstrom immer positiv, was einen negativen Verstärkungsterm zur Folge hat. Die Synapse wird abgeschwächt, damit N_{4+} nicht zum Pulsen angeregt wird. Die Zusammensetzung von $I_{PRE,4+}$ aus den Pulsen der Neuronen N_2 und N_{3+} ist notwendig, wenn N_1 und N_2 voll synchron pulsen (siehe schwarze Punkte in Abb. 2-12b). Damit verhindert der von N_2 ausgelöste präsynaptische Strom, dass N_{4+} fälschlicherweise einen unkorrelierten Puls von N_1 detektiert.

Nachdem nun die Pulsaktivität von Neuronen als Einzelkomponenten betrachtet wurden, wird nun auf die Sichtweise übergeleitet, wie Neuronen im menschlichen visuellen System eingebettet sind, um Bilddaten effizient anhand von Pulsen zu verarbeiten.

2.3 Das menschliche visuelle System

Das menschliche visuelle System beschreibt den menschlichen Sehvorgang, der bei den visuellen Feldern beginnt und über das Auge, die Retina und die Sehbahnen im Gehirn vom striären visuellen Cortex ausgehend in extrastriären visuellen Arealen hierarchisch weiterverarbeitet wird. Die Komplexität der Pulsverarbeitung im visuellen System ist noch immer nicht vollständig aufgeklärt, aber trotzdem gehört das visuelle System zu den am intensivsten untersuchten sensorischen Systemen des Menschen [Funk02].

Visuelle Felder: Gesichts-, Blick- und Ganzfeld

Das Gesichtsfeld bezeichnet das visuelle Feld, welches der Mensch ohne jegliche Augen-, Kopf- und Körperbewegung erfasst. Man unterscheidet das einäugige Gesichtsfeld (monokular) und das zweiäugige Gesichtsfeld (binokular). Das monokulare Gesichtsfeld wird in 2 Gesichtsfelder unterteilt, die sich aufgrund des Fixationspunktes ergeben. Das nasale Gesichtsfeld deckt das innere visuelle Feld (der Nase zugewandt) ab, und das temporale Gesichtsfeld bildet das äußere visuelle Feld. Bei dem binokularen Gesichtsfeld unterscheidet man das linke Gesichtsfeld, zusammengesetzt aus temporalem Gesichtsfeld des linken und nasalem Gesichtsfeld des rechten Auges, und das rechte Gesichtsfeld, welches das nasale Gesichtsfeld des linken und das temporale Gesichtsfeld des rechten Auges vereint. Das Blickfeld ist das visuelle Feld, welches das Auge einbeziehen kann, ohne den Kopf und Körper zu bewegen. Es ist um 120° in vertikaler Richtung und um 80° in horizontaler Richtung größer als das Gesichtsfeld [Birb02]. Ein visuelles Feld besteht in der Regel aus unterschiedlichen Formen und Konturen, die die Rezeptoren in der Netz-

haut (*Retina*) des Auges ständig reizen. Ein visuelles Feld ohne jegliche Konturen wird als Ganzfeld bezeichnet und lässt nach längerer Betrachtung die Rezeptoren ausbleichen.

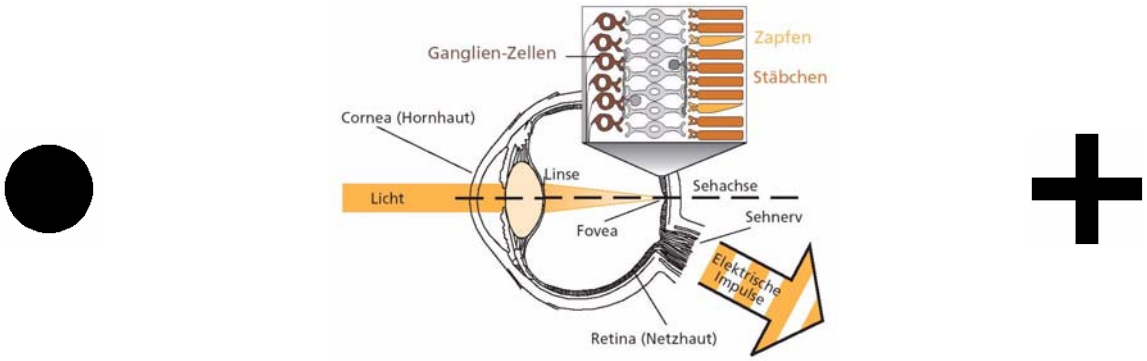


Abb. 2-13: Querschnitt des Auges (modifiziert von [Marc93])

Auge

Das Auge ist die Kontaktstelle von Umwelt und visuellem System und stellt ein zusammengesetztes optisches System dar, bestehend aus Hornhaut (*Cornea*), Linse und vorderer Augenkammer, die sich zwischen Cornea und Linse befindet (siehe Abb. 2-13). Dieses optische System entwirft auf der Retina ein umgekehrtes und stark verkleinertes Bild der Umwelt [Birb02]. Die Achse des optischen Systems (Sehachse) trifft bei normaler Ruhestellung auf die schärfste Stelle des Sehens, die Sehgrube (*Fovea*).

Das Auge besitzt die Fähigkeit durch Anpassung der Brechkraft der Linse, Gegenstände in der Ferne als auch in der Nähe scharfzustellen (Akkomodation). Durch Verringerung der Brechkraft entsteht eine flache Linse (Ellipsenform), um entfernte Gegenstände (Ferneinstellung) zu fokussieren. Durch Erhöhung der Brechkraft kommt es zu einer dicken Linse (Kugelform), um nahe Gegenstände (Naheinstellung) zu fokussieren. Liegt bei Fixation ferner Gegenstände der Brennpunkt innerhalb des Glaskörpers vor der Retina, liegt eine Myopie (Kurzsichtigkeit) vor. Fällt bei Fixation naher Gegenstände der Brennpunkt “hinter” die Retina, spricht man von Hyperopie (Weitsichtigkeit).

Retina

Die Retina ist der eigentliche Sensor beim Sehvorgang und besteht aus 3 Schichten (siehe Abb. 2-13). In der ersten Schicht befinden sich die Photorezeptoren. Man unterscheidet 2 Arten von Photorezeptoren: Stäbchen (*Rods*) und Zapfen (*Cones*). Im hellen Licht benutzt das visuelle System die $6 \cdot 10^6$ Zapfen (photopisches Sehen in Helligkeit) und unter schwacher Beleuchtung kommen die $120 \cdot 10^6$ Stäbchen zum Einsatz (skotopisches Sehen in der Dämmerung). Innerhalb der Retina ist nur in der Fovea die Konzentration von Zapfen größer als von Stäbchen. In der zweiten Schicht liegen die Bipolar-, Horizontal- und Amakrinzellen. Die Bipolarzellen sind für den neuronalen Signalfluss in vertikaler Richtung von den Photorezeptoren zu den in der dritten Schicht liegenden Ganglienzellen zuständig. Die Horizontalzellen sorgen für einen neuronalen Signalfluss in horizontaler Richtung (laterale Verbindungen) auf der Seite der Photorezeptoren und die Amakrinzellen auf der Seite der Ganglienzellen. Die 10^6 Ganglienzellen in der Retina integrieren die neuronalen Signale der Bipolar- und Amakrinzellen, um diese als verschlüsselte Information in Form von Pulsmustern an das Gehirn weiterzuleiten. Man unterscheidet 2 wesentliche Arten von Ganglienzellen. Zum einen die M-Zellen (magnozellulären Zellen), die etwa 10 % aller Ganglienzellen ausmachen und für die Bewegungsentdeckung zuständig sind, und zum anderen die P-Zellen (parvozellulären Zellen), die ungefähr 80 % aller Ganglienzellen bilden und für die Entdeckung stationärer Muster, wie Helligkeit und Farbe, verantwortlich sind [Kand00].

Eine Besonderheit der Retina ist, dass nur die Photorezeptoren in der Fovea unmittelbar vom Licht getroffen werden, ansonsten durchläuft das Licht erst einmal mühelos Ganglien-, Amakrin-, Bipolar- und Horizontalzellen, bevor es auf die Phototrezeptoren trifft. Dadurch ist der direkte Weg von den Ganglienzellen aus dem Auge heraus blockiert, und ein kleiner rezeptorenloser Bereich bleibt frei, durch den die Axone der Ganglienzellen das Auge als Sehnerv (*Nervus Opticus*) verlassen (siehe Abb. 2-13). Dieser rezeptorenlose Bereich wird als “Blinder Fleck” bezeichnet, da Abbildungen in diesem Bereich nicht gese-

hen werden. Dieser Bereich kann bewusst wahrgenommen werden, indem man das rechte Auge schließt und mit dem linken Auge das Kreuz in Abb. 2-13 auf der rechten Seite fixiert. Durch Änderung des Abstands von Auge zu Papier verschwindet der Kreis auf der linken Seite, wenn er genau auf dem rezeptorenlosen Bereich der Retina des linken Auges liegt.

Sehbahnen

Die Sehnerven beider Augen bilden den Anfang der Sehbahnen und laufen im weiteren Verlauf aufeinander zu und kreuzen sich in der Sehkreuzung (*Chiasma Opticum*), so dass die Sehnerven der linken Gesichtshälfte (linkes Gesichtsfeld) zur rechten Gehirnhälfte und die der rechten Gesichtshälfte (rechtes Gesichtsfeld) zur linken Gehirnhälfte verlaufen (siehe Abb. 2-14). Als Sehstränge (*Tracticus Opticus*) treffen dann 90 % der Sehnerven auf das jeweilige primäre Sehzentrum einer Gehirnhälfte, welches auch als lateraler Kniehöcker (*CGL = Corpus Geniculatum Laterale*) bezeichnet wird. Das CGL besitzt 2 Arten von Zellschichten, die sich aufgrund der 2 Hauptarten von Ganglienzellen ergeben: die magnozellulären und die parvozellulären Schichten. Des Weiteren hat das CGL einen zweiten eingehenden Signalfluss vom Cortex, der größer ist als der eingehende Signalfluss in Form von Sehsträngen. Den dritten und kleinsten Signalfluss stellt die Verbindung vom CGL zum Cortex dar [Gold02]. Diese Verbindungen werden als Sehstrahlungen (*Radiatio Optica*) bezeichnet und bilden den letzten Teil der Sehbahnen.

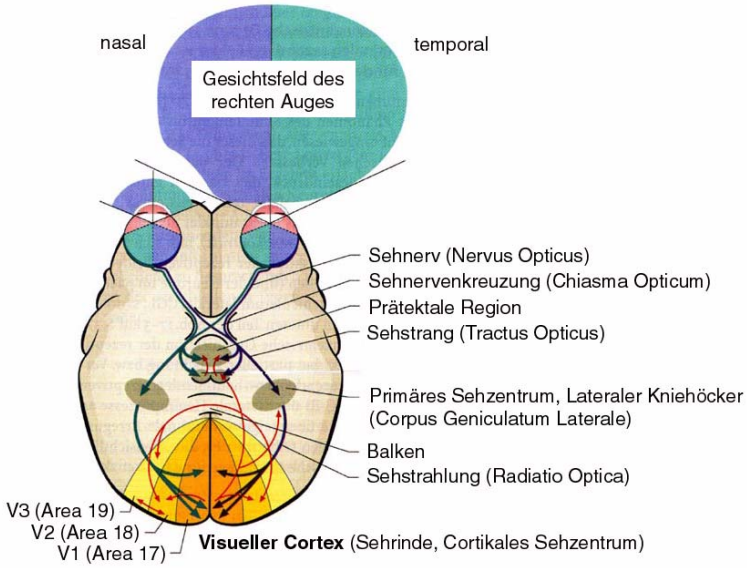


Abb. 2-14: Verlauf der Sehbahnen im menschlichen Gehirn (modifiziert von [Birb02])

Visueller Cortex

Die Sehbahnen enden in der Sehrinde, die auch als cortikales Sehzentrum oder visueller Cortex bezeichnet wird. Der visuelle Cortex empfängt sämtliche visuellen Informationen vom jeweiligen CGL und wird als das Gehirnareal V1 benannt, weil es die erste visuelle Verarbeitungsschicht im Gehirn darstellt. Das Areal V1 des Menschen besteht aus 6 unterscheidbaren Schichten, wobei die Schicht 4, die wiederum in 4 Unterschichten unterteilt ist (4A, 4B, 4Cα und 4Cβ), die meisten Eingangssignale vom CGL empfängt [Kand00]. Die Trennung von magno- und parvozellulären Schichten vom CGL wird im V1 weiter beibehalten, da die meisten Verbindungen aus der magnozellulären Schicht im CGL in die Schicht 4Cα und die meisten Verbindungen aus der parvozellulären Schicht im CGL in die Schicht 4Cβ projiziert werden. Vom Areal V1 aus werden die visuellen Informationen schließlich an die extrastriären visuellen Areale V2, V3 und V4 weitergeleitet. Das Areal V2 übernimmt dabei die Aufgabe der visuellen Gestalterkennung. Die Neuronen dieses Areals reagieren auf Scheinkonturen, die in Wirklichkeit nicht vorhanden sind, aber wahrgenommen werden. Das Areal V3 hat die Aufgabe der Gestalterkennung bewegter Objekte. Die Neuronen dieses Areals sind sensitiv auf bewegte Konturen. Das Areal V4 erfüllt die Aufgabe der Objekterkennung aufgrund von Oberflächenfarben und Farbkontrasten. Die Neuronen dieses Areals sind farbspezifisch organisiert. Eine normale visuelle Wahrnehmung ist nur bei ungestörter und koordinierter Aktivität aller dieser 3 Areale möglich [Birb02].

2 Visuelle Pfade

Bei der Pulsverarbeitung im visuellen System findet eine getrennte Informationsverarbeitung statt, die zum einen visuelle Informationen über Bewegung eines Objekts und zum anderen visuelle Informationen über Farbe und Form eines Objektes verarbeiten. Man unterscheidet 2 parallel verlaufende visuelle Pfade: den dorsalen und den ventralen Pfad (siehe Abb. 2-15).

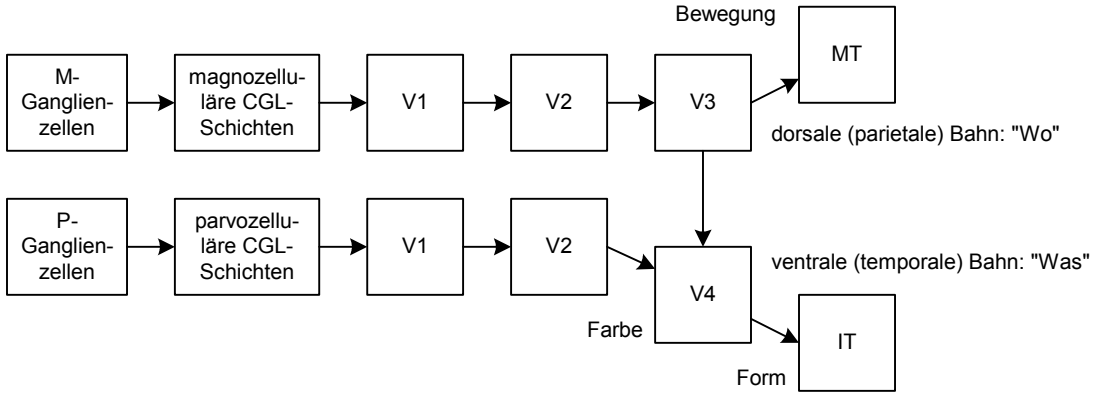


Abb. 2-15: Vereinfachtes Schema des Signalflusses der visuellen Verarbeitung (basierend auf [Gold02])

Der dorsale Pfad, auf dem die visuellen “Wo”-Informationen eines Objekts verarbeitet werden, beginnt bei den M-Ganglienzellen und verläuft bis zum MT-Areal (MT = mediotemporal) Areal im Gehirn. Im MT-Areal reagieren die Neuronen auf unterschiedliche Richtungen von Bewegungsmustern. Der ventrale Pfad, auf dem die visuellen “Was”-Informationen eines Objekts codiert werden, wird von den P-Ganglienzellen eingeleitet und führt bis zum IT-Areal (IT = inferotemporal) im Gehirn, wo Neuronen auf komplexe Formen von Objekten antworten. Innerhalb des IT-Areals befindet sich außerdem ein Gehirnareal, welches als Gyrus fusiformis bekannt ist. Untersuchungen mit Hilfe der fMR-Tomographie (fMR = funktionelle Magnetresonanz) haben gezeigt, dass die Neuronen in diesem Areal auf die Analyse von Gesichtern spezialisiert sind [Gold02]. Dieses Gehirnareal kann aber auch durch Training auf andere komplexe Objekte reagieren. Somit reagieren die Neuronen im Gyrus fusiformis nicht nur auf Gesichter, sondern auch auf visuelle Objekte, die man sich durch Expertenwissen angeeignet hat (z. B. Ingenieur erkennt speziellen Schaltkreis, Zoologe erkennt seltene Tierart, Maler erkennt einzigartiges Gemälde, etc.).

Nachdem nun der Signalfluss der Pulse im menschlichen visuellen System betrachtet wurde, wird nun die Pulsverarbeitung in der Retina und im visuellen Cortex beschrieben, die sich aufgrund der unterschiedlichen rezeptiven Felder voneinander unterscheidet.

2.3.1 Neuronale Pulsverarbeitung mit rezeptiven Feldern

Als RF (rezeptives Feld) eines Neurons im visuellen System wird jener Teilbereich der Retina bezeichnet (Netzhautareal), der durch Reizung dieses Neuron erregt. In der Fovea sind die RFs am kleinsten und zur Netzhautperipherie hin nimmt die Größe der Felder kontinuierlich zu. Anhand eines RFs werden im visuellen Feld Formen erkannt, die durch Konturen getrennt bzw. eingegrenzt sind. Konturen sind Stellen im retinalen Abbild, an denen sich die Intensität oder Zusammensetzung der Wellenlänge abrupt ändert [Gold02]. Innerhalb des menschlichen visuellen Systems unterscheidet man retinale RFs, die Ganglienzellen in der Retina und Neuronen im CGL (CGL-Zellen) erregen, und cortikale RFs, die Neuronen im visuellen Cortex erregen.

2.3.1.1 Retinale rezeptive Felder

On-Zentrum-/ Off-Zentrum-Zelle

Die retinalen rezeptiven Felder haben 2 Aktivitätsbereiche: ein Zentrum und ein Umfeld. Diese Aktivitätsbereiche sind somit konzentrisch angeordnet und haben eine gegensätzliche Wirkung (antagonistisch). Wenn das Zentrum eine exzitatorische und das Umfeld eine inhibitorische Wirkung auf helle visuelle Reize hat, handelt es sich bei der Ganglienzelle

oder CGL-Zelle um eine On-Zentrum-Zelle (*On-Center Cell*). Hat das Zentrum des RFs eine exzitatorische und das Umfeld eine inhibitorische Wirkung auf dunkle visuelle Reize, spricht man bei der Ganglienzelle oder CGL-Zelle von einer Off-Zentrum-Zelle (*Off-Center Cell*).

Die räumliche Struktur von retinalen RFs lässt sich anhand der Differenz von 2 Gauss-Funktionen durch die Funktion $D_R(x,y)$ beschreiben [Daya01],

$$D_R(x,y) = \pm\left(\frac{1}{2\cdot\pi\cdot\sigma_{cen}^2}\cdot e^{-\left(\frac{x^2+y^2}{2\cdot\sigma_{cen}^2}\right)} - \frac{B}{2\cdot\pi\cdot\sigma_{sur}^2}\cdot e^{-\left(\frac{x^2+y^2}{2\cdot\sigma_{sur}^2}\right)}\right) \qquad \textbf{(2-21)}$$

wobei σ_{cen} die Größe des Zentrums und σ_{sur} die Größe des Umfelds beschreibt. Der Parameter B steuert die Balance zwischen Zentrum und Umfeld und das positive Vorzeichen von $D_R(x,y)$ erlaubt die Darstellung einer On-Zentrum-Zelle bzw. einer Off-Zentrum-Zelle bei einem negativen Vorzeichen. Aufgrund der resultierenden Form in Abb. 2-16 wird $D_R(x,y)$ auch als “Mexican Hat”-Funktion bezeichnet [Hayk99].

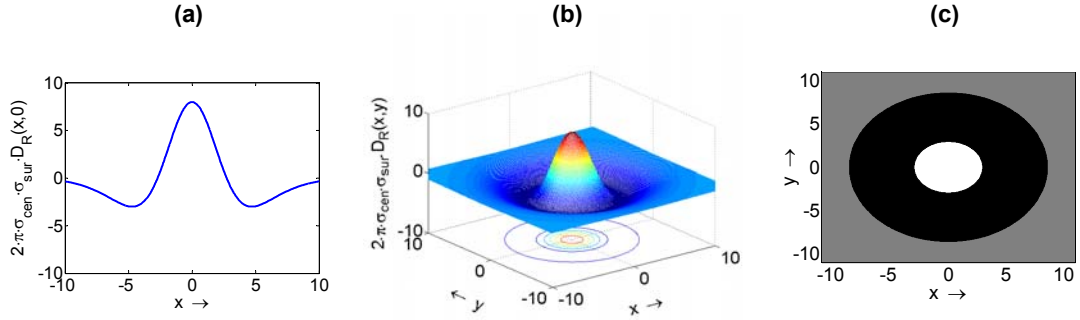


Abb. 2-16: Retinale rezeptive Feldstruktur einer On-Zentrum-Zelle: (a) zweidimensionale Darstellung, (b) räumliche Darstellung, (c) Aktivitätsbereiche (ON = weiss, OFF = schwarz) [B = σ_{cen} = 2, σ_{sur} = 4]

2.3.1.2 Cortikale rezeptive Felder

Die cortikalen rezeptiven Felder sind im Gegensatz zu retinalen RFs parallel angeordnet und können aus mehr als 2 Aktivitätsbereichen bestehen. Es gibt 2 wesentliche Zelltypen im visuellen Cortex, die durch Experimente von Hubel und Wiesel [Hube62] entdeckt wurden: die Einfache Zelle (*Simple Cell*) und die Komplexe Zelle (*Complex Cell*).

Einfache Zelle

Einfache Zellen reagieren auf visuelle Reizmuster mit einer bevorzugten Position, Form und Orientierung und werden durch mehrere CGL-Zellen stimuliert. Die RFs für Einfache Zellen setzen sich aus mehreren retinalen RFs zusammen. Die räumliche Struktur von cortikalen RFs einer Einfachen Zelle lässt sich anhand der Überlagerung von 2 Exponential-Funktionen durch die Funktion $D_C(x,y)$ beschreiben [Daya01],

$$D_C(x,y) = \pm\frac{1}{2\cdot\pi\cdot\sigma_x\cdot\sigma_y}\cdot e^{-\left(\frac{x^2}{2\cdot\sigma_x^2}+\frac{y^2}{2\cdot\sigma_y^2}\right)}\cdot\cos(k\cdot x-\varphi) \qquad \textbf{(2-22)}$$

Die Variablen σ_x und σ_y bestimmen die Ausbreitung in x- bzw. y-Richtung, k stellt die bevorzugte Raumfrequenz des RFs dar, und φ repräsentiert die bevorzugte räumliche Phase.

Anhand einer Koordinatentransformation nach Gl. 2-23,

$$x \to x\cdot\cos(\theta) - y\cdot\sin(\theta), \qquad y \to x\cdot\cos(\theta) + x\cdot\sin(\theta) \qquad \textbf{(2-23)}$$

wobei θ der bevorzugte Orientierungswinkel der Einfachen Zelle ist, lässt sich die Funktion $D_C(x,y)$ graphisch darstellen. In Abb. 2-17 ist die rezeptive Feldstruktur einer Einfachen Zelle mit 2 und in Abb. 2-18 mit 3 Aktivitätsbereichen dargestellt.

Gabor-Funktion

Die Funktion $D_C(x,y)$ wird auch als Gabor-Funktion oder Gabor-Wavelet bezeichnet [Daya01]. Gabor-Wavelets besitzten wie Sinus- und Cosinus-Funktionen einen begrenzten Definitionsbereich im Frequenzspektrum und sind zusätzlich im Zeitbereich begrenzt.

Daher haben Gabor-Wavelets die Form von nach außen hin auslaufenden (kleiner werdenden) Wellen.

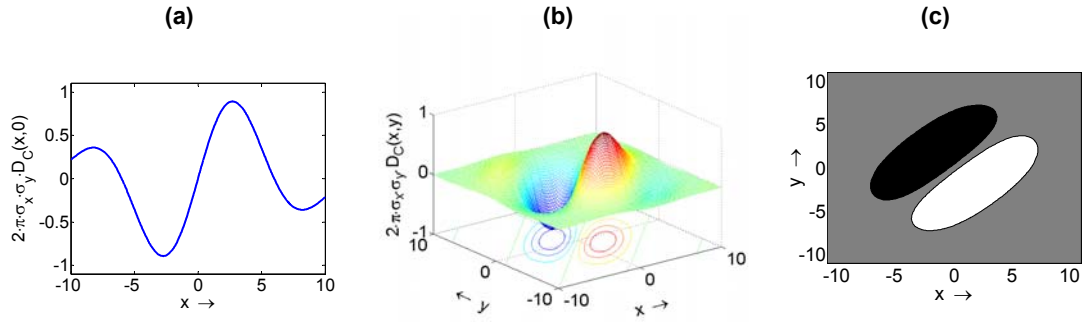


Abb. 2-17: Cortikale rezeptive Feldstruktur einer Einfachen Zelle mit 2 Aktivitätsbereichen: (a) zweidimensionale Darstellung, (b) räumliche Darstellung, (c) 2 Aktivitätsbereiche (OFF-ON, schwarz-weiss) [$\sigma_x = \sigma_y = 4$, $1° = 2\cdot\pi/360$, k = 30°, θ =45°, ϕ = 90°]

Komplexe Zelle

Komplexe Zellen werden durch mehrere Einfache Zellen stimuliert und sind für die Detektion von Konturunterbrechungen zuständig. Die RFs Komplexer Zellen sind in der Regel größer, da sie sich aus mehreren RFs Einfacher Zellen zusammensetzen. Im Vergleich zu Einfachen Zellen reagieren Komplexe Zellen effektiver auf Bewegung und genauer auf die Orientierung eines Reizmusters. Unter den Komplexen Zellen unterscheidet man noch die Hyperkomplexen Zellen, die auch als endinhibierte Zellen bezeichnet werden. Diese Zellen reagieren nur auf Reizmuster, die die Größe der Aktivitätsbereiche innerhalb des RFs nicht überschreiten (*End-Stopping*) [Hube68].

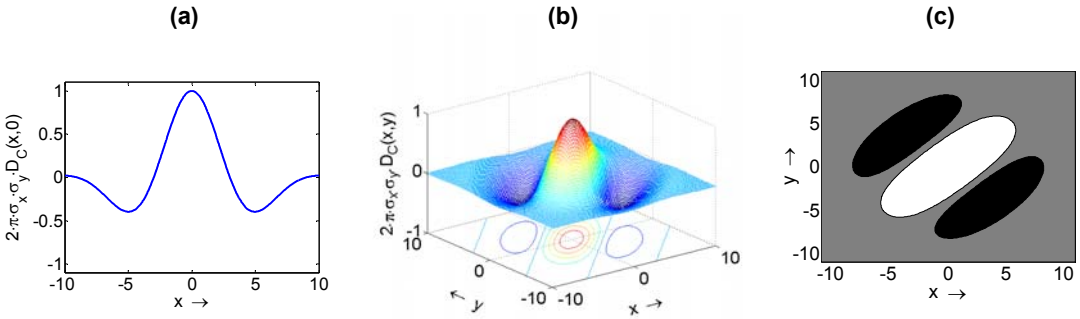


Abb. 2-18: Cortikale rezeptive Feldstruktur einer Einfachen Zelle mit 3 Aktivitätsbereichen: (a) zweidimensionale Darstellung, (b) räumliche Darstellung, (c) 3 Aktivitätsbereiche (OFF-ON-OFF, schwarz-weiss-schwarz) [$\sigma_x = \sigma_y = 4$, $1° = 2\cdot\pi/360$, k = 30°, θ =45°, ϕ = 0°]

Körperzentrierte RFs

Nicht alle rezeptiven Felder von Neuronen lassen sich in retinalen Koordinaten angeben. Dies gilt insbesondere für Neuronen, die auf 2 (bimodal) oder mehrere Sinne (multimodal) reagieren. Experimente haben gezeigt [Graz95], dass rezeptive Felder von bimodalen Neuronen, die auf taktile Reize (Berührung) der Hand und auf visuelle Reize um die Hand herum antworten, in Bezug auf Körperkoordinaten definiert werden. Wenn die Hand innerhalb des Gesichtsfeldes liegt, reagieren die Neuronen sowohl auf taktile als auch auf visuelle Reize, ansonsten lediglich auf taktile Reize. Da das rezeptive Feld mit Bezug auf die Hand definiert wird, handelt es sich hierbei um körperzentrierte Neuronen [Gold02].

Nachdem die Funktionsweise des menschlichen visuellen Systems beleuchtet wurde, wird nun die biologisch motivierte Bildverarbeitung betrachtet.

2.4 Biologisch motivierte Bildverarbeitung

Das menschliche visuelle System besitzt überragende Bilderkennungsfähigkeiten, die von heutigen technischen Systemen immer noch unzureichend gelöst werden, wie z. B. die 3D-Darstellung von speziellen Objekten, die Objekttrennung oder die Formbestimmung von verdeckten Objekten [Gold02]. Eine biologisch motivierte Bildverarbeitung versucht anhand einer Informationsverarbeitung mit Pulsen und den rezeptiven

Eigenschaften von retinalen und cortikalen Neuronen, die Fähigkeiten des menschlichen visuellen Systems für zukünftige technische Systeme umzusetzen. Einige Bildvorverarbeitungsansätze mit pulscodierten Neuronen versprechen eine robuste und effiziente Extraktion von orientierten Konturmerkmalen [John94] [Lind98b] [Stöc97] [Weit97]. Die besonderen Fähigkeiten sowie die Grenzen der biologischen Bildverarbeitung des Menschen werden nun kurz skizziert.

Invarianz

Eine herausragende Fähigkeit der biologischen Bildverarbeitung besteht darin, dass Gegenstände in verschiedenen Entfernungen, an verschiedenen Orten des visuellen Feldes oder aus verschiedenen Blickrichtungen erkannt werden. Neben Neuronen, die nur auf eine bestimmte Größe, Lage und Form eines Gegenstands reagieren und somit größen-, orts-, und sicht-spezifisch sind, gibt es im IT-Areal ebenfalls Neuronen, die unabhängig von der Größe oder Position der retinalen Abbildung eines Gegenstands antworten (größen- und orts-invariant) oder die unabhängig vom Blickwinkel feuern, unter dem das Objekt gesehen wird (sicht-invariant) [Gold02].

Mehrdeutigkeit und Ergänzung

Eine weitere überragende Fähigkeit ist zum einen die mehrdeutige Interpretation und zum anderen die zuverlässige Interpretation von visuellen Objekten aus wenigen Hinweisen (Ergänzung). In Abb. 2-19a und Abb. 2-19b sind visuelle Objekte dargestellt, die eine mehrdeutige Interpretation erlauben, wobei Kanisza-Figuren (siehe Abb. 2-19c) verdeutlichen, dass wenige Hinweise ausreichen, um ein Objekt aufgrund von Erfahrungen zu erkennen.

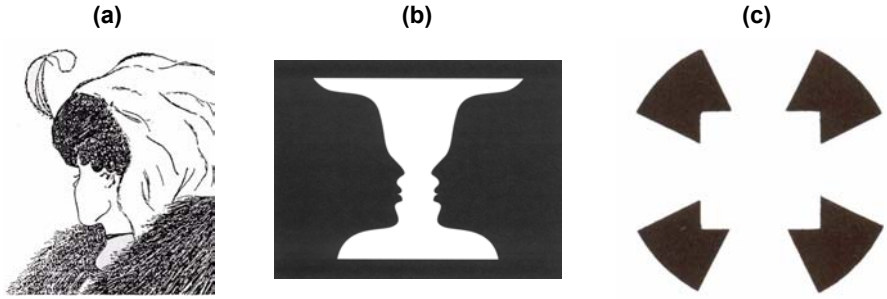


Abb. 2-19: Sinnestäuschung beruhend auf Mehrdeutigkeit und Gestaltergänzung: (a) alte/junge Frau [Attn76], (b) Rubin-Vase [Gold02], (c) Kanisza-Figur [Birb02]

Parallele und serielle Suche

Die beeindruckendste Fähigkeit ist jedoch die menschliche Verarbeitungsgeschwindigkeit bei der Erkennung bestimmter visueller Objekte. Dieser Vorgang wird bei der visuellen Suche deutlich, wo man zwischen paralleler und serieller Suche unterscheidet.

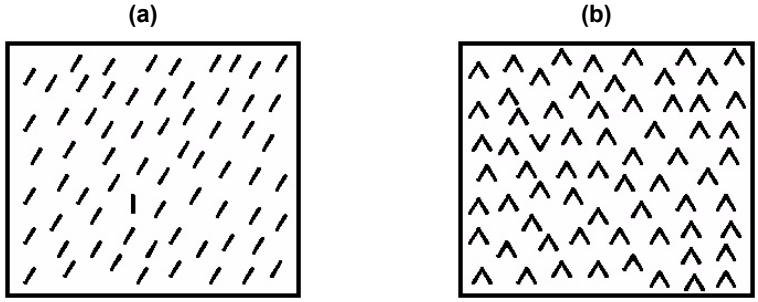


Abb. 2-20: Visuelle Suche: (a) parallele Suche, (b) serielle Suche

In Abb. 2-20a ist ein Beispiel für die parallele Suche illustriert. Die menschliche Bildverarbeitung kann sofort und mit einer unglaublichen Leichtigkeit erkennen, dass genau ein Balken vertikal angeordnet ist. Die erste Grenze der biologischen Bildverarbeitung wird anhand des Beispiels der seriellen Suche in Abb. 2-20b sichtbar. Man erkennt erst nach einem sequentiellen visuellen Abtasten des Bildes, dass eine Dachspitze nach unten zeigt. Das Erkennen von Details bei einer nicht vertrauten visuellen Umgebung resultiert in einer seriellen Suche und kann unter Umständen zu einem sehr langen Bildverarbeitungsprozess führen.

Aufmerksamkeit

Eine Besonderheit der menschlichen Bildverarbeitung, die weitere Grenzen aufzeigt, ist die aktive Suche von visuellen Informationen, die wichtig erscheinen. Diese aktive Suche ist die Aufmerksamkeit, die vom Gehirn aufgrund von Erfahrungen oder auch von visuellen Reizen selbst, wie in Abb. 2-20a durch den vertikalen Balken, gesteuert wird. In manchen Situation werden visuelle Reizmuster nicht gesehen, da ihnen keine Aufmerksamkeit geschenkt wird. Dies führt zur Blindheit durch Nichtaufmerksamkeit (*Inattentional Blindness*). Bei einer Sequenz von Reizmustern bestehend aus irrelevanten und 2 zu identifizierenden visuellen Objekten, die eines nach dem anderen für eine Zeitdauer von 100 ms dargeboten werden, kann es zum Moment der unterdrückten Aufmerksamkeit (*Attentional Blink*) kommen. Das zweite zu identifizierende visuelle Objekt kann innerhalb einer Zeitspanne von 500 ms nach Darbietung des ersten relevanten visuellen Objekts nicht erkannt werden [Gold02]. Die Blindheit für Veränderungen (*Change Blindness*) zeigt eine weitere Unfähigkeit der menschlichen Bildverarbeitung, wenn visuelle Szenen nicht mit Aufmerksamkeit verfolgt werden. Erfolgt beispielsweise eine Detailänderungen an 2 ansonsten identischen visuellen Szenen, kann diese anfangs nicht erkannt werden. Erst nach langem Vergleichen wird die Aufmerksamkeit auf die Detailänderung gelenkt und somit wahrgenommen.

Fehlinterpretation

Letztlich kann es bei der biologischen Bildverarbeitung zu Sinnestäuschungen kommen, die auf Fehlinterpretationen beruhen. Diese Fehlinterpretationen basieren auf Erfahrungen, die im Gehirn abgespeichert sind. In Abb. 2-21a und Abb. 2-21b wird die Länge der Pfeile bzw. Balken als unterschiedlich groß wahrgenommen, obwohl die Länge jeweils gleich ist. In Abb. 2-21c erscheinen die beiden vertikalen Linien gekrümmt, obwohl sie parallel zueinander verlaufen.

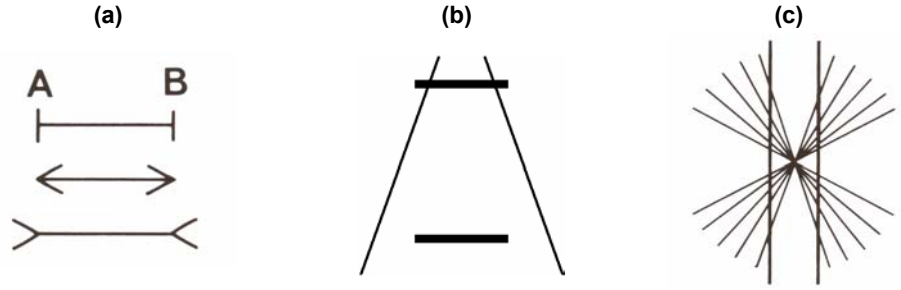


Abb. 2-21: Sinnestäuschung beruhend auf Fehlinterpretationen: (a) Müller-Lyer'sche Täuschung [Birb02], (b) Ponzo-Täuschung, (c) Hering'sche Täuschung [Birb02]

2 Verarbeitungsformen

Bei der biologischen Bildverarbeitung unterscheidet man 2 wesentliche Verarbeitungsformen [Gold02]. Das ist zum einen die datengesteuerte (*Bottom-Up*) Verarbeitung und zum anderen die konzeptgesteuerte (*Top-Down*) Verarbeitung. Die datengesteuerte Verarbeitung ist reizgetrieben, da bestimmte visuelle Objekte die Aufmerksamkeit auf sich ziehen (*Pop-Out*). Es findet hierbei kein Gebrauch vom Erfahrungsschatz statt. Bei der konzeptgesteuerten Verarbeitung hingegen lenken vorherige Erfahrungen eine aktive Suche nach bestimmten Mustern im visuellen Objekt.

2 Verbindungsstrukturen

Diese beiden Verarbeitungsformen werden durch 2 unterschiedliche Arten von neuronalen Verbindungsstrukturen gestützt, die durch Experimente bekannt geworden sind. Man unterscheidet detektierende (*Feature-Extracting*) und assoziative (*Assembly-Forming*) Verbindungstrukturen [Sing95]. Die detektierenden Verbindungsstrukturen verwenden Vorwärtsverbindungen (*Feed-Forward*), um visuelle Elementarmerkmale nach dem Bottom-Up Prinzip zu extrahieren. Diese Verbindungen sind durch eine spärliche Vernetzung charakterisiert und haben eine reduzierte Plastizität, da die Erkennung von visuellen Merkmalen in der Regel konstant bleibt. Die assoziativen Verbindungsstrukturen bestehen aus rückführenden (*Feed-Back*) und reziproken (*Reciprocal-Associative*) Verbindungen, um extrahierte Elementarmerkmale miteinander zu verknüpfen und als Repräsentationen abzuspeichern. Diese Art von Verbindungsstrukturen eignen sich für die Verarbeitung nach dem Top-Down Prinzip. Für die Merkmalsverknüpfung und die assoziative Speicherung wird eine dichtere Vernetzungsstruktur benötigt, und es ist eine erhöhte Plastizität dieser Verbindungen beobachtbar. Aufgrund der permanenten Adaptivität der assoziati-

ven Verbindungsstrukturen resultiert die Fähigkeit, für neue visuelle Objekte weitere Repräsentationen zu generieren. Basierend auf diesen beiden Verarbeitungsformen gibt es verschiedene Theorien und Modelle zur visuellen Objekterwahrnehmung.

2.4.1 Modelle zur visuellen Objektwahrnehmung

Unter Objektwahrnehmung versteht man das Erkennen von Objekten mittels des visuellen Systems [Gold02]. Dabei werden 3 Verarbeitungsstufen durchlaufen: Wahrnehmen, Wiedererkennen und Kategorisieren.

Wahrnehmen

Die Stufe des Wahrnehmens beinhaltet das Erfassen der Form von visuellen Objekten. Es werden zuerst einfache Merkmale und dann Merkmalskomplexe entdeckt, die dann in der weiteren Verarbeitung zu Einheiten zusammengefügt werden. Das Wahrnehmen visueller Objekte findet auf unterschiedlichen Komplexitätsebenen statt, da ein breites Spektrum von groben visuellen Elementen (Merkmalskomplexe) bis hin zu sehr feinen visuellen Elementen (einfache Merkmale) berücksichtigt wird.

Wiedererkennen

Die Stufe des Wiedererkennens umfasst die Identifizierung von wahrgenommenen visuellen Objekten. Dabei werden die wahrgenommenen visuellen Einheiten oder Objekte mit bereits im Gedächtnis abgespeicherten visuellen Objekten verglichen. Wenn der Eindruck von Familiarität vorliegt, findet ein Wiedererkennen von visuellen Objekten statt.

Kategorisieren

Die Stufe des Kategorisierens behandelt die Klassifizierung von wahrgenommenen und wiedererkannten visuellen Objekten. Es erfolgt ein Einsortieren der visuellen Objekte in eine bereits bestehende Ordnung bzw. Kategorie. Das korrekte Kategorisieren setzt voraus, dass der Kontext und die Beziehung zu anderen visuellen Objekten richtig erkannt wird.

Alle 3 Verarbeitungsstufen sind eng mit einem Lernprozess verbunden. Beim Wahrnehmen können dadurch feinere Unterscheidungen von Merkmalen vorgenommen werden, beim Wiedererkennen kann auf eine größere Anzahl an abgespeicherten Objekten zurückgegriffen werden, und schließlich erlaubt ein Lernprozess eine detailliertere Klassifizierung von visuellen Objekten aufgrund von Veränderungen der bereits bestehenden Ordnung. Anhand der beschriebenen 3 Verarbeitungsstufen ist die Objektwahrnehmung nicht ausschließlich von den visuellen Reizen abhängig (Wahrnehmen), sondern auch von kognitiven Faktoren (Wiedererkennen und Kategorisieren).

Marr's Theorie

Die Theorie von Marr [Marr82] beschreibt das visuelle System als ein algorithmisch arbeitendes System, dessen Stärke in der Analyse von visuellen Einzelmerkmalen liegt. Es werden dabei 3 Verarbeitungsstufen unterschieden. In der ersten Stufe wird aus dem Eingangsbild (visuelles Objekt auf der Retina), welches durch unterschiedliche Intensitätswerte repräsentiert wird, eine Rohskizze (*Primal Sketch*) von visuellen Elementarmerkmalen, wie Ecken und Kanten, angefertigt. Die zweite Stufe führt dann zu einer zweieinhalbdimensionalen Skizze (*2½-D Sketch*), in der wichtige Organisationsregeln aus der natürlichen Umwelt bereits berücksichtigt werden. Die letzte Stufe führt zur vollen dreidimensionalen Repräsentation des Objektes, die man schließlich wahrnimmt.

Biederman's Theorie

Bei der Theorie von Biederman [Bied87] basiert die Objektwahrnehmung wie bei der Rohskizze von Marr auf der Zerlegung von visuellen Elementarmerkmalen. Diese visuellen Merkmale sind allerdings hierbei elementare Teilkörper, wie Zylinder, Quader und andere Körper, die als Geone ("geometrisches Ion") bezeichnet werden. Den Analysen von Biederman zufolge ist ein Satz von 36 unterschiedlichen Geonen für die Erkennung visueller Objekte ausreichend, wenn die einzelnen Geone identifiziert werden können, aus denen die Objekte bestehen.

Merkmalsintegrationstheorie

Bei der Merkmalsintegrationstheorie (*Feature Integration Theory*) von Treisman [Trei80] umfasst die Objektwahrnehmung 2 Stufen (siehe Abb. 2-22). In der ersten präattentiven Stufe werden visuelle Elementarmerkmale, wie bei den Theorien von Marr und Biederman, extrahiert. Zu diesen Elementarmerkmalen (*Primitives*) gehören Linienbögen, Linienendpunkte, Linienneigungen, Farbe, Bewegung, geschlossene Flächen, Kontrast und Helligkeit. In dieser ersten Stufe wird keine Aufmerksamkeit benötigt. In der zweiten

Stufe der aufmerksamkeitsgerichteten Verarbeitung werden die extrahierten visuellen Merkmale kombiniert. Dazu ist der Fokus der Aufmerksamkeit notwendig.

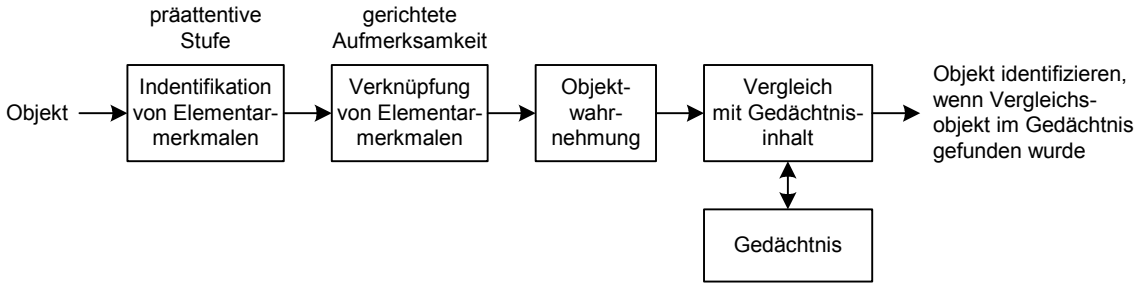


Abb. 2-22: Verarbeitungsstufen der Merkmalsintegrationstheorie (modifiziert von [Gold02])

Anhand dieser beiden Verarbeitungsstufen wird ein dreidimensionales Objekt wahrgenommen, welches mit einer im Gedächtnis abgespeicherten Repräsentation verglichen wird. Wenn eine Übereinstimmung gefunden wurde, kann das Objekt identifiziert werden.

Alle 3 Theorien gehen von einem Anfangsstadium aus, in dem das Objekt in visuelle Elementarmerkmale zerlegt wird. Dann folgen eine oder mehrere Stadien, indem die Elementarmerkmale wieder zum ganzen Objekt verknüpft werden. Bei Marr und Biederman findet ausschließlich eine Bottom-Up Verarbeitung statt, die lange Zeit als zu bedeutsam eingeschätzt wurde [Gold02]. Hierbei spielt die Aufmerksamkeit keine Rolle. Die Merkmalsintegrationstheorie dagegen verwendet für die Verknüpfung der Elementarmerkmale eine Top-Down Verarbeitung.

Bindungsproblem

Nach der Merkmalsintegrationstheorie fungiert die notwendige Aufmerksamkeit als "Leim", um verschiedene visuelle Elementarmerkmale zu einem visuellen Objekt zu verbinden. Neben der korrelierten bzw. synchronen Aktivität von Neuronen [Mals81] kann die Aufmerksamkeit ebenfalls zur Lösung des Problems führen, wie die Zuordnung einzelner visueller Merkmale zu einem visuellen Objekt erfolgt. Dieses Problem wird als Bindungsproblem (*Binding Problem*) bezeichnet. Das Bindungsproblem besteht nicht nur innerhalb des visuellen Systems, sondern auch innerhalb anderer Sinnessysteme oder bei der Zusammenfassung verschiedener Sinnessysteme. Die Betrachtung eines Autos (visuell) und die Wahrnehmung des Motorgeräuschs (auditiv) und des Abgasgeruchs (olfaktorisch), lässt unter Umständen auf einen leistungsstarken mit Diesel-Treibstoff angetriebenen Sportwagen schließen.

Low-Level/High-Level Operationen

Bei der Zerlegung in visuelle Einzelmerkmale findet eine Vielzahl von parallelen Low-Level Operationen statt, die mit detektierenden Verbindungsstrukturen die Bottom-Up Verarbeitung realisieren. Diese Low-Level Operationen setzen den ventralen Pfad im menschlichen visuellen System um, der die "Was"-Informationen der visuellen Objekte bearbeitet. Bei der Verknüpfung der Elementarmerkmale kommen serielle High-Level Operationen zum Einsatz, die mit assoziativen Verbindungsstrukturen die Top-Down Verarbeitung umsetzen. Hierbei wird der dorsale Pfad des visuellen Systems realisiert, der für die Verarbeitung der "Wo"-Informationen von visuellen Objekten zuständig ist.

Feature/Conspicuity/Saliency Maps

Das Modell von Itti [Itti98] in Abb. 2-23 basiert auf der Merkmalsintegrationstheorie und demonstriert zugleich welche Größe eines pulscodierten neuronalen Netzes für eine biologisch motivierte Objektwahrnehmung notwendig ist. Das Modell verwendet 3 Arten von Schichttypen. Die Merkmalsschichten (*Feature Maps*) erlauben eine parallele Verarbeitung und werden in die 3 Gruppen Farbe, Intensität und Orientierungen ausgehend vom Eingangsbild aufgeteilt. Die Merkmalsgruppen werden jeweils in Auffälligkeitsschichten (*Conspicuity Maps*) kombiniert, die dann in die Aufmerksamkeitsschicht (*Saliency Map*) projiziert werden. Die Motivation der Aufteilung in 3 Modalitäten (Farbe, Intensität und Orientierungen) ist, dass ähnliche Merkmale sehr stark um Aufmerksamkeit konkurrieren (Verarbeitung in der jeweiligen *Conspicuity Map*), während unterschiedliche Modalitäten unabhängig zur Aufmerksamkeit beitragen (unabhängige Verbindungen der *Conspicuity Maps* zur *Salicency Map*).

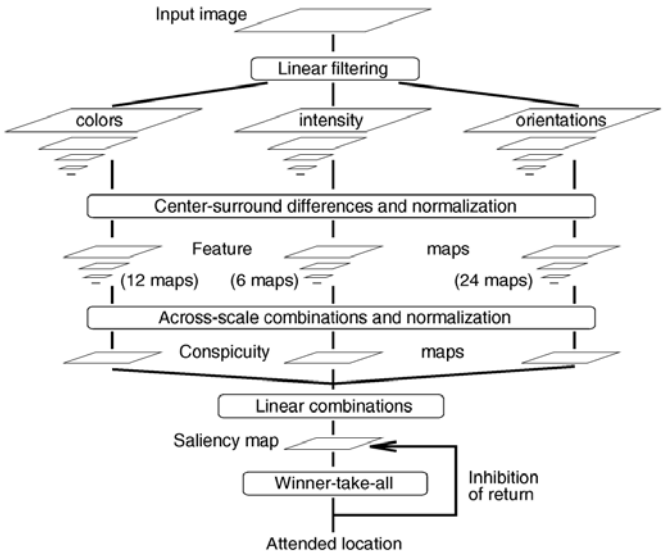


Abb. 2-23: “Model of Saliency-Based Visual Attention for Rapid Scene Analysis” [Itti98]

Resümee

Im Rahmen dieses einführenden Kapitels wurden die Funktionen und das dynamische Verhalten von biologischen Neuronen beschrieben. Es wurden 4 pulscodierte Neuronenmodelle (Integrate-and-Fire, Spike-Response und Marburger Modell, sowie das in dieser Arbeit verwendete Neuronenmodell mit adaptiven Synapsengewichten) vorgestellt, die eine einfache Modellierung des biologischen Neurons anstreben. Beim Neuronenmodell mit adaptiven Synapsengewichten handelt es sich um ein Integrate-and-Fire Modell ohne Leckwiderstand. Das bedeutet, dass das Membranpotential eines Neurons unverändert bleibt, wenn es nicht über einen externen Reiz oder über Pulse von präsynaptischen Neuronen stimuliert wird (siehe Gl. 2-13 auf Seite 19). Die Synapsengewichte werden permanent adaptiert und klingen exponentiell ab (siehe Gl. 2-14 auf Seite 20). Nur wenn das präsynaptische Neuron pulst und das postsynaptische Neuron nicht pulst, erfolgt eine positive oder negative Verstärkung des Synapsengewichts. Eine Verstärkung eines Synapsengewichts kann über Spannungsgrößen, die vom Membranpotential des postsynaptischen Neurons abhängig sind (siehe Gl. 2-15), oder über Stromgrößen, die die Pulsaktivität von mehr als einem präsynaptischen Neuron berücksichtigt (siehe Gl. 2-20 auf Seite 22), erfolgen. Weiterhin wurde die Pulsverarbeitung im menschlichen visuellen System skizziert, die für eine Objektwahrnehmung notwendig ist. Bei dieser Pulsverarbeitung findet eine getrennte und parallele Verarbeitung von visuellen “Wo”- und “Was”-Informationen statt (siehe Abschn. "2 Visuelle Pfade" auf Seite 25). Das Ziel einer technischen Lösung für die Objektwahrnehmung nach dem biologischen Vorbild zeigt, dass eine Vielzahl von Neuronen und Synapsen notwendig sind, um eine kombinierte daten- und konzeptgesteuerte Verarbeitung zu ermöglichen (siehe Abschn. "2 Verarbeitungsformen" auf Seite 29). Die datengesteuerte Bottom-Up Verarbeitung kommt mit einer spärlichen Verbindungsstruktur von Neuronen aus, während die konzeptgesteuerte Top-Down Verarbeitung eine dichte Verbindungsstruktur von Neuronen benötigt (siehe Abschn. "2 Verbindungsstrukturen" auf Seite 29). Den Gesamtprozess der menschlichen Objektwahrnehmung zu verstehen, indem man nur das Verhalten einzelner Neuronen untersucht, ist nach [Marr82] vergleichbar mit dem Versuch das Fliegen von Vögeln zu verstehen, indem man nur die Federn studiert. Erst nachdem die Aerodynamik verstanden wurde, werden die Strukturen der Federn und die unterschiedlichen Formen von Flügeln verständlich.

Es ist also erforderlich, Experimente und Untersuchungen im Kontext einer Vielzahl von Neuronen und Synapsen durchzuführen, um den Prozess der Objektwahrnehmung richtig zu verstehen. Dies erfordert die Simulation von großen pulscodierten neuronalen Netzen.

KAPITEL 3

Simulation pulscodierter neuronaler Netze

In diesem Kapitel wird der Ablauf einer digitalen Simulation von pulscodierten neuronalen Netzen (PCNN Simulation) vorgestellt. Dabei werden die Flaschenhals-Probleme der PCNN Simulation diskutiert und bekannte Konzepte skizziert, die der Simulationsbeschleunigung dienen. Schließlich erfolgt eine Gegenüberstellung von analoger und digitaler PCNN Emulation. Danach werden relevante digitale Simulationsplattformen und aktuelle Forschungsprojekte vorgestellt, die sich für die Simulationsbeschleunigung von pulscodierten neuronalen Netzen eignen. Eine Effizienzbetrachtung der unterschiedlichen Typen von Simulationsplattformen verdeutlicht die Vorteile einer FPGA basierten digitalen Emulation von pulscodierten neuronalen Netzen.

3.1 Grundlegender Simulationsablauf

Der grundlegende Ablauf einer PCNN Simulation lässt sich in 2 sequentiell ablaufende Simulationsphasen unterteilen (siehe Abb. 3-1).

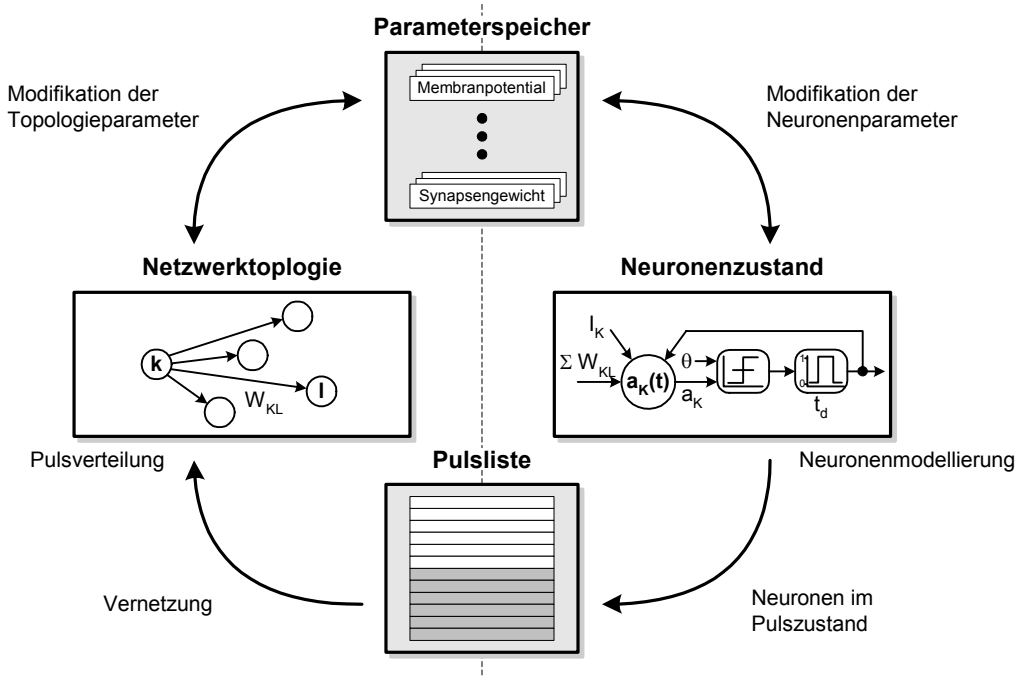


Abb. 3-1: Ablauf einer digitalen PCNN Simulation (basierend auf [Schä02])

2 Simulationsphasen

Die erste Simulationsphase umfasst die Neuronenzustandsberechnung (rechte Seite in Abb. 3-1). Hier werden für jedes Neuron im Netzwerk die jeweiligen Neuronenparameter aktualisiert. Die Neuronenparameter sind abhängig vom verwendeten Neuronenmodell und können neben dem jeweiligen Membranpotential eines Neurons eine Vielzahl von weiteren Parametern für die Modellierung von Dendritenpotentialen und Synapsengewichten darstellen. Für die Aktualisierung eines Neuronenzustands werden die entsprechenden Neuronenparameter vom Parameterspeicher gelesen und modifiziert zurückgeschrieben. Wenn das errechnete Membranpotential einen Schwellwert erreicht hat, wird dieses Neuronen gesondert in einer Pulsliste abgelegt, da es sich nun im Pulszustand

befindet. Da man durch die Modifikation der Neuronenparameter meistens ein Abklingverhalten modelliert und das Ausgangsverhalten der Neuronen ermittelt, wird diese Simulationsphase auch als Abklingphase (*Decay Phase* [Fran99]) oder als Ausgangsphase (*Output Phase* [Jahn96]) bezeichnet. Im weiteren Verlauf dieser Arbeit wird diese Simulationsphase als Neuronenzustandsphase bezeichnet, da in dieser Phase der aktuelle Neuronenzustand ermittelt wird.

Die zweite Simulationsphase behandelt die Netzwerktopologieberechnung (linke Seite in Abb. 3-1). In dieser Phase werden die im Pulszustand befindlichen Neuronen aus der Pulsliste gelesen und die Pulse in Abhängigkeit von der Vernetzungsstruktur (Netzwerktopologie) an die entsprechenden Neuronen im Netzwerk verteilt. Die Kennzeichnung, dass ein Neuron einen Puls empfängt, wird im Parameterspeicher abgelegt, damit der Einfluss dieses Pulses in der nächsten Neuronenzustandsberechnungsphase berücksichtigt werden kann. Werden durch die Pulsübertragung weitere Topologieparameter, wie die Verbindungsstärke zwischen Neuronen in Form von Synapsengewichten, beeinflusst, werden diese modifiziert und in den Parameterspeicher zurückgeschrieben. Da in dieser Simulationsphase im wesentlichen eine Pulsverteilung stattfindet, in der sich die Pulse im Netzwerk ausbreiten und somit andere Neuronen stimulieren, wird diese Phase auch als Ausbreitungsphase (*Propagate Phase* [Jahn96]) oder Stimulationsphase (*Stimulation Phase* [Fran99]) bezeichnet. In dieser Arbeit wird diese Simulationsphase als Netzwerktopologiephase bezeichnet, da in dieser Phase die Informationen zur Netzwerktopologie benötigt werden, um anhand der Einträge in der Pulsliste den Einfluss auf postsynaptische Neuronen oder die Aktivität von präsynaptischen Neuronen zu ermitteln.

3.1.1 Simulationsverfahren

Das Simulationsverfahren, das bei der Durchführung einer PCNN Simulation angewendet wird, kann man in Bezug auf 5 Eigenschaften differenzieren: Simulationsmethode, Simulationsschritte, Simulationsberechnung, Simulationstopologie und Simulationsparallelität.

Sequentiell/ Parallel

Die Simulationsmethode betrachtend unterscheidet man zwischen einem sequentiellen Simulationsverfahren, das auf einem Einprozessor-System abläuft, und einem parallelen oder verteilten Simulationsverfahren, das auf einem Multiprozessor-System abläuft.

Zeitscheibe/ Pulsereignis

Bezüglich der Simulationsschritte unterscheidet man zwischen einem Zeitscheiben-basierten (*Time-Driven*) und einem Pulsereignis-basierten (*Event-Driven*) Simulationsverfahren. Bei einem Zeitscheiben-basierten Simulationsverfahren wird ein Simulationsablauf nach Abb. 3-1 in fest definierten Zeitscheiben Δt durchlaufen. Bei einem Pulsereignis-basierten Simulationsverfahren wird das Netzwerk in unterschiedlich großen Zeitabständen aktualisiert, da nur bei einem Pulsereignis der Simulationsablauf nach Abb. 3-1 durchlaufen wird. Der Vorteil eines Pulsereignis-basierten Simulationsverfahrens wird deutlich, wenn man als Maßstab für die Rechenkomplexität einer PCNN Simulation die Aktualisierungen der Synapsengewichte pro Sekunde (SUPS = *Synaptic Updates Per Second*) und pro Neuron heranzieht [Matt00]. Wenn N der Anzahl aller Neuronen im Netzwerk, S der Anzahl aller Synapsen im Netzwerk und ν der Anzahl der Pulse entspricht, die ein Neuron im Mittel pro Sekunde generiert (Pulsfrequenz), so empfängt ein Neuron durchschnittlich $\nu \cdot S/N$ Pulse pro Sekunde. Das durchschnittliche Intervall zwischen 2 Pulsen, die ein Neuron empfängt, entspricht folglich $(\nu \cdot S/N)^{-1}$. Um neuronale Dynamikeffekte berücksichtigen zu können, die auf einzelnen Pulsen beruhen, muss bei einem Zeitscheiben-basierten Simulationsverfahren die Wahrscheinlichkeit vernachlässigt werden können, dass ein Neuron in einer Zeitscheibe mehr als einen Puls empfängt. Für die Zeitscheibe muss somit die Bedingung $\Delta t << (\nu \cdot S/N)^{-1}$ erfüllt sein. Wenn man diese Bedingung berücksichtigt, ergibt sich für die Rechenkomplexität eines Neurons folgende Gleichung,

$$SUPS_{Time-Driven} = \frac{S/N}{\Delta t} >> \nu \cdot S^2/N^2 \sim S^2 \qquad \textbf{(3-1)}$$

welche die Proportionalität zu S^2 für ein Zeitscheiben-basiertes Simulationsverfahren verdeutlicht. Bei einem Pulsereignis-basierten Simulationsverfahren beeinflusst ein Neuron $\nu \cdot S/N$ Synapsen pro Sekunde. Die Rechenkomplexität eines Neurons lässt sich nach Gl. 3-2 angeben:

$$SUPS_{Spike-Driven} = \nu \cdot S/N \sim S \qquad \textbf{(3-2)}$$

Im Vergleich zu einem Zeitscheiben-basierten Simulationsverfahren, bei der die Rechenkomplexität quadratisch zur Anzahl der Synapsen zunimmt, steigt bei einem Pulsereignis-basierten Simulationsverfahren die Rechenkomplexität nur linear mit der Anzahl der Synapsen an. Die Rechenlast bei einem Pulsereignis-basierten Simulationsverfahren lässt sich außerdem drastisch reduzieren, wenn aufgrund von niedrigen Pulsraten, wie sie bei Neuronen im menschlichen Gehirn vorkommen, relevante Pulsereignisse selten sind [Mari02] [Reut02].

Analytisch/Numerisch

Hinsichtlich der Simulationsberechnung differenziert man zwischen analytischem und numerischem Simulationsverfahren. Ein analytisches Simulationsverfahren kann immer dann angewendet werden, wenn eine exakte (analytische) Lösung für die Modellgleichungen der Neuronen ermittelt werden kann. Die Berücksichtigung einer Vielzahl von nichtlinearen Differentialgleichungen oder Integralen bei der Modellierung kann allerdings zu einer Komplexität führen, die eine exakte Lösung nicht immer ermöglicht. In diesem Fall muss ein numerisches Simulationsverfahren eingesetzt werden, welches anhand von numerischen Berechnungsverfahren näherungsweise die Lösungen der Modellgleichungen bestimmt. Der Vorteil eines numerischen Simulationsverfahrens liegt in der Flexibilität, wenn Änderungen bezüglich der Neuronenmodellierung erfolgen oder alternative Neuronenmodelle verwendet werden sollen. Zusätzlich zu berücksichtigende Differentialgleichungen oder Integrale resultieren lediglich in zusätzlichen Iterationen des implementierten numerischen Berechnungsverfahrens. Bei einem analytischen Simulationsverfahren sind Änderungen bei der Neuronenmodellierung mit einem neu zu ermittelnden Berechnungsweg für eine exakte Lösung verbunden. Somit ist dieses Simulationsverfahren nur geeignet, wenn ein einziges Neuronenmodell für die Simulation berücksichtigt werden soll und keine weiteren Modelländerungen zu erwarten sind. Im Vergleich zum analytischen Simulationsverfahren ist das numerische Simulationsverfahren allerdings zeitintensiver, da numerische Berechnungsverfahren aufgrund von iterativen Operationen eine lange Berechnungsdauer benötigen.

Empfänger-/Senderorientiert

Die Simulationstopologie betreffend klassifiziert man zwischen einem empfängerorientierten (*Receptive Method*) und einem senderorientierten (*Transmittive Method*) Simulationsverfahren [Bret94]. Bei einem senderorientierten Simulationsverfahren wird das zu berechnende Neuron als Sender bzw. als präsynaptisches Neuron betrachtet und alle Synapsengewichte, über die das Neuron einen Puls senden kann, werden als Parameterblock des Neurons abgespeichert. Zu jedem Synapsengewicht ist in diesem Parameterblock zusätzlich noch eine Adresse des postsynaptischen Neurons und eine Markierung, zu welchem postsynaptischen Potential das Synapsengewicht gehört, notwendig. Bei einem empfängerorientierten Simulationsverfahren wird das zu berechnende Neuron als Empfänger bzw. postsynaptisches Neuron angesehen und alle Synapsengewichte, über die das Neuron einen Puls empfangen kann, werden als Parameterblock des Neurons abgespeichert. Zu jedem Synapsengewicht ist in diesem Parameterblock ebenfalls eine Markierung notwendig, welcher präsynaptische Puls über das Synapsengewicht das postsynaptische Potential beeinflusst. Der Vorteil eines senderorientierten Simulationsverfahrens ist, dass bei der Pulsgenerierung eines Neurons (Eintrag in die Pulsliste am Ende der Neuronenzustandsphase) der zuständige Parameterblock sämtliche Synapsengewichte enthält, über die ein Puls übertragen wird. Anhand der abgelegten Adressen der postsynaptischen Neuronen kann somit in der Netzwerktopologiephase eine Aktualisierung der entsprechenden postsynaptischen Potentiale durch Zugriff auf den Parameterspeicher erfolgen. Es werden auf diese Weise nur die Synapsengewichte aus dem Parameterspeicher evaluiert, die aufgrund der Pulsübertragung zu einem weiteren Pulsereignis beitragen können. Eine weitere Aktualisierung der postsynaptischen Potentiale erfolgt dann wieder

in der Neuronenzustandsphase. Bei einem empfängerorientierten Simulationsverfahren hingegen kann die Aktualisierung der postsynaptischen Potentiale einmalig in der Neuronenzustandsphase durchgeführt werden. Anhand der Pulsliste kann ermittelt werden, welche präsynaptischen Neuronen sich im Pulszustand befinden. Mit dieser Information und dem zeitgleichen Zugriff auf den Parameterblock kann die Aktualisierung der postsynaptischen Potentiale aufgrund eines eventuell existierenden präsynaptischen Pulses und des Abklingverhaltens erfolgen. Der Hauptnachteil bei einem senderorientierten Simulationsverfahren ist, dass in der Neuronenzustandsphase die Information über die Pulszustände der präsynaptischen Neuronen nicht genutzt werden kann, da keine Betrachtung der zu berechnenden Neuronen als postsynaptisches Neuron erfolgt [Schä99]. Bei der Berücksichtigung von adaptiven Synapsengewichten, die für die Hebb'sche Lernregel (siehe auch Abschn. "Hebb'sche Lernregel" auf Seite 11) neben dem Synapsengewicht die Zustände vom präsynaptischen und vom postsynaptischen Neuron erfordern, muss folglich ein empfängerorientiertes Simulationsverfahren angewendet werden.

neuronen-/ synapsen-parallel

Die Simulationsparallelität betrachtend differenziert man zwischen einem neuronenparallelen (*Direct Method*) und einem synapsenparallelen (*Transposed Method*) Simulationsverfahren [Jahn96] [Bret94]. Bei einem neuronenparallelen Simulationsverfahren werden Neuronen mit dem zugehörigen Parameterblock einer PE zugeordnet. Bei einem synapsenparallelen Simulationsverfahren werden die Synapsengewichte im Parameterblock auf mehrere PEs verteilt, so das alle Synapsen parallel verarbeitet werden können. Insbesondere in der Neuronenzustandsphase ist ein neuronenparalleles Simulationsverfahren einem synapsenparallelen Simulationsverfahren überlegen, da für die Aktualisierung des Membranpotentials alle Synapsengewichte notwendig sind [Jahn96]. Bei einer synapsenparallelen Verarbeitung ist ein hoher Kommunikationsaufwand zwischen den PEs erforderlich, um alle relevanten Synapsengewichte derjenigen PE zukommen zu lassen, die für die Aktualisierung des Membranpotentials zuständig ist.

Der erhöhte Kommunikationsaufwand ist eines der Flaschenhals-Probleme, die bei der Durchführung einer PCNN Simulation berücksichtigt werden müssen.

3.2 Flaschenhals-Probleme bei der Simulation

Bei der PCNN Simulation kann es zu unterschiedlichen Flaschenhals-Problemen kommen, die eine zeiteffiziente Simulationsdurchführung verhindern. Bei der Simulationsbeschleunigung von PCNNs sind 5 Flaschenhals-Probleme zu beachten [Schä02]: Rechenoperationen, Kommunikationsaufwand, Lastverteilung, Speicherkapazität und Speicherbandbreite. Die Flaschenhals-Probleme Kommunikationsaufwand und Lastverteilung dominieren nur bei einem sehr hohen Parallelitätsgrad, wenn für die Simulationsbeschleunigung eine Vielzahl von PEs verwendet wird, die effizient miteinander kommunizieren müssen und denen jeweils eine gleichmäßig hohe Rechenlast zugeteilt werden muss.

3.2.1 Rechenoperationen

Die meisten Rechenoperationen werden bei einer PCNN Simulation für die Berechnung der Neuronenzustände durchgeführt. Dafür müssen relevante Neuronenparameter aus dem Parameterspeicher zuerst gelesen werden, um die nicht-lineare Membranpotentialfunktion eines Neurons zu bestimmen. Anschließend werden die aktualisierten Neuronenparameter wieder in den Parameterspeicher zurückgeschrieben (siehe Abb. 3-1). Bei einer Vielzahl von zu berechnenden Neuronen kann bei Einprozessor-Systemen, die eine einzige PE darstellen, nur eine reine sequentielle Ausführung der Rechenoperationen erfolgen. Zusätzlich entstehen aufgrund der Lese- und Schreiboperationen der Neuronenparameter Leerlaufzeiten zwischen den Rechenoperationen, die für die Zustandsberechnung der Neuronen notwendig sind.

3.2.2 Kommunikationsaufwand

Bei einem sehr hohen Parallelitätsgrad kann es zu einem erheblichen Kommunikationsaufwand zwischen einzelnen PEs kommen, der zum größten Flaschenhals-Problem der Simulation werden kann [Bret94]. Die Zustandsberechnung eines postsynaptischen Neurons erfordert immer die Zustände der verbundenen präsynaptischen Neuronen. Eine Kommunikation zwischen PEs ist somit immer dann erforderlich, wenn Synapsenverbindungen zwischen Neuronen bestehen, die von unterschiedlichen PEs bearbeitet werden. In Abb. 3-2 ist sowohl ein horizontaler als auch ein vertikaler Verbindungsschnitt eines neuronalen Netzes dargestellt, mit dem die Zuweisung von Neuronen auf unterschiedliche PEs erfolgt. Der vertikale Verbindungsschnitt (siehe Abb. 3-2b) ist in diesem Fall bezüglich eines geringen Kommunikationsaufwands vorteilhafter, da nur 2 Synapsenverbindungen geschnitten werden.

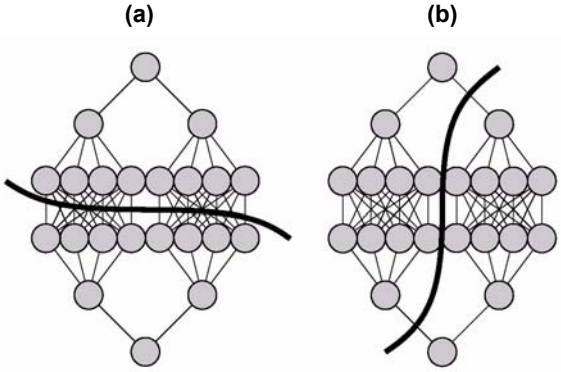


Abb. 3-2: Verbindungsschnitt eines neuronalen Netzes: (a) horizontal, (b) vertikal (modifiziert von [Wolf01])

Netzwerkpartitionierung

Die Suche nach optimalen Verbindungsschnitten wird durch eine geeignete Netzwerkpartitionierung gelöst [Prei01], die 2 Ziele verfolgt: die Reduzierung des Kommunikationsaufwands durch möglichst wenige geschnittene Synapsenverbindungen und die gleiche Lastverteilung von Rechenoperationen auf alle PEs. Simulationsexperimente [Mohr97] eines Netzwerks bestehend aus einer Eingangsschicht und 8 versteckten Schichten à 128×128 Neuronen, wobei jedes Neuron lateral mit seinen 15 nächsten Nachbarn verbunden ist und eine vorwärtsgerichtete Verbindung zu dem gleichpositionierten Neuron der nächsten Schicht hat, haben gezeigt, dass ein paralleles Simulationsverfahren mit 2 PEs genauso schnell ist wie ein sequentielles Simulationsverfahren mit nur einer PE. Das parallele Simulationsverfahren mit 8 PEs war sogar um 20 % langsamer als das sequentielle Simulationsverfahren. Die Netzwerkpartitionierung erfolgte bei diesen Experimenten nach vertikalen Spalten (*Vertical Column Strategy*), wonach jede Schicht in gleich große Stücke (*Chunks*) unterteilt wird, deren Anzahl nach Möglichkeit durch die Anzahl der verfügbaren PEs teilbar ist. Jeder PE werden dann von jeder Schicht Stücke zugeteilt, die bei einer vertikalen Anordnung der Schichten eine Spalte bilden. Aufgrund der Mehrzahl an lateralen Verbindungen gegenüber den Verbindungen zwischen den einzelnen Schichten (Faktor 15), kommt es bei dieser Netzwerkpartitionierung zu einem erhöhten Kommunikationsaufwand, der zwischen benachbarten vertikalen Spalten notwendig wird. Eine geeignetere Netzwerkpartitionierung ist in diesem Fall eine Schicht-basierte Zuordnung (*Layer Oriented Strategy*), wo eine komplette Schicht einer PE zugeteilt wird. Simulationsexperimente belegen hierbei den reduzierten Kommunikationsaufwand der parallelen Simulationsverfahren, da die Simulationszeiten mit 2 PEs um 50 % und mit 8 PEs um 74 % gegenüber einem sequentiellen Simulationsverfahren verkürzt werden [Mohr97].

3.2.3 Lastverteilung

Die Bereitstellung einer Vielzahl von PEs, die parallel Rechenoperationen durchführen können, führt nach [Mohr97] also nicht zwangsläufig zu einer Simulationsbeschleunigung. Neben einem erhöhten Kommunikationsaufwand kann eine zusätzlich ungleiche Lastver-

teilung das Flaschenhals-Problem weiter verschärfen. Eine ungleiche Lastverteilung äussert sich darin, dass die Hauptrechenlast für die Neuronenzustandsberechnung nur von einigen wenigen PEs getragen wird. Die restlichen PEs sind die meiste Zeit unbeschäftigt, da die zugeteilten Neuronen selten oder gar nicht von präsynaptischen Pulsen oder externen Eingangsreizen stimuliert werden. Dies zeigt, dass man für eine geeignete Netzwerkpartitionierung nicht nur die Netzwerkstruktur für die Reduzierung des Kommunikationsaufwands, sondern auch die Bedeutung von einzelnen Netzwerkschichten oder Neuronengruppen mit einbeziehen muss. Neuronen einer Eingangsschicht werden beispielsweise im Normalfall ständig erregt, so dass eine gleichmäßige Verteilung dieser Neuronen auf alle PEs sinnvoll ist. Des Weiteren kann die Modellierung von einzelnen Neuronengruppen eine komplexere Berechnung der Neuronenzustände erfordern, da eine Mehrzahl von nicht-linearen Neuronenparametern im Vergleich zu anderen Neuronen innerhalb des Netzwerks berücksichtigt werden muss. In diesem Fall ist es auch sinnvoll, diese Gruppe von berechnungsintensiven Neuronen gleichmäßig auf alle PEs zu verteilen.

Die Flaschenhals-Probleme der Kommunikation und Lastverteilung verdeutlichen, dass geeignete Strategien für die Netzwerkpartitionierung vor dem Simulationsablauf notwendig sind. Eine reine Parallelisierung von Rechenoperationen mittels raffiniert in Hardware implementierten PEs ist nicht ausreichend.

3.2.4 Speicherkapazität

Die Speicherkapazität stellt ein weiteres Flaschenhals-Problem dar, da sie den verfügbaren Parameterspeicher begrenzt und somit die Anzahl der maximal speicherbaren Membranpotentiale und Synapsengewichte bestimmt. Bei der Simulation von großen PCNNs muss ein ausreichend großer Parameterspeicher bereitgestellt werden, der den Zugriff von relevanten Neuronen- oder Topologieparametern mit relativ kurzen Latenzzeiten ermöglicht. Bei einer nicht ausreichenden Speicherkapazität des Parameterspeichers müssen die für die Simulation notwendigen Parameter auf Massenspeicher ausgelagert werden, die über eine ausreichende Speicherkapazität verfügen, aber dafür den Zugriff mit nur sehr langen Latenzzeiten ermöglichen. Zusätzlich ist bei Simulationsbeschleuniger-Systemen mit nicht ausreichender Speicherkapazität die Kommunikation mit einem Host-Rechner während der Simulation erforderlich, der den Datentransfer der notwendigen Parameter vom Massenspeicher zu den Speichern des Simulationsbeschleuniger-Systems übernimmt. Bei großen PCNNs mit einer dichteren Vernetzungsstruktur, die eine Vielzahl von Parametern für die Berechnung eines einzigen Neurons erfordert, kann der Datentransfer vom Massenspeicher zu Speichern mit kurzen Latenzzeiten zum dominanten Zeitfaktor während der Simulation werden.

3.2.5 Speicherbandbreite

Das größte Flaschenhals-Problem während der Simulation stellt die Speicherbandbreite dar. Bei Einprozessor-Systemen ist die Speicherbandbreite durch die CPU-Architektur festgelegt, da sie die Bit-Breiten für Daten und Befehle bestimmt (32 Bit- oder 64 Bit-Architektur). Das Produkt aus Datenbitbreite und Taktfrequenz ergibt die Datenübertragungsrate oder Speicherbandbreite in Bit pro Sekunde, die maximal möglich ist. Aufgrund der in der Einleitung beschriebenen Leistungskluft (siehe Abb. 1-1 auf Seite 2) kann die maximal mögliche Datenübertragungsrate, die sich aufgrund der hohen CPU-Taktfrequenz ergibt, bei einer großen PCNN Simulation von Einprozessor-Systemen nicht genutzt werden. Da der größte sequentielle Teil bei der Simulation der Datentransfer von den PEs zu den Parameter- bzw. Gewichtsspeichern ist [Reyn03], liegt der Hauptgrund für die schlechte Performanz von Einprozessor-Systemen während der PCNN Simulation in der limitierten Speicherbandbreite [Jahn96]. Auch eine Bereitstellung von vielen PEs, die nur sequentiell auf einen Gewichtsspeicher zugreifen können, wirkt diesem Flaschenhals-Problem nicht entgegen. Das Flaschenhals-Problem wird stattdessen zusätzlich ver-

schärft, wenn für die Simulation adaptive Synapsengewichte berücksichtigt werden müssen, da sich der notwendige Datentransfer zum Gewichtsspeicher erhöht. Simulationsbeschleuniger-Systeme operieren bei Taktfrequenzen, die 1 bis 2 Größenordnungen kleiner sind im Vergleich zu CPUs von Einprozessor-Systemen. Somit kann eine Simulationsbeschleunigung nur durch eine erhöhte Speicherbandbreite erzielt werden, die es den PEs ermöglicht, parallel auf die Gewichtsspeicher zuzugreifen. Dabei ist darauf zu achten, dass die maximal mögliche Speicherbandbreite effizient genutzt werden kann.

Es werden nun bekannte Konzepte zur Simulationsbeschleunigung vorgestellt, die alle das Ziel haben, ein oder mehrere der beschriebenen Flaschenhals-Probleme zu entschärfen.

3.3 Konzepte zur Simulationsbeschleunigung

Bei der Simulationsbeschleunigung ist der Grundsatz zu beachten, dass alles, was parallel programmiert ist, seriell reproduziert werden kann, aber nicht umgekehrt. Das bedeutet, dass die parallelen Operationen im Gehirn von einem seriell operierenden Programm, welches auf einem Einprozessor-System abläuft, reproduziert werden können (PCNN Simulation). Dieses seriell operierende Programm kann aber nicht ohne weiteres durch ein Simulationsbeschleuniger-System parallel reproduziert werden, sondern es müssen Konzepte zur Simulationsbeschleunigung entwickelt werden (PCNN Emulation).

Puls übertragen/ senden/empfangen

Der erste Schritt zur Simulationsbeschleunigung ist die Effizienzsteigerung des PCNN Algorithmus. Da Pulse die Hauptereignisse während einer Simulation sind, müssen nur die Neuronen und Synapsen betrachtet werden, die entweder zu einem Pulsereignis beitragen oder selber ein Pulsereignis auslösen können. Puls-übertragende Synapsen tragen zu einem Pulsereignis eines postsynaptischen Neurons bei, wenn sie einen Puls eines präsynaptischen Neurons übertragen. Puls-sendende präsynaptische Neuronen tragen folglich ebenfalls zu einem Pulsereignis eines postsynaptischen Neurons bei. Puls-empfangende postsynaptische Neuronen und Reiz-empfangende Neuronen befinden sich in einem erregten Zustand und sind in der Lage, ein Pulsereignis auszulösen, wenn die Erregung durch empfangene Pulse oder Reize ausreichend ist.

Im folgenden werden nun bekannte Konzepte vorgestellt, die der Simulationsbeschleunigung dienen.

3.3.1 Listenverwaltung

Anhand einer Listenverwaltung reduziert man den Simulationsaufwand, indem nur relevante Neuronen des neuronalen Netzes betrachtet werden.

Pulsliste

Die Verwendung einer Pulsliste (*Spike-Event List* [Watt94], *Spike List* [Fran99]) ermöglicht eine Pulsereignis-basierte Simulation. Neuronen, die ein Pulsereignis ausgelöst haben, werden in diese Liste eingetragen. Diese Pulsereignisse wiederum tragen zu Pulsereignissen anderer Neuronen bei, die bei Auslösung wiederum in die Liste eingetragen werden. Es entsteht somit immer eine zeitlich sortierte Liste an Pulsereignissen. Wenn sich keine Neuronen mehr in der Pulsliste befinden, bedeutet dies, dass keine Pulse gesendet werden und somit übertragen und empfangen werden können. Wenn zusätzlich keine externen Reize mehr existieren, erreicht das neuronale Netz den Zustand der kompletten Inaktivität.

Abklingliste

Die Abklingliste *(Decay List*) ist ebenfalls eine Ereignisliste, die alle Neuronen enthält, die während der Neuronenzustandsphase einer Simulation berechnet werden müssen [Fran99]. Das sind alle Neuronen, die sich nicht im Ruhezustand befinden und somit entweder einen Puls oder Reiz empfangen. In der Neuronenzustandsphase werden die Neuronen in der Abklingliste gelesen und deren Membranpotential aktualisiert. Nur wenn sich das Membranpotential eines Neurons zwischen Ruhepotential und Schwellwert befindet, wird das berechnete Neuron wieder in die Liste zurückgeschrieben. Die Neuronen, deren

Membranpotential den Schwellwert (Pulszustand) oder das Ruhepotential (Ruhezustand) erreicht haben, werden nicht mehr in die Abklingliste eingetragen. In der Abklingphase können nur aus der Abklingliste gelesene Neuronen wieder zurückgeschrieben werden, so dass sich die Anzahl der Neuroneneinträge entweder verringert oder gleich bleibt. Es kann somit eine problemlose Aktualisierung der Liste erfolgen, ohne dabei ein noch nicht bearbeitetes Neuron in der Liste zu löschen. In der Netzwerktopologiephase werden dann die Neuronen, die aufgrund der eingetretenen Pulsereignisse nicht mehr im Ruhezustand sind, der Abklingliste hinzugefügt.

Wiedervorlageliste

In die Wiedervorlageliste (*Resubmission List*) werden alle Neuronen eingetragen, die einen Puls ausgelöst haben und sich in der relativen Refraktärphase befinden [Wolf99] (siehe auch Abb. 2-2 auf Seite 8). Diese Liste ist allerdings nur sinnvoll, wenn das Neuronenmodell ein abklingendes Membranpotential mit einem dynamischen Schwellwert verwendet. In diesem Fall können die Abklingschritte des Membranpotentials zu größeren Abklingschritten zusammengefasst werden, um den Aktualisierungsaufwand des abklingenden Membranpotentials zu verringern. Das Neuron muss dann zu bestimmten Zeitpunkten wieder vorgelegt werden. Der Wiedervorlagezeitpunkt t_{i+1} lässt sich aus dem Membranpotential u und der exponentiell abklingenden Schwelle θ zum Zeitpunkt t_i wie folgt berechnen,

$$u(t_i) = [\theta(t_i) - \theta_0] \cdot e^{-\left(\frac{t_{i+1}}{\tau_\theta}\right)} + \theta_0 \Leftrightarrow -t_{i+1} = \tau_\theta \cdot \ln\left[\frac{u(t_i) - \theta_0}{\theta(t_i) - \theta_0}\right] \quad \textbf{(3-3)}$$

$$\Leftrightarrow t_{i+1} = \tau_\theta \cdot \{\ln[\theta(t_i) - \theta_0] - \ln[u(t_i) - \theta_0]\}$$

wobei τ_θ die Abklingkonstante der Schwelle und θ_0 den statischen Schwellwert darstellen (siehe auch Abb. 3-3).

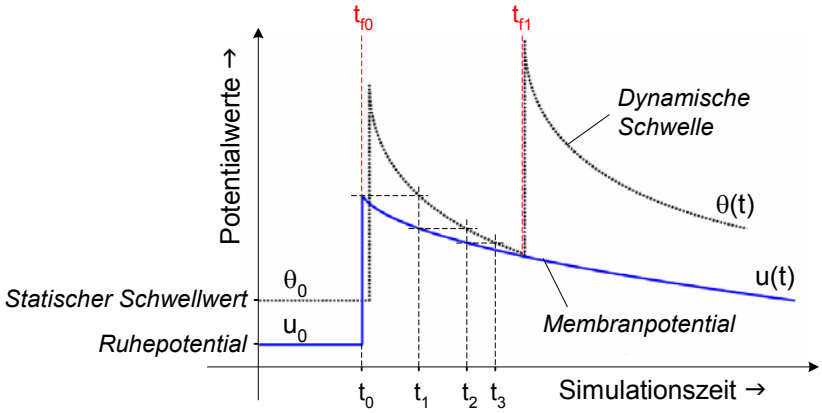


Abb. 3-3: Berechnung der Wiedervorlagezeitpunkte t_1, t_2 und t_3 (modifiziert von [Wolf01])

Für jedes Neuron in der Wiedervorlageliste müssen zusätzlich 2 Zeitstempel abgespeichert werden. Zum einen der Ablagezeitpunkt des Neurons in die Liste t_i und zum anderen der Zeitpunkt des nächsten Wiedervorlagezeitpunkts t_{i+1}. Die Abspeicherung des Ablagezeitpunkts ist notwendig, wenn das Neuron vor dem berechneten Wiedervorlagezeitpunkt stimuliert wird. In diesem Fall muss das Neuron auf Basis des Ablagezeitpunkts aktualisiert werden, um die aufgetretende Stimulation korrekt zu modellieren.

Eine direkte Berechnung des nächsten Feuerzeitpunktes t_{fi+1} (siehe Abb. 3-3) müsste nach Gl. 3-4 erfolgen,

$$[u(t_{fi}) - u_0] \cdot e^{-\left(\frac{t_{fi+1}}{\tau_u}\right)} + u_0 = [\theta(t_{fi}) - \theta_0] \cdot e^{-\left(\frac{t_{fi+1}}{\tau_\theta}\right)} + \theta_0 \quad \textbf{(3-4)}$$

wobei τ_u die Abklingkonstante des Membranpotentials und u_0 das Ruhepotential darstellen. Diese Berechnung ist allerdings sehr rechenaufwendig und führt unter Umständen zu keiner Lösung, wenn die Schwelle nicht sehr viel schneller als das Membranpotential abklingt ($\tau_u >> \tau_\theta$). Der Rechenaufwand ist deswegen so groß, weil sich das Abklingverhalten des Membranpotentials aus unterschiedlichen Abklingverhalten von Dendritenpo-

tentialen und Synapsen zusammensetzen kann und erhöht sich weiterhin, wenn τ_u nicht ein ganzzahliges Vielfaches von τ_θ ist (siehe auch Anhang D auf Seite 169).

Markierungs-speicher

Die kompakteste Listenform ist der Markierungsspeicher oder das Markierungsfeld (Tag-Speicher/-Feld). Hierbei werden Neuronen mit einer Markierung (*Tag*) in Form eines Bits versehen. Anhand einer spezifischen Codierung kann aus einer Neuronendarstellung das entsprechende Tag des Neurons im Tag-Speicher ermittelt werden. Das Lesen eines bestimmten Tags aus dem Speicher ermöglicht anhand einer entsprechenden Decodierung die ursprüngliche Neuronendarstellung. Bei Verwendung einer Abklingliste wird zusätzlich ein Tag-Speicher verwendet, um einen Mehrfacheintrag eines Neurons in die Abklingliste durch Überprüfung der entsprechenden Markierung zu vermeiden [Fran99]. Ein Tag-Speicher kann auch eine Pulsliste ersetzen, um eine größere Anzahl an pulsenden Neuronen berücksichtigen zu können. Hier wird dann nur das Tag des pulsenden Neurons abgelegt [Mehr03].

3.3.2 Parallelisierung

Eine wesentliche Maßnahme die PCNN Simulation zu beschleunigen, besteht in der Parallelisierung. Dabei unterscheidet man zwischen Parallelisierung von Low-Level Operationen und Parallelisierung von High-Level Operationen.

Low-Level Operationen

Bei der Parallelisierung von Low-Level Operationen kommen mehrere PEs zum Einsatz. Im Vergleich zu einer sequentiellen Verarbeitung mit nur einer PE (siehe Abb. 3-4a) wird dadurch die notwendige Zeit für die Rechenoperationen reduziert, und es können in der gleichen Zeit mehrere Rechenoperationen durchgeführt werden (siehe Abb. 3-4c). Aufgrund der Lese- und Schreiboperationen von Parametern, die für die Rechenoperationen bei der Aktualisierung der Neuronenzustände notwendig sind, ergeben sich zwischen den einzelnen Rechenoperationen Leerlaufzeiten. Diese Leerlaufzeiten lassen sich durch eine Fließbandverarbeitung (*Pipelining*) reduzieren, indem man die notwendigen Parameter für die nächstfolgende Rechenoperation rechtzeitig aus dem Speicher liest, damit direkt mit der nächsten Rechenoperation begonnen werden kann, während die aktualisierten Parameter der vorherigen Rechenoperation wieder in den Speicher zurückgeschrieben werden können (siehe Abb. 3-4b). Die effizienteste Form der Parallelisierung ergibt sich aus der Kombination von paralleler Verarbeitung und Pipelining (siehe Abb. 3-4d). Dabei wird der Speicherkanal für das Lesen und Schreiben der Parameter optimal ausgenutzt. Man erkennt in Abb. 3-4d, dass eine weitere höhere parallele Verarbeitung, indem die Zeit für die Rechenoperationen durch noch weitere PEs reduziert werden, keine Effizienzsteigerung zur Folge hat. Eine Effizienzsteigerung ist nur erzielbar, wenn gleichzeitig die Zugriffszeiten auf den Speicher durch eine höhere Speicherbandbreite oder durch eine komprimierte Parameterdarstellung reduziert werden.

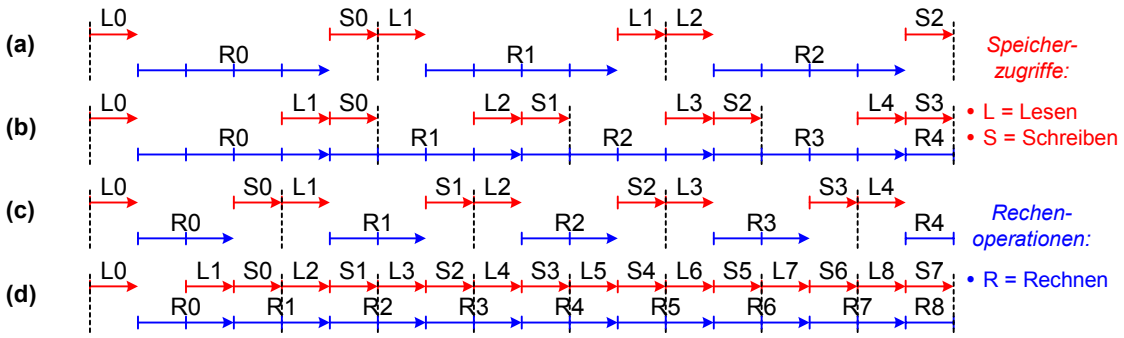


Abb. 3-4: Verarbeitungsformen: (a) Sequentielle Verarbeitung, (b) Fließbandverarbeitung (Pipelining), (c) Parallele Verarbeitung, (d) Parallele Verarbeitung mit Pipelining

High-Level Operationen

Als High-Level Operationen können bei der PCNN Simulation die unterschiedlichen Simulationsphasen betrachtet werden (siehe auch Abschn. "2 Simulationsphasen" auf Seite 33), die während des Simulationsablaufs durchlaufen werden. Eine Parallelisierung der beiden Simulationsphasen der Neuronenzustand- und Netzwerktopologieberechnung

(siehe Abb. 3-1) kann erzielt werden, indem man jede Simulationsphase als eigenständiges System vorsieht, die wie bei einer MIMD-Architektur parallel unabhängig voneinander arbeiten können. Die Parallelisierung von High-Level Operationen durch eine getrennte Implementierung der Simulationsphasen wird auch in einigen Simulationsbeschleuniger-Systemen berücksichtigt, wie *Neuron Unit* und *Weight Unit* [Schö98b] oder *Processing Chip* und *Connection Chip* [Jahn96].

3.3.3 Reduzierung der notwendigen Speicherbandbreite

Eine Reduzierung der notwendigen Speicherbandbreite wird in erster Linie durch eine Reduzierung der notwendigen Speicherkapazität erreicht, indem man nur Speicherplatz für Parameter vorsieht, die für die Simulation absolut notwendig sind.

Weight Sharing

Bei der Netzwerktopologieberechnung lässt sich die Speicherbandbreite einer regulären Vernetzungsstruktur für die Neuronen einer Verarbeitungsschicht reduzieren. Für jedes Neuron einer Schicht wird eine einzige Definition der Vernetzungsstruktur in Form einer Verbindungsmaske (*Connection Mask*) oder eines Gewichtsvektors (*Weight Vector*) herangezogen, die die regulären prä- oder postsynaptischen Verbindungen beschreibt. Somit ist es nicht erforderlich für jedes Neuron jeweils eine Definition der Vernetzungsstruktur mit Synapsengewichten im Speicher abzulegen, sondern es ist ausreichend, eine Definition der Vernetzungsstruktur pro Verarbeitungsschicht aus dem Speicher zu lesen. Da sich alle Neuronen einer Schicht die Synapsengewichte in der definierten regulären Vernetzungsstruktur teilen, wird diese Maßnahme zur Reduzierung der notwendigen Speicherbandbreite als Weight Sharing bezeichnet [Jahn98]. Ein Nachteil des Weight Sharings ist, dass nur konstante Synapsengewichte berücksichtigt werden können. Somit ist ein Lernprozess nicht durchführbar, der die individuelle Adaption von Synapsen jeder existierenden Verbindung erfordert [Schä02].

Vernachlässigbare Potentiale

Bei der Neuronenzustandsberechnung ist die erforderliche Speicherbandbreite besonders hoch, da eine Vielzahl von Neuronenparametern erforderlich ist, um den neuen Neuronenzustand zu ermitteln. Bei der Verwendung eines Neuronenmodells, bei dem das Abklingverhalten von Dendritenpotentialen berücksichtigt werden muss, lässt sich ein unterer Schwellwert ΔIP definieren, bei der Dendritenpotentiale vernachlässigbar sind (*Negligible Internal Potentials*) [Jahn96]. Dadurch verringert sich die Anzahl der notwendigen Dendritenpotentiale, die für die Berechnung des Membranpotentials vom Speicher gelesen werden müssen. Ein Nachteil bei der Vernachlässigung von Dendritenpotentialen ist, dass sich bei ungeeigneter Wahl von ΔIP das dynamische Neuronenverhalten verändern kann.

Markierte Potentialspeicherung

Einen effizienteren Speicherzugriff auf nur relevante Dendritenpotentiale lässt sich durch eine markierte Potentialspeicherung (*Dendrite Potential Compression/Tagging*) erzielen [Schö98b]. Hierbei werden alle nicht vernachlässigbaren und somit relevanten Dendritenpotentiale im Speicher hintereinander abgelegt, ohne dass Speicherplätze für vernachlässigbare Dendritenpotentiale berücksichtigt werden. Bei dieser Form der Abspeicherung geht allerdings die Zuordnung der Dendritenpotentiale zu dem jeweiligen Neuron verloren. Dieses Problem kann mit einem parallel zugreifbaren Markierungsspeicher gelöst werden, der alle im Netzwerk vorhandenen Dendritenpotentiale mit einem Tag berücksichtigt. Allerdings weisen nur relevante Dendritenpotentiale einen gültigen Tag auf. Durch das parallele Lesen der relevanten Dendritenpotentiale und des Bit-Stroms von Tags im Markierungsspeicher kann die Zuordnung der Dendritenpotentiale wiederhergestellt werden, die für die Berechnung des jeweiligen Neuronenzustands notwendig ist.

3.3.4 Nutzung von Cache-Speichern

Cache-Speicher sind sehr schnelle Pufferungsspeicher, die zwischen einem sehr schnellen Prozessor und langsamen Arbeitsspeicher in Form von DRAMs eingesetzt werden. Die Operation von Cache-Speichern erfolgt im wesentlichen nach 2 Kriterien: zeitli-

che Lokalität (vom Prozessor angefragte Daten werden sehr wahrscheinlich in naher Zukunft wieder angefragt) und räumliche Lokalität (Daten, die sich in unmittelbarer Nähe von angefragten Daten im Arbeitsspeicher befinden, werden sehr wahrscheinlich in naher Zukunft angefragt). Der Zugriff eines Cache-Speichers auf den Arbeitsspeicher erfolgt stets im Rahmen eines Blocks mehrerer Datenwörter, der auch als Cache-Zeile (*Cache Line*) bezeichnet wird [Gilo93]. Die Performanz von Cache-Speichern hängt im wesentlichen von der Fehlertrefferrate (*Miss Rate*) ab, die das Verhältnis der Anzahl an Fehlzugriffen zu der Gesamtanzahl an Zugriffen auf den Cache-Speicher darstellt. Der effiziente Einsatz von Cache-Speichern erfordert eine Minimierung der Fehlertrefferrate und verlangt somit nach einer maximal möglichen Cache-Größe.

Cache-Zeile/-Größe

Untersuchungen haben gezeigt, dass für die Simulation von neuronalen Netzen mit zunehmender Größe der Cache-Zeile die Fehlertrefferrate reduziert wird, wenn der Cache-Speicher zur Pufferung von Synapsengewichten verwendet wird [Tay92]. Außerdem zeigt sich bei einer geeigneten Netzwerkpartitionierung, dass die Fehlertrefferrate für unterschiedliche Cache-Größen konstant ist. Dies ermöglicht die Bestimmung einer minimal notwendigen Cache-Größe (*Optimal Cache Size*) [Tay92].

Weight Cache

Bei der Netzwerktopologieberechnung lassen sich Cache-Speicher einsetzen, die mit der Neuronenzustandsberechnungseinheit und dem Gewichtsspeicher, der sämtliche Synapsengewichte des zu emulierenden Netzwerks enthält, verbunden sind (*Weight Caches*) [Schö98b]. Für eine Effizienzsteigerung kommen 2 Weight Caches zum Einsatz, die wechselweise als Sender und als Akkumulator fungieren und jeweils aus 2 parallel zugreifbaren Speichern bestehen. Der erste Speicher eines Weight Caches muss über eine Speicherkapazität verfügen, um alle vorhandenen Dendritenpotentiale im Netzwerk abspeichern zu können. Der zweite Speicher eines Weight Caches dient als Markierungsspeicher aller Dendritenpotentiale. Dieser Markierungsspeicher ist für die Zuordnung zu einem Neuron notwendig. Wenn man das Marburger Neuronenmodell betrachtet (siehe Kapitel 2.2.3 auf Seite 18), müssen für die Berechnung des Neuronenzustands zum Zeitpunkt t_n alle Dendritenpotentiale DP_i aktualisiert werden. Die Dendritenpotentiale berechnen sich zum einen aus der Summe der eingehenden Pulse x_j, die über eine jeweilige Synapse w_{ij} verlaufen, und zum anderen aus dem exponentiellen Abklingverhalten anhand der Abklingkonstante τ_{DPi} (siehe Gl. 3-5).

$$DP_i(t_n) = \sum_j w_{ij} \cdot x_j(t_n) + DP_i(t_{n-1}) \cdot e^{-\frac{(t_n - t_{n-1})}{\tau_{DPi}}} = \Delta DP_i(t_n) + DP_i(t_{n-1}) \cdot e^{-\frac{(t_n - t_{n-1})}{\tau_{DPi}}} \quad \textbf{(3-5)}$$

Die Erhöhung eines Dendritenpotentials durch eingehende Pulse lässt sich, wie in Gl. 3-5 dargestellt, als ΔDP_i zusammenfassen und wird in einem Weight Cache abgelegt. Dafür sind neben dem Pulszustand, der von der Neuronenzustandsberechnungseinheit ermittelt wird, die entsprechenden Synapsengewichte aus dem Gewichtsspeicher notwendig. Zum Simulationszeitpunkt t_n werden die Dendritenpotentialerhöhungen $\Delta DP_i(t_n)$ aus einem Weight Cache gelesen und zur Neuronenzustandsberechnungseinheit gesendet (Sender-Funktion). Zeitgleich sammelt der andere Weight Cache die ermittelten Pulszustände des nächsten Simulationszeitpunkts t_{n+1} ein, um diese mit den entsprechenden Synapsengewichten zu akkumulieren und die resultierenden Dendritenpotentialerhöhungen $\Delta DP_i(t_{n+1})$ in den Cache zu schreiben (Akkumulator-Funktion). Zum Simulationszeitpunkt t_{n+1} wechseln die Weight Caches ihre Funktion, damit die aktuellen Dendritenpotentialerhöhungen wieder für die Neuronenzustandsberechnung zur Verfügung stehen.

Stimulation Cache

Bei der Neuronenzustandsberechnung kann ebenfalls ein Cache-Speicher zum Einsatz kommen, der Parameterblöcke mit allen notwendigen Stimulationsdaten des jeweiligen Neurons enthält (*Stimulation Cache*) [Fran99]. Wenn das Neuronenmodell ein abklingendes Membranpotential berücksichtigt, werden die Stimulationsdaten von Neuronen, die einen Puls generiert haben, im Stimulation Cache gehalten. Die Stimulierungen, die nach einem generierten Puls notwendig sind, können dann bei diesen Neuronen durchgeführt werden, ohne auf den externen Parameterspeicher zuzugreifen. Für den effizienten Einsatz eines Stimulation Caches muss sichergestellt werden, dass die Stimula-

tionsdaten eines Neurons nicht die Cache-Größe überschreiten. Weiterhin muss die Operation des Caches nach weiteren Neuronenmodell-spezifischen Kriterien erfolgen. Es müssen sich Stimulationsdaten von Neuronen im Cache befinden, die aufgrund einer hohen Stimulierungsaktivität (starker externer Eingangsreiz oder eine Vielzahl von präsynaptischen Verbindungen) oder eines sehr hohen Membranpotentials eine hohe Wahrscheinlichkeit aufweisen, in naher Zukunft einen Puls zu generieren.

Nachdem bekannte Konzepte für die Simulationsbeschleunigung diskutiert wurden, werden im folgenden digitale Simulationsplattformen vorgestellt, bei denen einige dieser Konzepte angewendet werden.

3.4 Digitale Simulationsplattformen

Die digitale PCNN Simulation kann auf unterschiedlichen Simulationsplattformen erfolgen. Eine Plattform bezeichnet in diesem Zusammenhang ein Hardware/Software-System, welches aus einem Rechner (*Computer*) mit einer bestimmten Hardware-Architektur und einem Betriebssystem in Software besteht. Ein Computer ist eine Datenverarbeitungsmaschine, die Informationen mit Hilfe eines Algorithmus verarbeitet.

Wird der PCNN Algorithmus nur von einem Software-Programm beschrieben, handelt es sich um eine Software Simulation. Werden Teile des PCNN Algorithmus oder der gesamte PCNN Algorithmus durch beschleunigende digitale Hardware abgebildet, spricht man von digitaler Emulation (siehe Abb. 3-5).

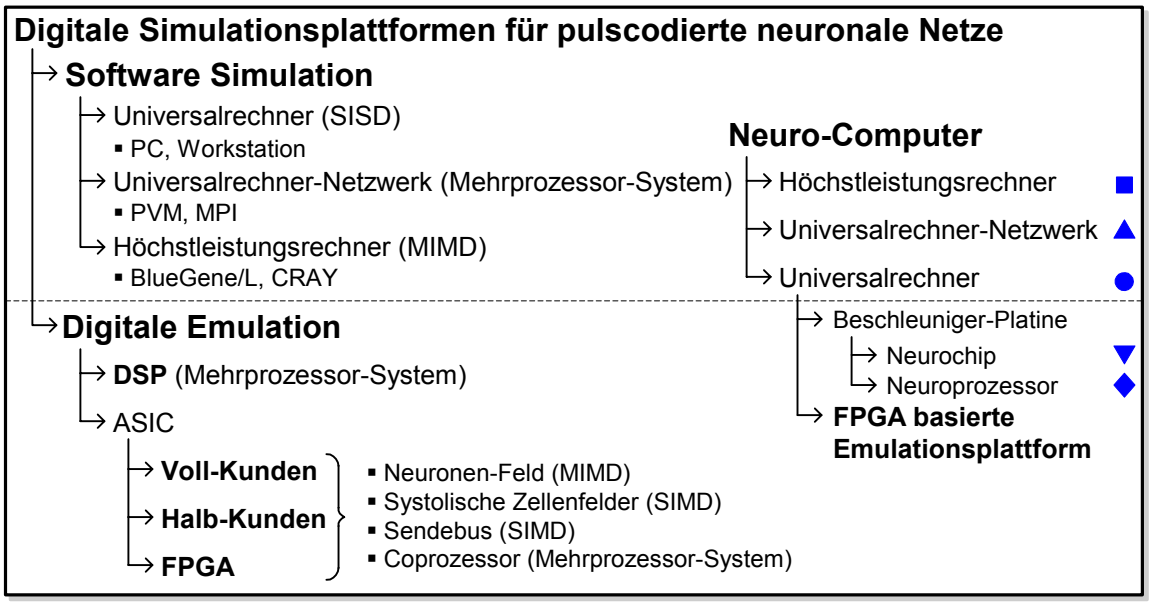


Abb. 3-5: Mögliche digitale Simulationsplattformen für neuronale Netze

Software Simulation

Eine Software Simulation wird durch Programmierung eines Universalrechners (*General-Purpose Computer*) durchgeführt. Ein Universalrechner in Form eines PCs oder einer Workstation stellt ein Einprozessor-System dar. Die Verarbeitungsgeschwindigkeit einer Software Simulation lässt sich erhöhen, indem man mehrere Universalrechner zu einem Netzwerk zusammenschaltet, um eine verteilte Simulation durchzuführen. Die einzelnen Rechner dieses Universalrechner-Netzwerks benötigen für solch eine Simulation einen effizienten Austausch von Informationen untereinander [Seif02], der durch geeignte Kommunikationsprotokolle, wie das PVM-Protokoll (*Parallel Virtual Machine*) [PVM05] oder das MPI-Protokoll (*Message Passing Interface*) [MPI05], realisiert werden kann. Die maximale Verarbeitungsgeschwindigkeit erzielt man allerdings, wenn die Software Simulation auf einem Höchstleistungsrechner (*Supercomputer*) durchgeführt wird. Als Höchstleistungsrechner werden die an Rechengeschwindigkeit und Speicherkapazität leistungsfähigsten Rechner bezeichnet. Der zur Zeit leistungsfähigste Supercomputer ist BlueGene/L der Firma IBM und wird für die Entschlüsselung der menschlichen Genstruk-

tur (*Blue Gene Project* [Blue04a]) und für die Gehirnmodellierung (*Blue Brain Project* [Blue05]) eingesetzt. BlueGene/L setzt sich aus 32768 Prozessoren (PowerPC-440 CPUs mit 0,7 GHz) zusammen, die jeweils mit 256 MB DDR-SDRAM ausgestattet sind, und ist in der Lage bis zu $70720 \cdot 10^9$ Fließkomma-Operationen pro Sekunde (70720 GFlops, *Flops* = *Floating-point Operations Per Second*) zu verarbeiten [TOP05].

Digitale Emulation

Wenn die Software Simulation auf einem Computer nachgebildet wird, der speziell für die Implementierung von neuronalen Netzen entwickelt wurde, spricht man von digitaler Emulation. Eine digitale Emulation kann mit Hilfe einer Beschleuniger-Platine (*Accelerator Board*) durchgeführt werden. Dabei übernimmt ein Universalrechner die Ansteuerung und Kommunikation zur Beschleuniger-Platine und fungiert somit als Host-Rechner. Auf einer Beschleuniger-Platine können sich Neuroprozessoren oder Neurochips befinden. Neuroprozessoren sind Software-programmierbare Schaltkreise, die mit DSPs oder ASICs realisiert werden können. Neurochips stellen den Schaltkreis eines oder mehrerer Neuronen eines speziellen Netzwerktyps dar. Digitale PCNN Emulatoren werden auch als digitale Neurohardware bezeichnet, und man klassifiziert DSP- und ASIC-basierte Implementierungen [Asan03] (siehe auch [Heem95] [Ienn96] [Lind98a] [Reyn03] für alternative Klassifizierungsbetrachtungen digitaler Neurohardware). Bei den ASIC-basierten digitalen Emulatoren unterscheidet man 4 verschiedene Implementierungsansätze: Neuronen-Feld, Systolische Zellenfelder, Sendebus und Coprozessor (siehe Abb. 3-5). Diese unterschiedlichen Implementierungsansätze werden bei der Vorstellung von digitalen Emulatoren, die auf Voll-Kunden-Entwürfen basieren, näher erläutert.

Neuro-computer

Als Neurocomputer werden in dieser Arbeit digitale Simulationsplattformen bezeichnet, die für die Simulation von pulscodierten neuronalen Netzen geeignet sind. Somit kann ein Neurocomputer durch einen Universalrechner, ein Universalrechner-Netzwerk, einen Höchstleistungsrechner oder eine Beschleuniger-Platine mit Neuroprozessoren oder Neurochips (inklusive Host-Rechner) realisiert werden.

Analog/Digital

Die PCNN Emulation kann sowohl mit integrierter Digitaltechnik als auch mit integrierter Analogtechnik realisiert werden. Eine Gegenüberstellung von analoger und digitaler PCNN Emulation ist der Tab. 3-1 zu entnehmen, wobei die 4 Gesichtspunkte Kosten, Performanz, Flexibilität und PCNN Größe betrachtet werden. Bei der analogen Emulation können nicht-lineare Operationen, wie Multiplikation, Division und Tangens-Hyperbolikus, die für die Neuronenmodellierung notwendig sind, mit nur wenigen Transistoren realisiert werden [Hamm03]. Somit ergibt sich auch eine geringere Verlustleistung gegenüber einer digitalen Emulation. Den Hauptvorteil einer Emulation mit integrierter Analogtechnik stellt die erzielbare Emulationsgeschwindigkeit dar, die aufgrund einer massiven parallelen Verarbeitung erreicht werden kann. Diese massive parallele Verarbeitung ist realisierbar, wenn jedes einzelne Neuron als Hardware-Komponente innerhalb eines Neuronen-Feldes implementiert wird. Ein Nachteil bei einer analogen Implementierung ist die Störanfälligkeit gegenüber Schwankungen der Temperatur und der Versorgungsströme und -spannungen. Im Gegensatz dazu arbeitet eine digitale Implementierung mit den lediglich auszuwertenden Pegeln “0” und “1” wesentlich zuverlässiger. Die Hauptvorteile einer digitalen PCNN Emulation bestehen aber in der Flexibilität und der realisierbaren PCNN Größe. Aufgrund der Flexibilität ist eine kontinuierliche Modellanpassung möglich (Programmierbarkeit), und bei FPGA basierten Implementierungen besteht zusätzlich die Möglichkeit einer Anpassung der Hardware-Architektur (Rekonfigurierbarkeit). Die PCNN Größe wird neben der Anzahl an Neuronen von der realisierbaren Vernetzungsstruktur und somit von der Anzahl an Synapsen bestimmt. Bei einer analogen 2D-Implementierung besteht bei einer dichteren Vernetzungsstruktur das Problem, dass Verbindungsleitungen zwischen prä- und postsynaptischen Neuronen durch Multiplexen mehrfach genutzt werden müssen. Dies lässt sich durch eine rein analoge Implementierung nur sehr schwer realisieren. Dieses Problem kann allerdings umgangen werden, indem man das Multiplexen mit Hilfe des Kommunikationsprotokolls AER (*Address-Event Representation*) mit Digitallogik implementiert [Lazz93] [Gold01]. Die Bereitstellung zusätzlicher Verbindungsleitungen kann durch eine 3D-Implementierung ermöglicht werden, indem Chips gestapelt werden, und man auf diese Weise mit Hilfe von Durchbohrungen (Vias) Verbindungsleitungen in

der vertikalen Ebene schafft. Ein weiteres Problem bei der analogen PCNN Emulation ist die Gewichtsspeicherung, die bei adaptiven Synapsen notwendig wird. Hier kann man wieder die Digitallogik anwenden und interne digitale Speicherzellen verwenden. Allerdings wären dann die Hauptvorteile der massiven parallelen Implementierung und des geringen Flächenbedarfs einer analogen Emulation nicht mehr gewährleistet.

	Analoge PCNN Emulation	**Digitale PCNN Emulation**
Kosten • Fläche • Verlustleistung	+ wenige Transistoren realisieren notwendige nicht-lineare Operationen (*High Computational Density*) + geringe Verlustleistung (aufgrund weniger Transistoren und kurzer Operationsdauer)	– eine Vielzahl von Transistoren realisieren notwendige nicht-lineare Operationen (*Low Computational Density*) – hoher Leistungsverbrauch (aufgrund vieler Transistoren und längerer Operationsdauer)
Performanz • Geschwindigkeit • Qualität	+ massive parallele Verarbeitung (Neuronen-Feld basierte Implementierung) – störanfällig (gegenüber Temperatur- und Strom-/Spannungsschwankungen)	– sequentielle Verarbeitung (Speicherbandbreite ist zusätzlich durch maximalen Chip-Durchmesser begrenzt) + robust und zuverlässig
Flexibilität • Programmierbarkeit • Rekonfigurierbarkeit	– Schaltung an sich stellt Algorithmus dar (keine Modellanpassung möglich) – Hardware-Architektur ist fest	+ Computerprogramm repräsentiert Algorithmus (Modellanpassung möglich) + Hardware-Architektur ist rekonfigurierbar (einstellbare Genauigkeit von Festkomma-Arithmetik)
PCNN Größe • Netzwerktopologie • Anzahl an Synapsen • Anzahl an Neuronen	– begrenzte Vernetzungsmöglichkeit (Multiplexen für Mehrfachnutzung von Verbindungsleitungen ist schwierig) – Probleme bei der Synapsengewichtsspeicherung (aufgrund analoger Signale) – begrenzte Anzahl an Neuronen in einem Neuronen-Feld (durch maximale Chip-Fläche begrenzt)	+ Multiplexen von digitalen Signalen ist wesentlich einfacher + kompakte interne Speicherzellen und externe Speicher mit hoher Speicherkapazität können für die Synapsengewichtsspeicherung genutzt werden + Anzahl der Neuronen ist von der verfügbaren Speicherkapazität abhängig

Tab. 3-1: Vergleich zwischen analoger und digitaler PCNN Emulation

3.4.1 Software Simulation

Simulatoren für die Software Simulation von neuronalen Netzen werden auch als Neurosimulatoren bezeichnet (siehe auch Übersicht in [Murr03] und [Zell00]). Der Hauptvorteil von Software Simulatoren besteht in der Flexibilität, die durch die Programmierbarkeit von Software-Programmen bereitgestellt wird. Dadurch kann das Neuronenmodell sehr einfach und schnell um neue neuronale Paradigmen erweitert werden. In Bezug auf die PCNN Simulation kann man 3 Typen von Software Simulatoren unterscheiden: universelle, PCNN und Ad-hoc Simulatoren.

Universelle Simulatoren

Bei den universellen Simulatoren steht das Verhalten von einzelnen Neuronen im Vordergrund. Anhand einer graphischen Oberfläche (GUI = *Graphical User Interface*) können die Zeitverläufe von Membranpotentialen und anderen Neuronenparametern genau studiert werden. Die Simulatoren sind in der Lage, Neuronenmodelle unterschiedlicher Komplexität vom Hodgkin-Huxley Modell bis zum Integrate-and-Fire Modell zu berücksichtigen. Universelle Simulatoren, wie GENESIS (*General Neural Simulation System*) [GENE05], NEURON [NEUR05] oder SNNS (*Stuttgart Neural Network Simulator*) [SNNS05] haben ihren Ursprung in akademischen Projekten und sind frei erhältlich. Der bekannteste kommerziell erhältliche universelle Simulator ist die Matlab Neural Network Toolbox von Mathworks [MATL05].

PCNN Simulatoren

PCNN Simulatoren sind speziell auf die Simulation von pulscodierten Neuronenmodellen ausgerichtet. Im Gegensatz zu universellen Simulatoren steht bei diesen Simulatoren das Verhalten von Neuronengruppen im Vordergrund. Die PCNN Simulatoren verwenden alle ein ereignisbasiertes Simulationsverfahren, welches zuerst vom Simulator SPIKE [Watt94] angewandt wurde. Der Simulator SpikeNNS [Spik05] verwendet das Spike-Response Modell, während die beiden Simulatoren SpikeNet [Delo03] und SpikeLab [Gras98] das Integrate-and-Fire Modell verwenden. In Bezug auf Performanz und PCNN Größe ist SpikeNet der leistungsfähigste PCNN Simulator. SpikeNet benötigt für die Simulation zur Identifizierung von 400 Gesichtern auf einem 910×700 Pixeln großen Eingangsbild etwa 30 Minuten (266 MHz PowerPC-750) und ist in der Lage das für diese Aufgabe notwendige Netzwerk von $32 \cdot 10^6$ Neuronen und $145 \cdot 10^9$ Verbindungen zu berücksichtigen [Delo01].

Ad-hoc Simulatoren

Ad-hoc Simulatoren sind im Gegensatz zu universellen und PCNN Simulatoren unflexibel, da sie nur für die Simulationen eines einzigen Neuronenmodells konzipiert sind und eigens zu diesem Zweck (ad-hoc) entwickelt werden. Ein Ad-hoc Simulator besteht zumeist aus textbasierten Software-Programmen und verzichtet auf jegliche graphische Oberflächen, die der Visualisierung von Neuronen- oder Topologieparametern dienen. Der Vorteil von Ad-hoc Simulatoren ist, dass sie, im Vergleich zu universellen Simulatoren, die Simulationszeiten bis zu einem Faktor von 10 verkürzen können [Reyn03] und für die Evaluation und Erweiterung eines einzigen Neuronenmodells völlig ausreichend sind.

3.4.2 DSP basierte Emulation

DSP basierte Emulationsplattformen sind durch eine SPMD-Architektur (*Single Program Multiple Data*) und durch eine verteilte Speicherarchitektur charakterisiert. Das notwendige Software-Programm für die PCNN Simulation wird zeitgleich von mehreren DSPs ausgeführt, die jeweils unterschiedliche Datensätze bearbeiten. Diese Datensätze sind in Speichern abgelegt, auf die nur der jeweilige DSP zugreifen kann. Ein separater Host-Rechner übernimmt die Steuerung des Software-Programms und ist für den Datentransfer zu den einzelnen DSPs zuständig.

MUSIC

Die MUSIC-Plattform (*Multiprocessor System with Intelligent Communication*) [Müll95] besteht aus 21 Beschleuniger-Platinen, die jeweils mit 3 DSPs ausgestattet sind. Für die Kommunikation der insgesamt 63 DSPs von Motorola (DSP96002) wird ein Ring-Bus verwendet, der mit 5 MHz betrieben wird. Die Kommunikationslogik zwischen den DSPs wird mit FPGAs von Xilinx (XC3090) realisiert. Jeder DSP ist mit 1 MB SRAM (*Static Random Access Memory*) und 2 MB DRAM ausgerüstet und kann bei einer Taktfrequenz von 40 MHz bis zu 60 MFlops bearbeiten [Müll95]. Dies entspricht einer maximalen Gesamtperformanz von 3,78 GFlops.

ParSPIKE

Die ParSPIKE-Plattform [Wolf99] besteht aus 3 Beschleuniger-Platinen, die über einen VME-Bus (VME = *Versa Module Eurocard*) miteinander verbunden sind, und ist für die PCNN Simulation von 2^{20} Neuronen ausgelegt. Als DSP wird ein Sharc-DSP von Analog Devices (ADSP21060) verwendet, der aufgrund des relativ großen On-Chip-SRAMs von 512 KB auf einen zusätzlichen externen Speicher verzichten kann. Der On-Chip-SRAM ist in 2 Blöcke unterteilt, die jeweils unabhängig voneinander angesteuert werden können (*Dual-Ported*). Dies ermöglicht den zeitgleichen Zugriff von 2 Operanden. Der Sharc-DSP verfügt über 3 parallel nutzbare Fließkomma-Einheiten (Multiplizierer, ALU und Schieberegister) und besitzt neben einer multiprozessorfähigen Busschnittstelle eine zweite parallele Schnittstelle (*Link-Port*). Die Operationsfrequenz von 40 MHz erlaubt somit eine theoretische Performanz von 120 MFlops. Anstatt eines Ring-Busses werden die Sharc-DSPs über die multiprozessorfähige Busschnittstelle durch eine Baumstruktur miteinander verbunden, um bei einer hohen Netzwerkaktivität eine Busüberlastung zu vermeiden. Diese Baumstruktur wird im Rahmen eines Kommunikationssystems realisiert, wobei die DSPs die Baumblätter und der VME-Bus die Baumwurzel bilden [Wolf01]. Die NRC-Module (NRC = *Non-Regular Connection*) werden durch 2 Beschleuniger-Platinen dargestellt, die jeweils mit 16 DSPs bestückt sind und 2^{18} Neuronen mit irregulären Vernet-

zungsstrukturen berücksichtigen können. Die Definition von irregulären Netzwerkverbindungen wird innerhalb eines Speichersubsystems im Topologiespeicher abgelegt. Die Übertragung der Topologieparameter vom Speichersubsystem erfolgt über den Link-Port der DSPs. Die dritte Beschleuniger-Platine wird als RC-Modul bezeichnet (RC = *Regular Connection*) und ist für die Simulation von 2^{19} Neuronen mit einer regulären Vernetzungsstruktur vorgesehen. Da aufgrund der regulären Vernetzungsstruktur kein Speichersubsystem notwendig ist, können für dieses Modul 32 DSPs berücksichtigt werden. Die notwendige Logik sowohl für das Kommunikations- als auch für das Speichersubsystem werden mit FPGAs von Xilinx (XC4000) realisiert.

3.4.3 Voll-Kunden-Entwurf basierte Emulation

Im Rahmen der Emulationsplattformen, die auf einem Voll-Kunden-Entwurf basieren, werden die 4 unterschiedlichen Implementierungsansätze nach Abb. 3-5 auf Seite 44 vorgestellt.

3.4.3.1 Neuronen-Feld

BIONIC

Der digitale Neurochip BIONIC (*Biology Oriented Neural Integrated Circuit*) [Pran93] emuliert bei einer Taktfrequenz von 2 MHz 16 Neuronen nach dem Marburger Neuronenmodell (siehe Kapitel 2.2.3 auf Seite 18), die jeweils 16 Synapsen berücksichtigen können. Als Kommunikationsschnittstellen besitzt der IC (*Integrated Circuit*) einen Pulseingang und einen Pulsausgang (jeweils 1 Bit) und einen Kaskadier-Eingang, einen Kaskadier-Ausgang und einen Linking-Eingang (jeweils 24 Bit). Die integrierten HW-Module sind 16 Synapsen-Blöcke, ein 24 Bit-Addierer für die Berücksichtigung externer PSPs am Kaskadier-Eingang (die Summe wird durch den Kaskadier-Ausgang anderen externen ICs zur Verfügung gestellt), ein 24×24 Bit-Multiplizierer für die Modulation von PSPs am Linking-Eingang und ein Ausgangsfunktions-Block, der das berechnete Membranpotential mit einem Schwellwert vergleicht und für die Pulsgenerierung über den Pulsausgang zuständig ist. Die Synapse eines Neurons wird durch einen Synapsen-Block berechnet, der für die Berücksichtigung von 16 Neuronen 2 Parameterspeicher à 16×8 Bit besitzt, in denen jeweils Gewichtswert und Abklingkonstante abgelegt sind. Bei einem vorhandenen Puls wird der Gewichtswert mit dem PSP (16 Bit) addiert, welches sich aus der Multiplikation mit der Abklingkonstante und dem vorherigen PSP ergibt. In einer teilseriellen Verarbeitung werden die eingehenden Pulse seriell für jeden Synapsenblock abgespeichert. Aufgrund der implementierten Pipeline-Architektur kann beim Eingang des ersten Pulses mit der Verarbeitung der ersten Synapse des ersten Neurons begonnen werden. Während im ersten Synapsen-Block mit der Berechnung der ersten Synapse des zweiten Neurons fortgefahren werden kann, wird beim Eingang des zweiten Pulses die zweite Synapse des ersten Neurons verarbeitet. Nach 16 Takten ist die Berechnung des PSPs, welches sich aufgrund von eingehenden Pulsen ergibt, für ein Neuron abgeschlossen. Das Membranpotential setzt sich dann aus der Addition dieses PSPs und eines externen PSPs und aus der anschließenden Multiplikation durch ein Linking-Potential zusammen. Für die PCNN Simulation von $16 \cdot 10^3$ Neuronen mit insgesamt $256 \cdot 10^3$ Synapsen wäre die Kaskadierung von 1000 BIONICs notwendig, was keiner kompakten und kostenwirksamen Emulationsplattform entspricht.

3.4.3.2 Systolische Zellenfelder

SYNAPSE

Die Emulationsplattform SYNAPSE (*Synthesis of Neural Algorithms on a Parallel Systolic Engine*) [Rama95] besteht aus 4 HW-Einheiten: Kontroll-Einheit (*Control Unit*), Daten-Einheit (*Data Unit*), Neuroprozessor-Einheit (*Neuroprocessor Unit*) und Gewichtsspeicher-Einheit (*Weight Memory Unit*). Die Kontroll-Einheit ist mit einer CPU von Motorola (MC68040) ausgestattet und sowohl für die Kommunikation zu einem Host-Rechner über einen VME-Bus als auch für den direkten Datenzugriff auf den Gewichtsspeicher zuständig. Die Daten-Einheit ist ebenfalls mit einem MC68040 versehen und ist über 4

Datenbusse mit je 16 Bit und über 4 Akkumulatorbusse mit je 48 Bit mit der Neuroprozessor-Einheit verbunden. Die CPU und ein 16 MB großer Zwischenspeicher für Ein- und Ausgangsdaten sind über eigenständige Schnittstellen mit dem VME-Bus verbunden. In der Neuroprozessor-Einheit sind 8 Prozessoren (MA16), die speziell für die neuronale Signalverarbeitung entwickelt wurden [Beic93], als zweidimensionales Zellenfeld (*Bidimensional Mesh*) mit 2 Reihen und 4 Spalten angeordnet. Diese MA16-Prozessoren bestehen aus 4 parallel nutzbaren 16×16 Bit Festkomma-Multiplizierern mit einer Carry-Save-Architektur. Die Ausgänge der 4 Multiplizierer sind zu einer Addierer-Kette verschaltet, um die in der neuronalen Signalverarbeitung häufig auftretenen Operationen der Multiplikation und Akkumulation (MAC = *Multiply And Accumulate*) zu beschleunigen. Jede der 4 Spalten des Zellenfeldes ist über einen 64 Bit-breiten Datenbus mit der Gewichtsspeicher-Einheit verbunden, die eine Speicherkapazität von 128 MB für Gewichtswerte zur Verfügung stellt. Ein MA16-Prozessor operiert bei 40 MHz und ist somit in der Lage bei 16 Bit-breiten Gewichtswerten $640 \cdot 10^6$ Verbindungen pro Sekunde (CPS = *Connections Per Second*) zu verarbeiten. Bei 8 MA16-Prozessoren ist die SYNAPSE-Plattform somit in der Lage eine Gesamtperformanz von $5{,}12 \cdot 10^9$ CPS zu erzielen.

3.4.3.3 Sendebus

CNAPS

Die Emulationsplattform CNAPS (*Connected Node Architecture for Parallel Systems*) [Hamm95] basiert auf einer SIMD-Architektur, die durch Verwendung eines Sendebusses realisiert ist. Ein Chip für die Ablaufsteuerung (*Sequencer*), der mit einem Host-Rechner verbunden ist, schickt einen Befehl über einen Sendebus an 64 Prozessoren, die alle in einem zweiten Chip integriert sind. Die 64 Prozessoren können für die Gewichtsspeicherung nur auf einen 256 KB großen On-Chip-Speicher zurückgreifen, da eine externe Speicheranbindung nicht möglich ist. Jeder Prozessor beinhaltet im wesentlichen eine Festkomma-ALU für 16 Bit-breite Daten. Bei einer Taktfrequenz von 25 MHz verfügt die CNAPS-Plattform über eine Performanz von $1{,}6 \cdot 10^9$ CPS.

3.4.3.4 Coprozessor

Die bisherigen Implementierungsansätze basieren auf einer physikalischen Trennung von Kontrollfunktionen (Host-Rechner, separate CPU) und von paralleler Verarbeitung durch PEs. Ein alternativer Implementierungsansatz besteht darin durch einen Coprozessor, sowohl Kontrollfunktionen als auch parallele Verarbeitung auf einem einzigen Chip zu integrieren. Der Coprozessor bildet dabei mit einem zentralen Prozessor ein Mehrprozessor-System und unterstützt diesen bei der Durchführung von speziellen Aufgaben. Ein Coprozessor-Befehl führt wie bei einer SIMD-Architektur zur simultanen Ausführung mehrerer Berechnungen.

MMX/SSE

Bei Pentium-Prozessoren von Intel gibt es 2 Arten von Befehlssatzerweiterungen, die einem Coprozessor zugeordnet sind. Der Befehlssatz MMX (*Multimedia Extension*) wurde für die Beschleunigung von Algorithmen zur Video-, Bild- und Sprachdatenverarbeitung entwickelt und bietet die Möglichkeit, 8×8 Bit-Daten oder 4×16 Bit-Daten in einem 64 Bit-Register abzulegen, das vom Coprozessor gelesen werden kann. Der Befehlssatz SSE (*Streaming SIMD Extension*) wurde für die Performanzsteigerung von 3D-Berechnungen eingeführt, die eine höhere Genauigkeit erfordern. Hierbei werden 4×32 Bit-Fließkomma-Daten zu einem 128 Bit-Register für den Coprozessor zusammengefügt. Untersuchungen [Stre01b] haben ergeben, dass sich durch den gezielten Einsatz von MMX-Befehlen die Simulation von neuronalen Netzen um den Faktor 9,8 beschleunigen lässt. Des Weiteren konnte festgestellt werden, dass gezielte Implementierungen mit SSE-Befehlen stets langsamer sind als Implementierungen mit MMX-Befehlen.

T0 Vektorprozessor

Der T0 Vektorprozessor [Wawr96] vereint eine MIPS-II 32 Bit RISC CPU mit einem Vektor-Coprozessor für Festkomma-Arithmetik. Weitere Komponenten innerhalb des T0 Vektorprozessors sind ein 1 KB großer Befehlscache und eine 8 Bit-breite serielle Schnittstelle (TSIP). Der Vektor-Coprozessor besteht aus 16 Vektor-Registern, die jeweils

32×32 Bit Daten speichern können, 2 Vektor-Arithmetik-Einheiten (VP0 und VP1), die jeweils über 8 parallele Pipelines verfügen und somit bis zu 8×32 Bit Operationen pro Takt verarbeiten können, und einer Vektor-Speicher-Einheit (VMP), die über einen 128 Bit-breiten Datenbus auf externe Speicher zugreifen kann. Im Rahmen des mit 40 MHz betriebenen Beschleuniger-Systems Spert-II (*Synthetic Perceptron Testbed II*) für Workstations, kommen der T0 Vektorprozessor mit 16 SRAM-Speichern und ein FPGA von Xilinx zum Einsatz. Die SRAM-Speicher bieten eine Speicherkapazität von 8 MB, und das FPGA ist für die serielle Kommunikation zwischen Vektorprozessor und Workstation zuständig. Der Beschleunigungsfaktor für die Simulation eines neuronalen Netzes mit 4003 Neuronen wird für das Spert-II-System mit 6 im Vergleich zu einem RS/6000-590 Rechner von IBM (CPU mit 71,5 MHz, 256 KB Cache und 768 MB DRAM-Arbeitsspeicher) angegeben.

3.4.4 Halb-Kunden-Entwurf basierte Emulation

Als Beispiele für Emulationsplattformen, die auf einem Halb-Kunden-Entwurf basieren, werden 3 Architekturkonzepte vorgestellt, die an der Technischen Universität Berlin entwickelt wurden und speziell für das Marburger Neuronenmodell ausgelegt sind.

3.4.4.1 NESPINN

Die Emulationsplattform NESPINN (*Neurocomputer for Spiking Neural Networks*) [Jahn96] operiert bei einer Taktfrequenz von 50 MHz und besteht aus einer Beschleuniger-Platine, die 2 Chips beinhaltet: den Connection Chip und den Processing Chip (siehe Abb. 3-6a). Der Connection Chip ist für die Bereitstellung von Synapsengewichten im Rahmen einer regulären Vernetzungsstruktur zuständig. Für die Berücksichtigung von irregulären Vernetzungsstrukturen wird parallel ein DRAM mit 128 MB eingesetzt. Im Processing Chip wird die Neuronenzustandsberechnung durchgeführt, indem die eingehenden Synapsengewichte von 2 Gewichts-Einheiten (*WU* = *Weight Unit*) entgegengenommen werden, um diese an eine der 4 vorhandenen PEs weiterzuleiten. Jede PE besitzt einen On-Chip-SRAM von 2 KB für die Zwischenspeicherung von Dendritenpotentialen und hat zusätzlich über einen 24 Bit-breiten Datenbus Zugriff auf einen eigenen externen SRAM von 64 KB. In diesem externen SRAM ist für jedes Neuron ein Parameterblock abgelegt, der bis zu 16 Dendritenpotentiale (à 16 Bit) mit jeweilger Abklingkonstante (à 8 Bit) enthalten kann und einen Programm-Code, der die Tags (siehe auch Abschn. "Markierungsspeicher" auf Seite 41) für jedes Dendritenpotential (16 Bit), die arithmetische Verknüpfung der Dendritenpotentiale und den Schwellwert (8 Bit) beinhaltet.

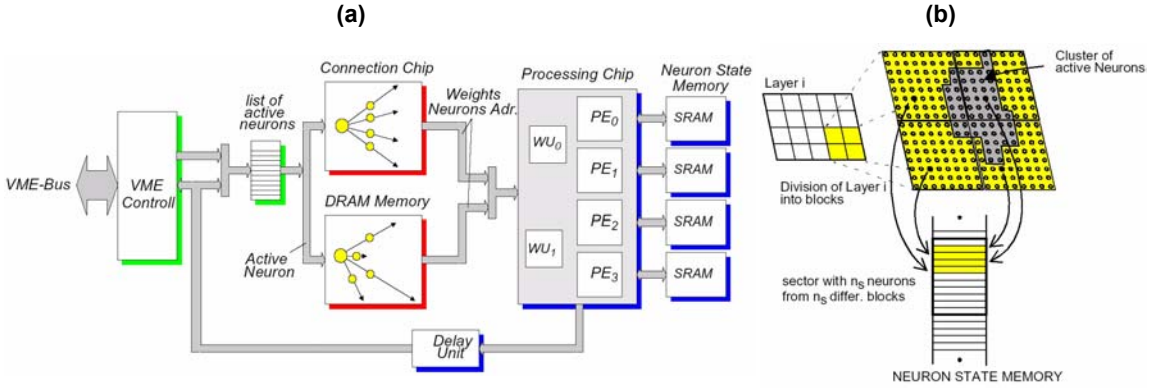


Abb. 3-6: (a) Konzept der NESPINN-Architektur [Schö98a], (b) Sektorenaufteilung [Jahn96]

Als Beschleunigungsmaßnahmen werden bei dem senderorientierten und dem neuronenparallelen Simulationsverfahren eine schnelle Festkomma-Arithmetik, vernachlässigbare Dendritenpotentiale und eine Pulsliste verwendet. Vernachlässigbare Dendritenpotentiale werden anhand von Tags im Programm-Code markiert und reduzieren bei der Neuronenzustandsberechnung die notwendige Speicherbandbreite, da diese Potentiale nicht mehr von den PEs aus dem externen SRAM gelesen werden müssen

(siehe auch Kapitel 3.3.3 auf Seite 42). Die Pulsliste wird über eine Verzögerungs-Einheit (*Delay Unit*) vom Processing Chip mit pulsenden Neuronen beschrieben, die in der nächsten Zeitscheibe vom Connection Chip gelesen werden.

Sektorenaufteilung

Für eine ausgeglichene Lastverteilung verarbeiten die PEs Sektoren von Neuronen. Hierfür werden zuerst die Netzwerkschichten in gleichgroße rechteckige Blöcke unterteilt, die in Abb. 3-6b jeweils 64 Neuronen enthalten. Jedes Neuron, welches die gleiche Position innerhalb eines Blocks hat, wird dem gleichen Sektor zugeordnet. Nach Abb. 3-6b entspricht die Anzahl der Sektoren N_S = 64. Die Anzahl der Neuronen pro Sektor entspricht der Anzahl der Blöcke N_B des gesamten Netzwerks [Roth97].

SIMD/Dataflow

Die NESPINN-Plattform ist für die PCNN Simulation von 10^5 Neuronen ausgelegt und operiert für die Bearbeitung eines Sektors in 3 Phasen: Init-, Input&Decay- und Output-Phase. In der Init-Phase werden die Programm-Codes für jedes Neuron eines Sektors in die PE geladen. In der Input&Decay-Phase empfangen die PEs gewichtete Pulse vom Connection Chip und berechnen das Abklingverhalten der Dendritenpotentiale. Die Berechnungen der PEs werden in dieser Phase vom Datenfluss (*Dataflow*) bestimmt. In der Output-Phase wird die Ausgangsfunktion der Neuronen von den PEs berechnet, die hierfür sämtliche aktualisierte Dendritenpotentiale benötigen. Aufgrund der Datenflussgesteuerten Berechnungen in der Input&Decay-Phase und den SIMD-Operationen in der Output-Phase wird die NESPINN Architektur auch als SIMD/Dataflow-Architektur bezeichnet [Jahn96].

Performanzabschätzungen zur Folge erzielt die NESPINN-Plattform einen Beschleunigungsfaktor von 20 im Vergleich zu der bereits vorgestellten CNAPS (siehe Abschn. "CNAPS" auf Seite 49) mit 256 Prozessoren.

3.4.4.2 MASPINN

Die Emulationsplattform MASPINN (*Memory Optimized Accelerator for Spiking Neural Networks*) [Schö98b] gilt als Nachfolger der NESPINN-Plattform. Die MASPINN-Plattform ist eine Beschleuniger-Platine, die mit 100 MHz getaktet werden kann und aus 2 Berechnungseinheiten besteht: der Connection Unit und der Neuron Unit (siehe Abb. 3-7).

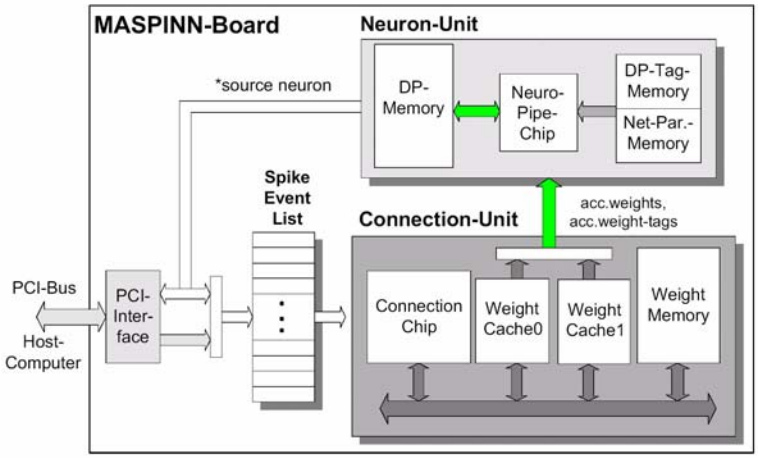


Abb. 3-7: Konzept der MASPINN-Architektur (modifiziert von [Schö00])

Die Connection Unit besteht aus einem Connection Chip, einem Weight Memory, der sämtliche Synapsengewichte des Netzwerks enthält, und 2 Weight Caches, die mit der Neuron Unit verbunden sind. Die Neuron Unit beinhaltet SRAM-Speicher, um gültige Dendritenpotentiale (*DP-Memory*), Markierungen aller Dendritenpotentiale (*DP-Tag-Memory*) im Netzwerk und Netzwerkparameter (*Net-Par.-Memory*) abzulegen.

Neuro-Pipe-Chip

Die Neuronenzustandsberechnung innerhalb der Neuron Unit wird vom Neuro-Pipe-Chip durchgeführt [Schö02a]. Diese Berechnung erfolgt in einer Pipeline-Struktur in 5 Prozessstufen: Abklingstufe (*Decay*), Ausbreitungsstufe (*Propagate*), Filterstufe (*HOF = High Order Filter*), Sortierstufe (*Sorting*) und Ausgangsstufe (*PE Output*). In der Abklingstufe werden durch 2 parallele Datenpfade mit jedem Takt Dendritenpotentiale und die zugehö-

rigen Markierungen aus dem DP-Memory bzw. aus dem DP-Tag-Memory gelesen (siehe Abb. 3-7). Relevante Dendritenpotentiale werden in der Ausbreitungsstufe weiterverarbeitet, indem akkumulierte Gewichte aus dem Weight Cache mit dem entsprechenden Dendritenpotential nach Gl. 3-5 auf Seite 43 addiert werden. Die berechneten Dendritenpotentiale werden schließlich in der Filterstufe kaskadiert, um Filterfunktionen höherer Ordnung zu modellieren. In der folgenden Sortierstufe werden die Dendritenpotentiale eines Neurons sortiert, damit diese von einer der insgesamt 2 PEs verarbeitet werden können. In der abschließende Ausgangsstufe werden die Membranpotentiale jedes Neurons ermittelt und festgestellt, ob das Neuron einen Puls generiert und somit in die Pulsliste eingetragen werden muss.

Als Maßnahmen zur Simulationsbeschleunigung wurden bei der MASPINN-Plattform das Konzept der Weight Caches (siehe auch Abschn. "Weight Cache" auf Seite 43) in der Connection Unit und das Konzept der markierten Potentialspeicherung (siehe auch Abschn. "Markierte Potentialspeicherung" auf Seite 42) in der Neuron Unit eingeführt. Innerhalb des Neuro-Pipe-Chips führen die Konzepte der Daten-Preanalyse (*Data Preanalysis*) und der globalen Inhibition (*Global On-Chip Inhibition*) zu weiteren Erhöhung der Simulationsperformanz. Mit der Daten-Preanalyse sollen irrelevante Berechnungen ermittelt werden, die bei dem Marburger Neuronenmodell gegeben sind, wenn das Feeding-Potential den Wert Null annimmt. In diesem Fall kann auf die Berechnung des Linking-Potentials verzichtet werden, welches ansonsten mit dem Feeding-Potential multipliziert wird (siehe auch Gl. 2-12 auf Seite 18). Eine globale Inhibition entspricht zwar nicht den Beobachtungen in der Biologie, aber bewirkt eine effiziente Kontrolle der Netzwerkaktivität, indem ein einziges Neuron einen “Winner-Take-All”-Mechanismus bewirkt [Hayk99].

Der Beschleunigungsfaktor einer MASPINN-Plattform für ein einschichtiges Netzwerk mit 10^6 Neuronen wird mit dem Faktor 100 im Vergleich zu einer Workstation mit einer Taktfrequenz von 500 MHz angegeben [Schö02a].

3.4.4.3 SP²INN

Die Emulationsplattform SP²INN (*Synapse Plasticity in Spiking Neural Networks*) [Mehr03] stellt die Synapsenplastizität bei der PCNN Simulation in den Vordergrund. Die Beschleuniger-Platine besteht aus mehreren externen Speichern und einem Neuroprozessor, der durch einen modularen Aufbau von Synapsen-, Integrier-, Feuer- und Axon-Einheit versucht, einen biologienahen Neuronenaufbau zu realisieren (siehe Abb. 3-8a).

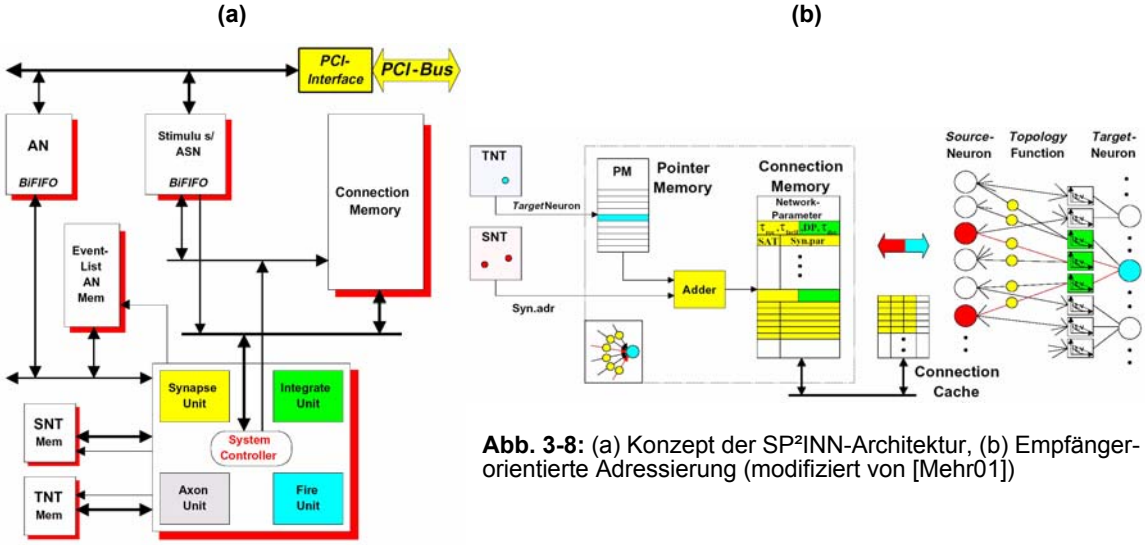


Abb. 3-8: (a) Konzept der SP²INN-Architektur, (b) Empfängerorientierte Adressierung (modifiziert von [Mehr01])

Die Verbindung zu einem Host-Rechner wird durch eine PCI-Schnittstelle (PCI = *Peripheral Component Interconnect*) gewährleistet, an der 2 bidirektionale FIFO-Speicher (FIFO = *First In First Out*) mit jeweils 512×24 Bit angeschlossen sind, die dem Senden und Empfangen von pulsenden oder erregten Neuronen sowie der Aktualisierung des Eingangsreizes dienen. Ein SDRAM-DIMM-Speicher der Größe 64 M×36 Bit stellt den

Verbindungsspeicher (*Connection Memory*) dar. In diesem Verbindungsspeicher ist eine Zeigertabelle (*Pointer Memory*) zusammen mit den Eingangsreizen, den Netzwerkparametern (Topologieinformationen) und den Neuronenparametern (Dendritenpotentiale und Synapsengewichte) abgelegt. Eine zusätzliche Ereignisliste (256 K×24 Bit) ist für die Speicherung von verzögert pulsenden Neuronen vorgesehen. Für jedes pulsende Neuron werden bis zu 2 Verzögerungszeiten berücksichtigt, die maximal 32 Zeitscheiben betragen können. Aus diesem Grund ist diese Liste in 32 Blöcke unterteilt, in denen jeweils 2^{13} Neuronen einbezogen werden können. Schließlich werden vom Neuroprozessor noch 2 Markierungsspeicher für Quell-Neuronen (SNT = *Source Neuron Tag*) und für Ziel-Neuronen (TNT = *Target Neuron Tag*) angesteuert. Der TNT-Speicher (32 K×32 Bit) wird aufgrund der 2 zu berücksichtigenden Verzögerungszeiten in 2 Blöcke gleicher Größe aufgeteilt. Die Größe eines solchen Verzögerungsblocks bestimmt die maximale Anzahl von $2^{14} \cdot 2^5 = 2^{19}$ Neuronen, die von der SP²INN-Plattform berechnet werden können. Der SNT-Speicher ist im Vergleich zum TNT-Speicher mit der doppelten Speicherkapazität ausgelegt. Der Grund hierfür ist, dass in der aktuellen Zeitscheibe *n* der Pulsstatus der beiden Verzögerungsblöcke gelesen wird (Tags werden in Neuronenadressen umgewandelt), während zeitgleich die beiden Verzögerungsblöcke für die nächstfolgende Zeitscheibe *n+1* beschrieben werden (Neuronenadressen werden in Tags umgewandelt). In der aktuellen Zeitscheibe *n+1* werden nun diese Verzögerungsblöcke gelesen, während die Verzögerungsblöcke, die in der Zeitscheibe *n* gelesen wurden, für die nun folgende Zeitscheibe *n+2* aktualisiert werden.

Empfängerorientierte Adressierung

Aufgrund der implementierten Synapsenplastizität erfolgt eine empfängerorientierte Adressierung von Neuronen, die in Abb. 3-8b illustriert ist. Ein Ziel-Neuron wird aus dem TNT-Speicher gelesen, und die generierte Neuronenadresse zeigt auf eine Speicherzelle der Zeigertabelle innerhalb des Verbindungsspeichers. In dieser Speicherzelle ist eine Adresse hinterlegt, die wiederum auf den ersten Neuronenparameter des Ziel-Neurons zeigt. Zu dieser Adresse werden die Synapsenadressen von den pulsenden Quell-Neuronen addiert, die durch Lesen des SNT-Speichers ermittelt werden können. Somit ist es möglich auf die Parameter der Synapsen zuzugreifen, die einen Puls übertragen und die durch den Puls eines Quell-Neurons das resultierende Dendritenpotential mit beeinflussen. Für die Synapsenplastizität ist der Zeitparameter *SAT* (*Spike Arrival Time*) der Synapsen, der den letzten Pulsübertragungszeitpunkt darstellt, und der Zeitparameter *FT* (*Fire Time*) der Ziel-Neuronen, der den letzten Pulszeitpunkt darstellt, ausschlaggebend. Treten die Pulse von Quell- und Ziel-Neuron innerhalb eines bestimmten Zeitfensters Δt auf, findet eine Adaption der Synapsen statt. Wenn die Bedingung $0 < (FT\text{-}SAT) < \Delta t$ erfüllt ist, wird die Synapse verstärkt, wenn die Bedingung $0 > (FT\text{-}SAT) > -\Delta t$ erfüllt ist, wird sie abgeschwächt.

Der Beschleunigungsfaktor einer SP²INN-Plattform für ein einschichtiges Netzwerk mit 2^{18} Neuronen mit jeweils 81 exzitatorischen und 80 inhibitorischen Synapsen wird mit 35 im Vergleich zu einer Workstation mit einer Taktfrequenz von 500 MHz angeben [Mehr03].

3.4.5 FPGA basierte Emulation

Den vielversprechendsten Ansatz für die digitale Emulation von Neuronenmodellen stellen heutige FPGAs dar [Hamm03]. Die Emulation von großen neuronalen Netzen erfordert eine Vielzahl von Off-Chip-Speichern. Eine Beschleuniger-Platine mit mehreren FPGAs ist in der Lage durch den parallelen Zugriff auf mehrere Off-Chip-Speicher eine PCNN Simulation zeiteffizienter durchzuführen als ein einzelner leistungsfähiger Universalrechner. Im folgenden werden 3 FPGA basierte Emulationsplattformen mit zunehmender Leistungsfähigkeit vorgestellt. Die Steigerung der Leistungsfähigkeit dieser Emulationsplattformen beruht auf der technologischen Entwicklung von FPGAs, die in den letzten 10 Jahren stattgefunden hat. In dieser Entwicklungsdekade sind FPGAs mit kontinuierlich zunehmender Integrationsdichte und Verarbeitungsperformanz realisiert worden.

3.4.5.1 SPIKE128K

Die Emulationsplattform SPIKE128K [Fran99] ist ebenfalls für das Marburger Neuronenmodell ausgelegt und wird mit einer Taktfrequenz von 10 MHz betrieben. Für die PCNN Emulation werden für jedes Neuron 5 Variablen berücksichtigt: 2 exzitatorische Dendritenpotentiale (*EP1* und *EP2*, jeweils 14 Bit), ein inhibitorisches Dendritenpotential (*IP*, 14 Bit), ein Linking-Potential (*LP*, 16 Bit) und eine dynamische Schwelle (*DT*, 16 Bit). Die Architektur der SPIKE128K-Plattform besteht aus 5 Modulen: dem Initialisierungs-, Neuronen-, Kommunikations-, Topologie- und Lernmodul (siehe Abb. 3-9).

Das Initialisierungsmodul dient als Kommunikationsschnittstelle zum Host-Rechner über einen VME-Bus und ist für die Speicherinitialisierung der SPIKE128K-Plattform verantwortlich.

Pulsvektor/-maske

Die Neuronenzustandsberechnung wird vom Neuronenmodul durchgeführt, welches aus einem Basismodul und 6 Submodulen besteht. Das Basismodul ist mit 2 unterschiedlichen FPGAs von Xilinx (XC4008 unhd XC3190) bestückt, und jedes Submodul besteht aus einem FPGA von Xilinx (XC3190), in dem digitale Komponenten, wie Addierer, Multiplexer und Schieberegister integriert sind, und 3 externen Speichern für Variablen (128 K×16 Bit), Parameter (128 K×16 Bit) und Look-Up-Tabellen (256 K×16 Bit). Diese Speicher bilden den Neuronenzustandsspeicher und sind so dimensioniert, dass maximal 128 K Neuronen für die Simulation berücksichtigt werden können. 5 Submodule berechnen jeweils eine der 5 Variablen eines Neurons. Das sechste Submodul ist für die Berechnung einer verzögerten Pulsgenerierung zuständig, welche eine axonale Verzögerung bei pulsenden Neuronen modelliert. Dazu werden 2 Variablen à 16 Bit benötigt: der Pulsvektor (*Spike Vector*) und die Pulsmaske (*Spike Mask*). Die Pulsmaske bleibt während der Simulation konstant und bestimmt durch Setzen eines einzigen Bits die zeitliche Verzögerung eines Pulses, die maximal 15 Zeitscheiben betragen kann. Werden mehrere Bits in der Pulsmaske gesetzt, kann ein explosiv pulsendes Neuron modelliert werden. Immer wenn das Membranpotential *MP* eines Neurons den Schwellwert *DT* erreicht, wird der Pulsvektor des Neurons gelesen, mit der Pulsmaske verodert, um 1 Bit verschoben und wieder zurückgeschrieben. Wenn das unterste Bit des Pulsvektors gesetzt ist, wird ein Puls generiert. Das Basismodul des Neuronenmoduls berechnet das Membranpotential jedes Neurons, welches sich aus den 5 Variablen des Neurons zusammensetzt. Die Neuronenzustandsberechnung, die die Adressierung des Neurons, das Lesen der Variablen, die Berechnung des Membranpotentials, den Vergleich mit dem Schwellwert und die Aktualisierung sowohl des Schwellwertes und als auch des Pulsvektors umfasst, wird in einer 10-stufigen Pipeline-Struktur realisiert. Die Bereitstellung von 6 Submodulen realisiert eine teilweise parallele Verarbeitung.

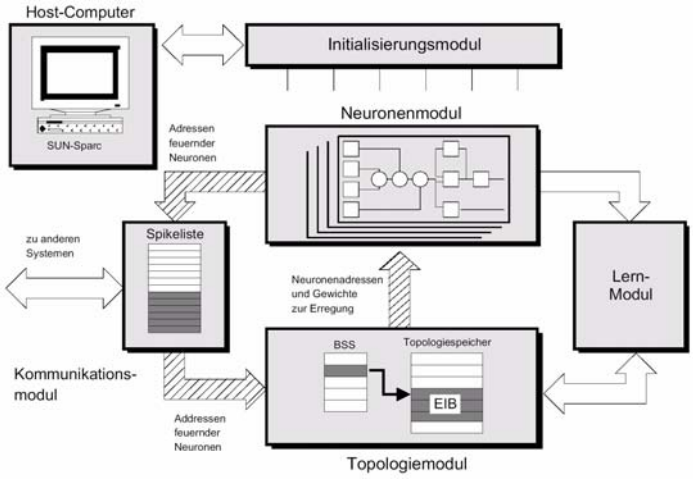


Abb. 3-9: Architektur der SPIKE128K-Plattform [Wolf01]

Das Kommunikationsmodul besteht aus 2 Neuronenlisten, die in Form von FIFOs (jeweils 32 K×27 Bit) realisiert sind. Die Erregungsliste enthält sämtliche Neuronen, die

das PCNN erregen. Die Pulsliste beinhaltet alle Neuronen, die aufgrund der Neuronenzustandsberechnung einen Puls generieren.

Das Topologiemodul stellt anhand von DRAM-Speichern einen BSS (Block-Start-Speicher) der Größe 2 M×32 Bit und einen Topologiespeicher der Größe 4 M×4×32 Bit bereit (siehe Abb. 3-9). Zusätzlich notwendige Logik für eine Pipeline-Steuerung ist im Topologiemodul mit einem weiteren FPGA von Xilinx (XC3190) realisiert. In der Netzwerktopologiephase werden die Neuronenadressen von der Erregungs- und der Pulsliste gelesen. Die Neuronenadressen zeigen auf eine Speicherzelle im BSS, wo für jedes Neuron ein Zeiger hinterlegt ist, der wiederum auf einen Speicherbereich im Topolgiespeicher zeigt. Der Parametersatz, der für jedes Neuron im Topologiespeicher abgelegt ist, wird als EIB (Erregungs-Informations-Block) bezeichnet. Ein Datenwort im EIB setzt sich aus der Adresse des Empfangsneurons (19 Bit), dem Synapsengewicht (9 Bit) und Kontrollinformationen (4 Bit) zusammen. Anhand der Adresse des Puls-empfangenden Neurons können im Neuronenzustandspeicher entsprechenden Teilpotentiale, die sich auf dem Neuronenmodul befinden, aktualisiert werden. Aufgrund der Topologiespeichergröße können somit maximal 16 M Synapsen abgelegt werden.

Lernschwelle

Das Lernmodul ermöglicht die Adaption von Synapsengewichten. Hierfür muss der Neuronenzustandsspeicher um 2 Variablen erweitert werden: die Lernschwelle (*LT*, 16 Bit) und ein Kontrollbit, welches anzeigt, ob das Neuron lernbereit ist (*RFL = Ready For Learning*). Wenn das Empfangsneuron lernbereit ist und einen Puls erhält, wird das Synapsengewicht *w* um einen Term *f(w)* erhöht. Dieser Term wird von einer Look-Up-Tabelle gelesen und ist von der Größe des Synapsengewichts abhängig (*Learning Term*) [Schä99]. Bleibt das Neuron über eine feste Anzahl von Zeitscheiben lernbereit, wird das Synapsengewicht *w* zusätzlich um einen Term *g(w)* erniedrigt, um eine Gewichtssättigung zu vermeiden (*Forgetting Term*). Die Adaption der Synapsengewichte erfolgt immer 2 Zeitscheiben später, nachdem die Pulse generiert wurden. In der Zeitscheibe n-1 werden die Pulse in der Netzwerktopologiephase durch die Aktualisierung der Neuronenzustandspeicher entsprechend verteilt. In der Zeitscheibe n werden in der Neuronenzustandsphase die Membranpotentiale mit der Lernschwelle verglichen und entschieden, ob Neuronen lernbereit sind. In der Neuronenzustandsphase der nächsten Zeitscheibe n+1 erfolgt schließlich die Adaption der Synapsen, indem eine Kopie der Erregungs- und Pulsliste aus der Zeitscheibe n-1 herangezogen wird. Anhand dieser Liste ist es möglich die Synapsengewichte aller lernbereiten Empfangsneuronen zu adaptieren.

Die Beschleunigung der SPIKE128K-Plattform für die Emulation von 128 K Neuronen wird mit einem Faktor von 333 im Vergleich zu einer SPARC20-Workstation angegeben. Dabei wird die Workstation mit einer Prozessor-Frequenz von 50 bis 125 MHz betrieben, und die Netzwerkaktivität des PCNNs beträgt 0,01 % .

3.4.5.2 RAPTOR2000

Die Emulationsplattform RAPTOR2000 [Porr02] stellt ein flexibles und modulares Rapid-Prototyping-System dar. Die Flexibilität im Vergleich zur SPIKE128K-Plattform basiert auf der Tatsache, dass mit der RAPTOR2000-Plattform unterschiedliche neuronale Implementierungen, wie selbstorganisierende Karten (*SOM = Self-Organising Feature Map*), assoziative Speicher und RBF-Netzwerke (*RBF = Radial Basis Function*), realisiert werden können. Die Modularität ist dadurch begründet, dass neben einer Basisplatine (*Motherboard*) 6 ASMs (*Application Specific Modules*) als eigenständige Zusatzplatinen angeschlossen werden können (siehe Abb. 3-10).

Die Basisplatine stellt die Kommunikationsinfrastruktur der ASMs über einen Sendebus (*Broadcast Bus*) und einen lokalen Bus bereit und die des Host-Rechners über eine PCI-Schnittstelle. Der Sendebus dient ausschließlich der simultanen Kommunikation zwischen den ASMs, und der lokale Bus ist für die interne Kommunikation zu anderen Modulen oder Komponenten und für die externe Kommunikation zu einem Host-Rechner oder anderen PCI-Komponenten zuständig. Ein ASM besteht aus einem FPGA von Xilinx (XCV1000), das an einen 128 MB großen SDRAM-Speicher angeschlossen ist und 4 externe Schnitt-

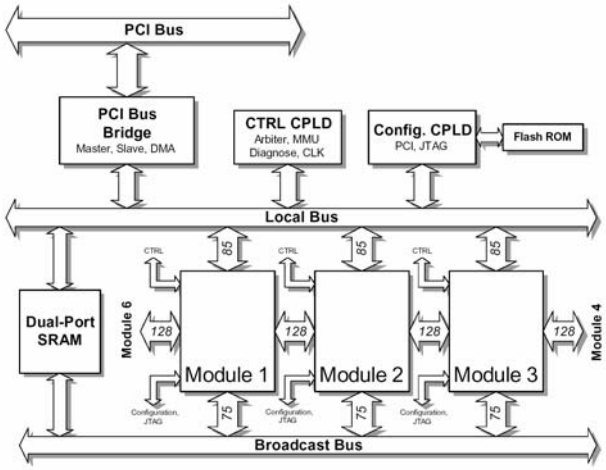


Abb. 3-10: Architektur der RAPTOR2000-Plattform [Porr02]

stellen zur Verfügung hat. Jeweils eine externe Schnittstellen dient dem Anschluss an den Sendebus und an den lokalen Bus und die verbleibenden 2 Schnittstellen à 128 Pins stehen für die direkte Kommunkation zu anderen ASMs zur Verfügung.

Dynamische Rekonfiguration

Eine Besonderheit der RAPTOR2000-Plattform ist, dass eine dynamische Rekonfiguration unterstützt wird. Dies bedeutet, dass während des Betriebs über den lokalen Bus eine Rekonfiguration eines ASMs erfolgen kann, die weniger als 20 ms in Anspruch nimmt [Porr02].

Für die Simulation von 250×250 Neuronen, die in einer selbstorganisierenden Karte eingebettet sind, können mit Hilfe von 4 ASMs 81 PEs implementiert werden, die bei einer Taktfrequenz von 65 MHz operieren und jeweils mit einem internen SRAM-Speicher von 2 KB ausgestattet sind. Die RAPTOR2000-Plattform ist in der Lage für diese Implementierung eine Performanz von $4400 \cdot 10^6$ CPS zu erzielen. Im Vergleich zu einer Implementierung in Software (AMD-Athlon CPU mit 800 MHz), die $80 \cdot 10^6$ CPS erreicht, ergibt sich somit ein Beschleunigungsfaktor von 55.

3.4.5.3 BEE

Die Emulationsplattform BEE (*Berkeley Emulation Engine*) [Chan03a] ist primär für das Anwendungsgebiet der Software konfigurierbaren Mobilfunksysteme (SDR = *Software Defined Radio*) konzipiert. Aber aufgrund der hohen parallelen Architektur und der hohen Speicherbandbreite ist die BEE-Plattform ebenfalls für eine PCNN Emulation geeignet. Eine SDR-Architektur ermöglicht eine flexible und schnelle Einführung von Mobilfunkdiensten, -anwendungen und -funktionen und besteht aus den folgenden 3 Komponenten: analoge Front-Ends (UWB AD/DAC = *Ultra-Wide Band Analog-Digital/Digital-Analog Conversion*, MCMA = *Multi-Channel-Multi-Antenna*), digitale Front-Ends und einer Basisband-Signalverarbeitung, die von einem Universalrechner durchgeführt wird. Alle 3 Komponenten sind durch Software programmierbar, und zusätzlich ist die Hardware des digitalen Front-Ends rekonfigurierbar.

BEE Processing Unit

Das digitale Front-End der BEE-Plattform wird als BPU (*BEE Processing Unit*) bezeichnet (siehe Abb. 3-11). Es besteht aus einer Prozessor-Platine (MPB = *Main Processing Board*), 8 Riser-IO-Platinen für analoge Front-Ends, die an das MPB über die mehr als 2000 externen IO Pins angeschlossen werden können, und einer Kommunikations-Platine (SBC = *Single Board Computer*). Die MPB-Platine ist eine 26-lagige und 58×53 cm große Platine, die 20 FPGAs von Xilinx (XCV2000E) beinhaltet. 16 FPGAs stehen als PEs zur Verfügung und sind über 60 IO-Pins mit einem 1 MB großen externen SRAM-Speicher verbunden. Diese 16 PE-FPGAs sind als 4×4 großes Rechteck auf der MPB-Platine angeordnet (siehe Abb. 3-11a) und über 48 Bit-breite Datenbusse mit den 8 nächsten benachbarten FPGAs verbunden (siehe Abb. 3-11b). Dies bedeutet, dass die 4 inneren PE-FPGAs keine Verbindung zu den externen IO-Pins haben. 4 FPGAs, die als XBAR

bezeichnet werden und ebenfalls keine Verbindungen zu externen IO-Pins haben, stehen für die interne Kommunikation zwischen den FPGAs zur Verfügung und verbinden eine Gruppe von 4 PE-FPGAs über 58 Bit-breite Datenbusse. Zusätzlich verfügen die XBAR-FPGAs untereinander über 84 Bit-breite Kommunikationsbusse (siehe Abb. 3-11a).

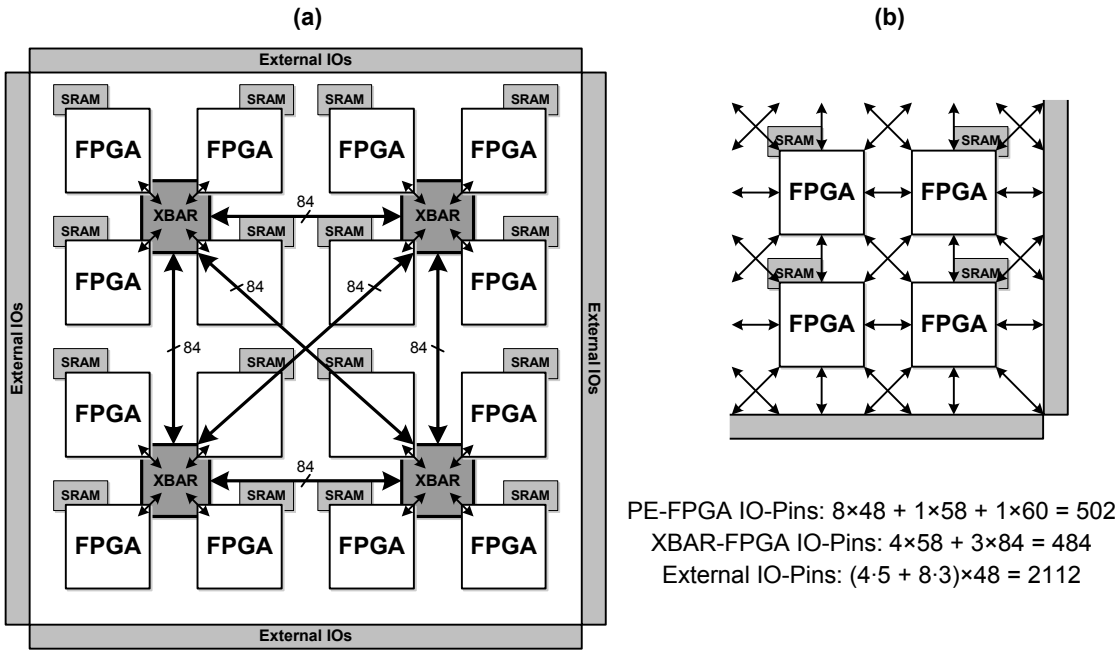


Abb. 3-11: Architektur der BEE Processing Unit: (a) Verbindungsstruktur der 4 XBAR-FPGAs, (b) Verbindungsstruktur der 16 PE-FPGAs (8-Way Mesh) (basierend auf [Chan03a] [Chan03b])

Die SBC-Platine ist für die Programmierung der FPGAs und das Power Management verantwortlich. Diese Aufgaben werden durch einen mit 206 MHz getakteten StrongARM-Prozessor von Intel durchgeführt, auf dem das Betriebssystem Linux läuft und der auf einen 32 MB großen SDRAM-Speicher und einen 16 MB großen Flash-Speicher zugreifen kann.

Für eine Abschätzung der Gesamtperformanz wurde der BCJR-Algorithmus (BCJR = *Bahl-Cocke-Jelinek-Raviv*) implementiert, der wie der Viterbi-Algorithmus zur Fehlerkorrektur bei einer Funkübertragung verwendet wird. Die Decodierung von jeweils 10^9 Eingangs-Vektoren nach dem BCJR-Algorithmus konnte von den insgesamt 16 FPGAs bei einer Taktfrequenz von 13,2 MHz in 20,2 Minuten (1212 Sekunden) durchgeführt werden. Ein Software Simulation (Intel-Pentium-4-Xean CPU mit 1,8 GHz) benötigt für die Decodierung von 10^3 Eingangsvektoren 70 Sekunden. Daraus ergibt sich ein Beschleunigungsfaktor von $16 \cdot 10^6 \cdot 70/1212 \approx 10^6$ [Chan03b].

3.4.6 Effizienzbetrachtung digitaler Simulationsplattformen

Die Effizienz (Wirkungsgrad) ist definiert als das Verhältnis von Nutzen zu Aufwand. Bei der Effizienzbetrachtung von ASIC Architekturen wird Effizienz als Quotient aus Performanz (definiert als Datendurchsatz, der von der Pipelinestufenanzahl und dem Parallelisierungsgrad abhängt) pro Kosten (bestehend aus Schaltkreisgröße, Leistungsverbrauch und Fertigungskosten) angegeben [Jesc05].

Für eine qualitative Effizienzbetrachtung digitaler Simulationsplattformen (Neurocomputer) sind neben den Gesichtspunkten Kosten und Performanz die Gesichtspunkte Flexibilität und emulierbare PCNN Größe heranzuziehen, die ebenfalls bei der Gegenüberstellung von analoger und digitaler PCNN Emulation in Tab. 3-1 auf Seite 46 verwendet wurden. Diese Effizienzbetrachtung beruht auf den Erkenntnissen der in dieser Arbeit vorgestellten digitalen Simulationsplattformen und soll die Motivation für den Entwurf von FPGA basierten Emulationsplattformen demonstrieren. Die digitalen Simulationsplattformen wurden nach Abb. 3-5 auf Seite 44 in 5 Gruppen, von Universalrechner bis

Beschleuniger-Platine mit Neuroprozessoren, unterteilt, die in Abb. 3-12 nochmals aufgelistet sind.

Kosten/ Performanz

In Abb. 3-12a sind die Kosten und die Performanz gegenübergestellt. Bei der Effizienzbetrachtung werden als Kosten sowohl die Anschaffungskosten als auch die einmaligen Entwicklungskosten (*NRE* = *Non-Recurring Engineering*) einer Simulationsplattform betrachtet. Die Performanz bezieht sich auf die erzielbare Simulationsgeschwindigkeit. Ein Universalrechner (Kreis) stellt die geringste Performanz bei den niedrigsten Anschaffungskosten (höchste Kosteneffizienz) von unter 1500 US-Dollar dar. Anhand einer Beschleuniger-Platine mit Neuro-Chips (umgedrehtes Dreieck, siehe auch Abschn. "BIONIC" auf Seite 48) ist man in der Lage eine massive parallele Realisierung eines PCNNs durchzuführen, was einer hohen Performanz aber einem hohen Kostenaufwand entspricht, da NRE-Kosten für die HW-Entwicklung anfallen (siehe auch [Euro05]). Höchstleistungsrechner wie BlueGene/L (Quadrat, siehe auch Abschn. "Software Simulation" auf Seite 44) bieten die höchste Performanz von über 70000 GFlops bei den höchsten Anschaffungskosten von $1{,}5 \cdot 10^6$ US-Dollar [Blue04b], was einem Faktor 1000 im Vergleich zu einem Universalrechner entspricht.

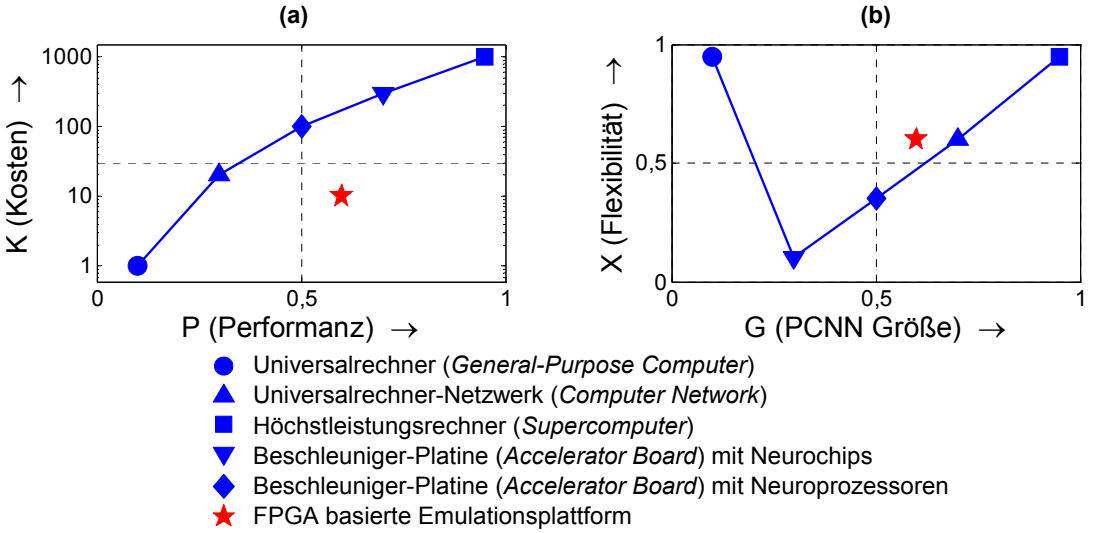


Abb. 3-12: Betrachtung unterschiedlicher digitaler Simulationsplattformen: (a) nach Kosten K (Entwicklungsaufwand) und Performanz P, (b) nach Flexibilität X (Programmierbarkeit) und simulierbarer Größe (Neuronen) und Dichte (Synapsen) pulscodierter neuronaler Netze

Flexibilität/ PCNN Größe

In Abb. 3-12b sind die beiden Eigenschaften Flexibilität und PCNN Größe dargestellt. Die Flexibilität beinhaltet Programmierbarkeit (durch Software) und Rekonfigurierbarkeit (der Hardware) einer Simulationsplattform. Die PCNN Größe weist darauf hin, welche Netzwerkgrößen (Anzahl von Neuronen und Synapsen) berücksichtigt werden können. Ein Universalrechner bietet die höchste Flexibilität für Simulationen von allerdings sehr kleinen Netzwerken. Eine Beschleuniger-Platine mit Neurochips kann etwas größere Netzwerke behandeln, ist aber auf eine spezielle Neuronenmodellierung fixiert. Bei der Verwendung von Neuroprozessoren (Raute, siehe auch Kapitel 3.4.4 auf Seite 50) können aufgrund der Programmierbarkeit und der sequentiellen Verarbeitung der Grad an Flexibilität und die emulierbare PCNN Größe erhöht werden. Für Software-programmierbare Universalrechner-Netzwerke (Dreieck) und Höchstleistungsrechner steigt die Flexibilität und die emulierbare PCNN Größe weiter an, da keine spezialisierte HW notwendig ist und größere Speicher-Ressourcen für die Netzwerkparameter zur Verfügung stehen.

FPGA basierte Emulationsplattformen bieten geringe Entwicklungskosten (siehe auch Abb. 1-3 auf Seite 4) und eine sehr hohe Performanz, da sie zum einen Operationsfrequenzen wie bei Halb-Kunden Entwürfen erzielen und zum anderen aufgrund des großen Chip-Durchmessers eine hohe Speicherbandbreite zur Verfügung stellen. In Abb. 3-12a sind diese Emulationsplattformen somit im optimalen Bereich des unteren rechten Rechtecks einzuordnen. Die Flexibilität von FPGA basierten Emulationsplattformen wird durch die Rekonfigurierbarkeit im Vergleich zu Halb-Kunden Entwürfen zusätzlich erhöht, und bei ausreichender Speicherkapazität lassen sich auch große PCNNs simulieren. Aus

diesen Gründen sind diese Emulationsplattformen in Abb. 3-12b wiederum im effizientesten Bereich des oberen rechten Rechtecks einzustufen.

η für IC-Architektur

Für eine quantitative Effizienzbetrachtung digitaler Simulationsplattformen lässt sich die Effizienz η_{IC} einer digitalen integrierten Schaltungsarchitektur nach [Klar04] in Gl. 3-6 heranziehen,

$$\eta_{IC} = \frac{F \cdot D}{A} \tag{3-6}$$

wobei *F* dem Aufwand einer Operation entspricht, *D* die Durchsatzrate und *A* die verwendete Chip-Fläche der Schaltungsarchitektur darstellt. Bei einer Schaltungsarchitektur, die eine Fließbandverarbeitung realisiert, entspricht die Durchsatzrate der Taktfrequenz, mit der die Schaltung betrieben wird. Wenn für den portablen Betrieb die Verlustleistung eine Rolle spielt, wird bei der Effizienzbetrachtung nach Gl. 3-6 im Nenner zusätzlich die Verlustleistung *V* berücksichtigt, da eine Verringerung der Verlustleistung eine Erhöhung von η_{IC} zur Folge hat. Die Effizienz formuliert folglich für die Implementierung einer komplexen Operation den immer währenden Zielkonflikt (*Trade-Off*) zwischen Operationsfrequenz (*Speed*), erforderlicher Chip-Fläche (*Area*) und Verlustleistung (*Power*).

Bei der Effizienzbetrachtung digitaler Simulationsplattformen wird vom gleichen PCNN Modell ausgegangen, so dass der Aufwand *F* als konstant betrachtet werden kann. Die Durchsatzrate *D* lässt sich als Datenübertragungsrate bzw. Speicherbandbreite, also aus dem Produkt von Takrate *f* und Datenbitbreite *B* (zum Gewichts- und Parameterspeicher) darstellen (siehe auch Kapitel 3.2.5 auf Seite 38). Die Effizienz eines Universalrechners für die PCNN Emulation kann folglich anhand Gl. 3-7 ausgedrückt werden.

$$\eta_{\bullet} = \frac{f \cdot B}{A} \tag{3-7}$$

Bei der PCNN Emulation durch ein Universalrechner-Netzwerk oder einen Höchstleistungsrechner werden Prozessoren mit der gleichen Taktrate wie bei einem Universalrechner verwendet (der Supercomputer BlueGene/L verwendet Prozessoren mit 0,7 GHz, siehe auch Abschn. "Software Simulation" auf Seite 44). Die Chip-Fläche und die Speicherbandbreite erhöhen sich entsprechend der Anzahl *N* an Prozessoren. Somit entspricht die Effizienz eines Universalrechner-Netzwerks oder eines Höchstleistungsrechners dem eines Universalrechners, wie in Gl. 3-8 dargestellt.

$$\eta_{\blacktriangle} = \eta_{\blacksquare} = \frac{f \cdot B \cdot N}{N \cdot A} = \eta_{\bullet} \tag{3-8}$$

Beschleuniger-Platinen mit Neurochips bzw. Neuroprozessoren (mit Taktfrequenzen von 100-300 MHz) haben eine um den Faktor 10 reduzierte Taktrate im Vergleich zu einem Universalrechner (mit Prozessoren, die bei 1-3 GHz operieren). Um diesen Umstand auszugleichen, wird durch die spezielle Hardware-Implementierung eine parallele Verarbeitung der PCNN Emulation erzielt, die die Emulation mindestens um den Faktor der reduzierten Taktrate beschleunigen muss. Die Effizienz von Beschleuniger-Platinen mit Neurochips und Neuroprozessoren ist in Gl. 3-9 dargestellt,

$$\eta_{\blacktriangledown} = \eta_{\blacklozenge} = \frac{f/10 \cdot B \cdot N_{10}}{A} \geq \eta_{\bullet} \quad | \; N_{10} \geq 10 \tag{3-9}$$

wobei N_{10} den Beschleunigungsfaktor darstellt, der mindestens einer Anzahl von 10 Prozessor-Elementen (N = 10) entsprechen muss.

FPGA basierte Emulationsplattformen (mit Taktraten von 50-150 MHz) haben im Vergleich zu Beschleuniger-Platinen mit Neurochips bzw. Neuroprozessoren eine um den Faktor 2 reduzierte Taktrate. Allerdings kann dieser Umstand durch eine erhöhte Speicherbandbreite bei FPGAs (in Gehäusen mit mehr als 1000 nutzbaren IO Pins) mehr als neutralisiert werden. Die Effizienz FPGA basierter Emulationsplattformen kann anhand Gl. 3-7 ausgedrückt werden,

$$\eta_{\bigstar} = \frac{f/20 \cdot n \cdot B \cdot N_{10}}{A} \geq \eta_{\bullet} \quad | \; n \geq 2, N_{10} \geq 10 \tag{3-10}$$

wobei n den Faktor der erhöhten Speicherbandbreite darstellt.

Diese quantitative Effizienzbetrachtung demonstriert, dass nur die Entwicklung von FPGA basierten Emulationsplattformen, die durch eine spezielle Hardware-Implementierung eine parallele Verarbeitung der PCNN Emulation erzielen ($N_{10} \geq 10$) und im Vergleich zu Beschleuniger-Platinen eine mehr als doppelt so hohe Speicherbandbreite zur Verfügung stellen ($n \geq 2$), für die Beschleunigung einer PCNN Simulation angemessen ist.

Anforderungen

Zusammenfassend lassen sich nun die wesentlichen Anforderungen an Emulationsplattformen benennen, die für die Beschleunigung einer PCNN Simulation notwendig sind:

- Maximal mögliche Flexibilität, die durch Programmierbarkeit anhand von Software, wie bei Universalrechnern, und durch Rekonfigurierbarkeit von Hardware erreicht wird
- Maximal mögliches Verhältnis von Performanz zu Kosten, welches durch den Einsatz von heutigen FPGAs erzielt wird
- Hardware-Entwurf von arithmetischen Einheiten für die Neuronenzustandsberechnung, die anhand eines eigenen Datenkanals einen effizienten Zugriff auf Neuronen- und Synapsenparameter haben
- Ausreichende Speicherkapazität und vor allem Speicherbandbreite, die physikalisch verfügbar ist und nicht wie bei Universalrechnern virtuell verwaltet wird, um große PCNNs simulieren zu können
- Ausreichende Genauigkeit von Festkomma-Arithmetik
- Realisierung von Testkonzepten (Debuggen) und Möglichkeiten zur Netzwerkbeschreibung und Netzwerkkonfiguration für die Inbetriebnahme der Emulationsplattform

Resümee

Im Rahmen dieses dritten Kapitels wurde der Ablauf einer PCNN Simulation und die damit verbundene Aufteilung in Neuronenzustands- und Netzwerktopologiephase vorgestellt. Die bei einer PCNN Simulation auftretenden 5 Flaschenhals-Probleme (Rechenoperationen, Kommunikationsaufwand, Lastverteilung, Speicherkapazität und Speicherbandbreite) wurden diskutiert und bekannte Konzepte zur Simulationsbeschleunigung dargestellt (siehe Kapitel 3.3 auf Seite 39). Der Vergleich von analoger und digitaler Emulation (siehe Tab. 3-1 auf Seite 46) verdeutlicht die Überlegenheit einer digitalen Emulation bezüglich der Flexibilität und simulierbaren PCNN Größe. Die Performanz bei der digitalen Emulation lässt sich durch ein geeignetes Architekturkonzept und durch eine minimal notwendige Genauigkeit der Festkomma-Arithmetik steigern. Insbesondere die Genauigkeit der Festkomma-Arithmetik ist vom verwendeten Neuronenmodell abhängig. Der Überblick von unterschiedlichen digitalen Simulationsplattformen zeigt, dass die digitale Emulation anhand von Voll-Kunden-Entwürfen wie auch von Halb-Kunden-Entwürfen das größte Flaschenhals-Problem der limitierten Speicherbandbreite nicht lösen (siehe Kapitel 3.2.5 auf Seite 38). Aufgrund der Pin-Anzahl, die durch das Chip-Gehäuse bei diesen Entwürfen beschränkt wird, können keine parallel nutzbaren Datenkanäle mit entsprechender Bit-Breite realisiert werden. Dies ist der Hauptgrund warum die digitale Emulation anhand von heutigen FPGAs den vielversprechendsten Ansatz darstellen [Hamm03].

Im folgenden Kapitel wird ein neues Architekturkonzept für eine FPGA basierte Emulationsplattform vorgestellt, das für die Emulation von pulscodierten Neuronen mit adaptiven Synapsengewichten geeignet ist.

KAPITEL

4

SEE: eine FPGA basierte Emulationsplattform

In diesem Kapitel wird das im Rahmen dieser Arbeit konzipierte Simulationsbeschleuniger-System SEE (*Spiking Neural Network Emulation Engine*) vorgestellt, welches ein sehr flexibles Hardware/Software-System für die Emulation von pulscodierten neuronalen Netzen darstellt. Die für die Simulationsbeschleunigung notwendigen Hardware-Module werden durch rekonfigurierbare FPGAs realisiert, und die für die Simulationsablaufsteuerung notwendigen Software-Module werden von programmierbaren CPUs ausgeführt, die sich innerhalb der FPGAs befinden. Aufgrund dieser Eigenschaften stellt SEE eine FPGA basierte Emulationsplattform dar, deren Systemarchitektur nach dem jetzigen Stand der Technik noch nicht existiert. Nachdem die Notwendigkeit für eine neuartige Emulationsplattform herausgestellt wurde, werden die Maßnahmen, die bei SEE für die Beschleunigung der Emulation ergriffen worden sind, vorgestellt. Im weiteren Verlauf wird die Neuronenzustandsberechnung bei SEE diskutiert, bei der das numerische Integrationsverfahren nach Bulirsch-Stoer zum Einsatz kommt. Abschließend wird die Netzwerktopologieberechnung behandelt, die unterschiedliche Netzwerkarchitekturen für das Neuronenmodell mit adaptiven Synapsengewichten berücksichtigen muss.

4.1 Ansatz für eine FPGA basierte Emulationsplattform

Die Notwendigkeit für eine neuartige Emulationsplattform beruht auf der Tatsache, dass existierende akademische und kommerziell erhältliche Prototypenplattformen für die Emulation von großen PCNNs nicht geeignet sind. Aufgrund des Prototypencharakters basieren diese Plattformen auf FPGAs, und man unterscheidet 2 Arten von Prototypenplattformen [Krup00]: Rapid-Prototyping- und ASIC-Prototyping-Plattformen.

Rapid Prototyping

Rapid-Prototyping-Plattformen werden zur Evaluation und Funktionsüberprüfung von eingebetteten Digitalsystemen, die auch als SoCs (*System-on-Chips*) bezeichnet werden, eingesetzt. RAPTOR2000 beispielsweise (siehe Kapitel 3.4.5.2 auf Seite 55) stellt eine solche Plattform im akademischen Bereich dar. Das kommerzielle Angebot an diesen Plattformen ist mittlerweile sehr vielfältig und ausreichend geworden (siehe [Avne05] [Sund05], [XESS05] und [Xili05a]). Rapid-Prototyping-Plattformen basieren meistens auf einem einzigen FPGA, welches auf einige diskrete Speicher, wie SRAM und SDRAM, zugreifen kann. Zusätzlich können vom FPGA gängige Schnittstellen, wie USB (*Universal Serial Bus*) und Ethernet, angesteuert werden, um mit einem Host-Rechner zu kommunizieren. Die Leiterplatinen von Rapid-Prototyping-Plattformen können aufgrund der geringen Anzahl an notwendigen Schaltkreisen sehr kompakt realisiert werden. Dies ermöglicht die Verwirklichung von kurzen Leiterbahnen, was zu geringen Leitungsverzögerungen führt. Dadurch können Operationsfrequenzen im dreistelligen MHz-Bereich erzielt werden. Das Digitalsystem auf einer Rapid-Prototyping-Plattform kann man sich folglich als ein kleines Einprozessor-System vorstellen, dessen Operationsfrequenz um den Faktor 10 kleiner ist im Vergleich zu einem Einprozessor-System, welches dem neuesten Stand der Technik entspricht. Für die Emulation von großen PCNNs besteht bei diesen Plattformen das Problem, dass zum einen keine ausreichende Speicherbandbreite für eine effiziente Simulationsbeschleunigung zur Verfügung steht und dass zum anderen keine ausrei-

chende Speicherkapazität auf der Plattform vorhanden ist, um adaptive Netzwerkparameter, wie Synapsengewichte und Membranpotentiale, zwischenzuspeichern.

ASIC Prototyping

ASIC-Prototyping-Plattformen werden für die Verifikation und für die Implementierung von hochkomplexen und parallel operierenden Digitalsystemen verwendet, da die Verifikation bzw. die Implementierung solcher Systeme auf Einprozessor-System sehr zeitineffizient ist. BEE (siehe Kapitel 3.4.5.3 auf Seite 56) stellt eine solche Plattform im akademischen Bereich dar. Das kommerzielle Angebot an diesen Plattformen ist ebenfalls sehr vielfältig und wird sich zukünftig insbesondere im Bereich der Software konfigurierbaren Mobilfunksysteme weiter ausbauen (siehe [Dini05], [Nall05], [Pent05] und [ProD05]). ASIC-Prototyping-Plattformen basieren meistens auf einer zweistelligen Anzahl von FPGAs, die einen hohen Grad an Parallelverarbeitung ermöglichen. Der Entwurf der Leiterplatinen solcher Plattformen erweist sich als sehr komplex, da viele und vor allem lange Leiterbahnen zur Verfügung gestellt werden müssen, um alle FPGAs miteinander zu verbinden. Dies führt zu großen Leitungsverzögerungen, wodurch nur Operationsfrequenzen im ein- oder zweistelligen MHz-Bereich realisiert werden können. Für die Emulation von großen PCNNs besteht bei diesen Plattformen das Problem, dass ein zeiteffizienter Zugriff auf externe Speicher, die notwendige Netzwerkparameter enthalten, oder auf eine zentral abgelegte Liste mit erregten Neuronen nicht realisiert werden kann. Weiterhin werden die Flaschenhals-Probleme des Kommunikationsaufwands und der Lastverteilung (siehe Kapitel 3.2.2 und Kapitel 3.2.3 auf Seite 37) verschärft, da aufgrund der Vielzahl an FPGAs eine hohe Parallelverarbeitung stattfinden kann.

PCNN Prototyping

Die Emulationsplattform SEE soll als Experimentierplattform für große pulscodierte neuronale Netze (PCNN-Prototyping-Plattform) dienen und stellt einen Kompromiss zwischen einer Rapid-Prototyping- und einer ASIC-Prototyping-Plattform dar. Die effiziente Nutzung von FPGA-Ressourcen ermöglicht einen kompakten Leiterplatinen-Entwurf, der kurze Leiterbahnen realisiert und somit Operationsfrequenzen wie bei Rapid-Prototyping-Plattformen zulässt. Die Bereitstellung einer ausreichend hohen Speicherbandbreite wie bei ASIC-Prototyping-Plattformen ermöglicht eine effiziente Parallelisierung des PCNN Algorithmus. Schließlich kann eine autarke Emulationsplattform nur dann gewährleistet werden, wenn zusätzlich ausreichende Speicherkapazitäten für adaptive Synapsengewichte und Membranpotentiale auf der Plattform vorhanden sind, damit die zeitintensive Kommunikation zu einem Host-Rechner während der Emulation vermieden werden kann.

4.1.1 Vorteile heutiger FPGA-Technologien

Die Emulationsplattform SEE nutzt die Vorteile heutiger FPGA-Technologien, die sich in den letzten 10 Jahren insbesondere bezüglich der Integrationsdichte und der realisierbaren Operationsfrequenzen weiterentwickelt haben.

Glue Logic

Vor 10 Jahren wurden FPGAs aufgrund ihrer geringen Integrationsdichte meist nur zur Realisierung von notwendiger digitaler Kommunikationslogik verwendet, um Voll-Kunden- oder Halb-Kunden-ASICs miteinander zu “verleimen” (*Glue Logic*). Weitere nachteilige Eigenschaften waren die nicht ausreichende Performanz bezüglich der Operationsfrequenz, die hohe Verlustleistung und die Kostenunwirksamkeit für hohe Stückzahlen.

SRAM basiert

Heute hat sich in der FPGA-Technologie die SRAM basierte Konfigurationstechnik gegenüber anderen Methoden, die auf EPROM-, E²PROM-, Flash- oder Fuse-Techniken (*Fusible-Link* bzw. *Antifuse*) beruhen, durchgesetzt [Maxf04]. Somit lassen sich FPGAs als SRAM basierte digitale Schaltkreise betrachten, die im wesentlichen von den beiden Herstellerfirmen Altera (31 % Marktanteil) und Xilinx (49 % Marktanteil) angeboten werden [Reed03].

Logic Cell

Eine logische Zelle bei FPGAs, die bei Altera als LE (*Logic Element*) und bei Xilinx als LC (*Logic Cell*) bezeichnet wird, basiert auf einer LUT (*Look-Up-Table*) mit 4 Eingangssignalen. Anhand dieser LUT lässt sich jede beliebige logische Funktion realisieren. In Abb. 4-1 ist das Konzept einer SRAM basierten LUT-Initialisierung dargestellt. Dabei werden die Eingangssignale als Zeiger eines Multiplexers verwendet. Der Inhalt der

Tabelle in Form von SRAM Zellen entspricht den ermittelten Ausgangswerten der Wahrheitstabelle und wird so arrangiert, dass die Zelle den gewünschten Wert bei allen möglichen Eingangssignalkombinationen anzeigt.

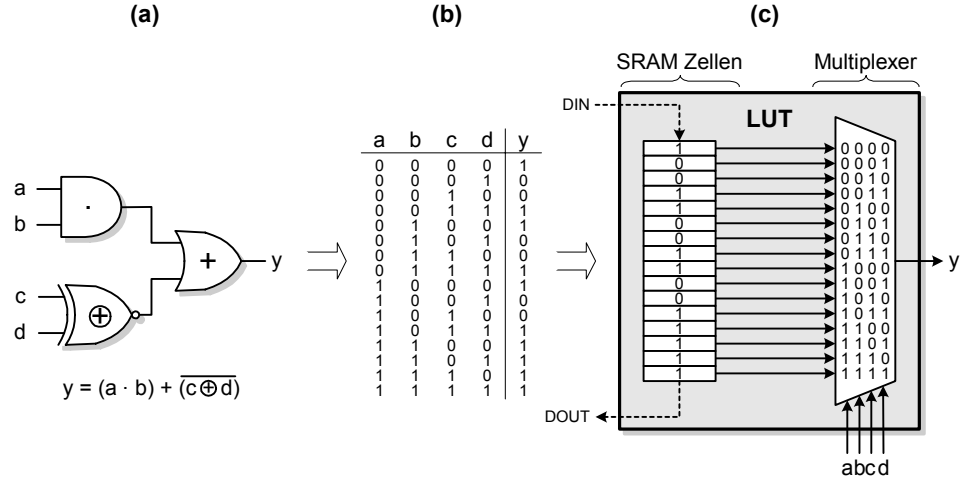


Abb. 4-1: Konzept der LUT-Initialisierung: (a) benötigte Funktion, (b) entsprechende Wahrheitstabelle, (c) realisierte LUT (basierend auf [Maxf04])

Slice

Der Ausgang einer LUT kann zusätzlich mit einem Register versehen werden. Außerdem steht eine LC nicht nur zur Realisierung von logischen Funktionen zur Verfügung, sondern kann sowohl als 16-Bit-Register als auch als 16-Bit-Schieberegister genutzt werden (siehe Abb. 4-2a). Bei FPGAs von Xilinx werden 2 LCs zu einem Slice zusammengefasst.

CLB

Ein Logik-Block bei FPGAs, der bei Altera als LAB (*Logic Array Block*) und bei Xilinx als CLB (*Configurable Logic Block*) bezeichnet wird, besteht aus mehreren logischen Zellen (siehe Abb. 4-2b). Diese Logik-Blöcke sind untereinander mit konfigurierbaren Verbindungsmatrizen (*Switch*) verbunden, die ebenfalls auf SRAM Zellen basieren (siehe Abb. 4-2c).

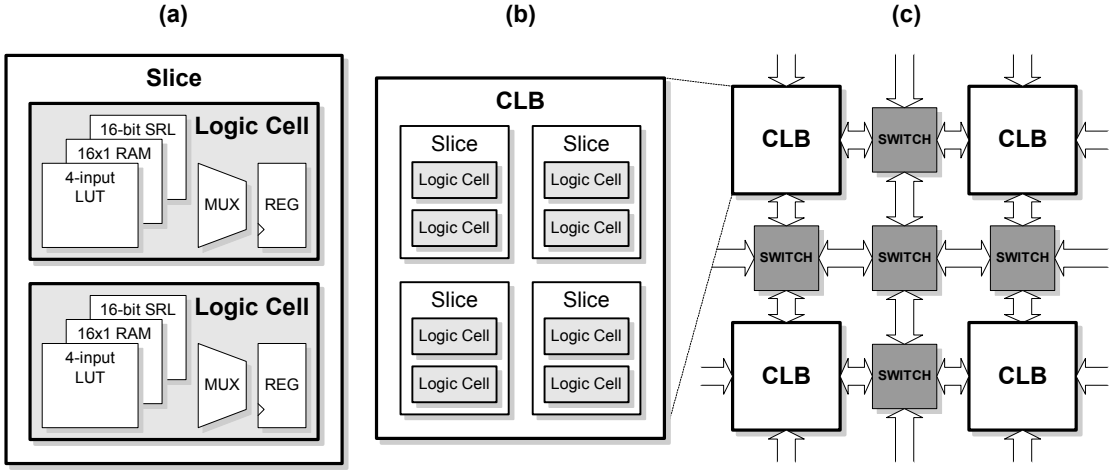


Abb. 4-2: Logik-Hierarchie bei FPGAs: (a) Aufbau einer Slice, (b) Aufbau einer CLB, (c) konfigurierbare Verbindungsmatrizen zwischen CLBs (basierend auf [Maxf04])

Plattform

Die Integrationsdichte ist folglich von der Anzahl der CLBs abhängig, die auf einem FPGA untergebracht werden können. Die heutige FPGA-Technologie stellt neben diesen Logik-Blöcken weitere Makro-Blöcke zur Verfügung. Dies sind zum einen Speicher-Blöcke, die als BRAM-Speicher (BRAM = *Block Random Access Memory*) bezeichnet werden, vorgefertigte Multiplizierer und eingebettete Prozessoren, auf denen Software-Programme ablaufen können. Die FPGAs von Altera (Excalibur) stellen ARM-Prozessoren von ARM (ARM922T) zur Verfügung, die mit maximal 200 MHz betrieben werden können [Alte03], während FPGAs von Xilinx (Virtex-II Pro) über PowerPC-Prozessoren von IBM (PPC405) verfügen, die bei einer maximalen Taktfrequenz von 400 MHz operieren können

[Xili05b]. Aufgrund dieser Integrationsmöglichkeiten hat sich das Einsatzgebiet von FPGAs von der Implementierung von Glue Logic bis zur Realisierung einer Hardware/Software-Plattform erweitert. Die Pins der FPGAs können als Ein- und Ausgänge konfiguriert werden und unterstützen sowohl unterschiedliche Versorgungsspannungen (1,5 V bis 3,3 V) als auch unterschiedliche Signalstandards (Single-Ended/Differentiell), wie LVCMOS (*Low Voltage Complementary Metal Oxide Semiconductor*), LVTTL (TTL = *Transistor-Transistor Logic*) oder LVDS (DS = *Differential Signaling*). Für FPGAs mit hoher Integrationsdichte werden nicht mehr QFP-Gehäuse (*Quad Flat Pack*) verwendet, bei denen die Pins an den 4 Gehäusekanten sitzen, sondern es kommen ausschließlich BGA-Gehäuse (*Ball- Grid-Array*) zum Einsatz, bei denen sich die Pins unterhalb des Gehäuses befinden. Die maximal nutzbare Anzahl an Pins liegt bei Virtex-II FPGAs von Xilinx bei 1108, wenn das BGA-Gehäuse FF1517 (*Flip-Chip Fine-Pitch*) verwendet wird [Xili05c]. Für Takteingänge (*Clock*) sind bei den FPGAs spezielle Pins zu verwenden, die direkt in die Mitte des FPGAs geführt werden, um von dort aus mit entsprechenden Treiberschaltungen (*Buffer*) alle Logik- und Makro-Blöcke zu versorgen (siehe Abb. 4-3).

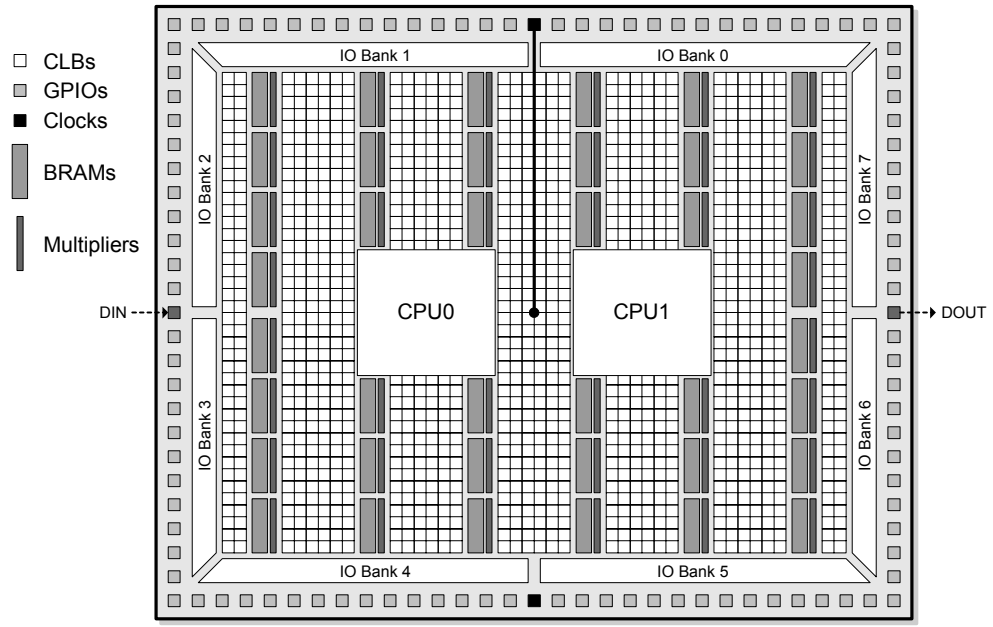


Abb. 4-3: Integrationsdichte heutiger FPGAs anhand von internen Speichern, Multiplizierern und CPUs

5 FPGA-Vorteile

Die Vorteile heutiger FPGA-Technologien liegen neben der Flexibilität (durch die Rekonfigurierbarkeit der Hardware) folglich in der hohen Integrationsdichte, der ausreichend hohen Performanz bezüglich der Operationsfrequenzen, der Möglichkeit aufgrund der hohen Anzahl an verfügbaren Pins eine hohe Speicherbandbreite zu erzielen und vor allem in der Kostenwirksamkeit. Die neue Emulationsplattform dient als Prototyp (sehr geringe Stückzahl) und benötigt zur Effizienzsteigerung BGA-Gehäuseformen, die eine hohe Speicherbandbreite erlauben. Die Verwendung solcher kostenaufwendiger Gehäuseformen sind für Voll- oder Halb-Kunden-Entwürfe nur bei sehr hohen Stückzahlen wirtschaftlich. Aus diesem Grund erfordert eine effiziente Emulationsplattform ein FPGA basiertes Architekturkonzept.

4.1.2 Genauigkeitsanforderung an Festkomma-Arithmetik

Bei der Software Simulation von PCNNs wird für alle Rechenoperationen eine Fließkomma-Arithmetik verwendet. Eine zeiteffiziente PCNN Emulation ist jedoch nur durch die Verwendung von Festkomma-Arithmetik gewährleistet, da die Datenbreite von Festkomma-Zahlen im Vergleich zu Fließkomma-Zahlen wesentlich kleiner ist. Weiterhin ist sowohl der Flächenbedarf als auch der Entwurfs- und Berechnungsaufwand von arithmetischen Hardware-Modulen, wie Addierern und Multiplizierern, für Festkomma-Zahlen geringer als für Fließkomma-Zahlen.

Fließkomma-Arithmetik

Bei der Festkomma-Darstellung unterscheidet man 2 Komponenten: die Anzahl der binären Vorkommastellen *i* (*Integer Part*) und die Anzahl der binären Nachkommastellen *f* (*Fractional Part*). Eine Festkomma-Zahl kann folglich mit *i.f* dargestellt werden. Die größte darstellbare Zahl ist somit 2^{i}-2^{-f}, und die kleinste Zahl beträgt 2^{-f}. Bei der Fließkomma-Darstellung unterscheidet man die 3 Komponenten Vorzeichen *v*, Exponent *e* und Mantisse *m*, und es gibt 2 wesentliche Darstellungsformen: eine 32 Bit-Darstellung (*Single Precision*: *v* | *e* | *m* = 1 | 8 | 23 Bit) und eine 64 Bit-Darstellung (*Double Precision*: *v* | *e* | *m* = 1 | 11 | 52 Bit) [Mano97]. Eine Fließkomma-Zahl wird binär mit 1.*e*.*m* dargestellt und dezimal mit $(-1)^{v}\cdot M\cdot 2^{e}$ berechnet ($1 \le M < 2$, $M = 1+m\cdot 2^{-23|-52}$). Beim Exponenten wird eine Bias-Darstellung verwendet, wobei der Binär-Wert von einem Bias-Wert *b* subtrahiert wird. Bei einer Single bzw. Double Precision-Darstellung beträgt dieser Bias-Wert 127 bzw. 1023. Die numerische Zahlendarstellung Null bzw. Unendlich ist gegeben, wenn der Binär-Wert des Exponenten *e* nur aus Nullen (0-*b* = -*b*) bzw. nur aus Einsen (2^{8} | 2^{11}-1-*b* = *b*+1) besteht. Im Vergleich zu einer Festkomma-Darstellung wird mit der Fließkomma-Darstellung ein viel größerer Wertebereich abgedeckt. Bei einer Single bzw. Double Precision-Darstellung kann ein Zahlenbereich von $\pm 2^{-126}$ bis $\pm(2-2^{-23})\cdot 2^{127}$ bzw. ein Zahlenbereich von $\pm 2^{-1022}$ bis $\pm(2-2^{-52})\cdot 2^{1023}$ dargestellt werden.

2 Qualitätsmaße

Bei der Transformation von Fließkomma-Arithmetik zur Festkomma-Arithmetik ist bei der PCNN Simulation darauf zu achten, dass durch die verminderte Genauigkeit der Festkomma-Arithmetik die Netzwerkdynamik nicht beeinträchtigt wird. Die Genauigkeitsanforderungen an die Festkomma-Arithmetik werden durch minimal notwendige Datenbreiten von Netzwerkparametern definiert und sind vom zu verwendenden Neuronenmodell abhängig. Bei dem Marburger Neuronenmodell (siehe Kapitel 2.2.3 auf Seite 18) beispielsweise wurde in [Roth95] evaluiert, dass Datenbreiten von 5 bis 7 Bit für Synapsengewichte, Membranpotentiale und Schwellwerte ausreichend sind. Für das Neuronenmodell mit adaptiven Synapsengewichten wurden im Rahmen dieser Arbeit erstmals Untersuchungen für eine geeignete Transformation von Fließkomma-Arithmetik zu Festkomma-Arithmetik vorgenommen. Die Anforderungen an die Genauigkeit der Festkomma-Arithmetik wird durch 2 Qualitätsmaße bestimmt, die die Übereinstimmung der Netzwerkdynamik im Vergleich zur Fließkomma-Arithmetik widerspiegeln. Das erste Qualitätsmaß ist die identische Pulsfrequenz ν, und das zweite Qualitätsmaß ist die übereinstimmende Standardabweichung der Pulszeitabstände Δt.

Pulsfrequenz

Bei dem Neuronenmodell mit adaptiven Synapsengewichten, welches im Kapitel 2.2.4 auf Seite 19 vorgestellt wurde, werden 2 Arten von Netzwerkparametern unterschieden: Modellparameter (MP = *Model Parameter*) und Zeitparameter (TP = *Time Parameter*). Modellparameter sind alle Netzwerkparameter, die unmittelbar für die Neuronen- und Synapsenmodellierung verwendet werden, wie a_K, i_K, w_{KL}, μ, γ und θ, und Zeitparameter sind sämtliche Zeitvariablen, die für die Zeitpunktbestimmung von Pulsereignissen notwendig sind.

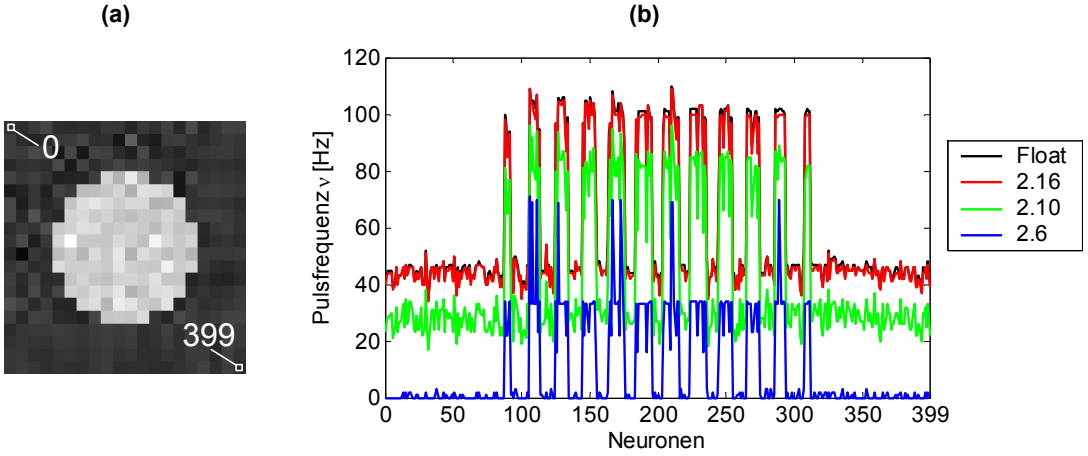


Abb. 4-4: Pulsfrequenzen: (a) Stimulus, (b) Pulsfrequenzen von 3 verschiedenen Festkomma-Repräsentationen der Modellparameter

In Abb. 4-4 sind die Pulsfrequenzen von 400 Neuronen einer 20×20 großen Netzwerkschicht mit einer Vierer-Nachbarschaftsverbindung dargestellt. Bei der Fließkomma-Darstellung (*Float*) feuern die Neuronen, die von den Pixeln des hellen Kreises stimuliert werden (siehe Abb. 4-4a) mit einer Frequenz von etwas über 100 Hz, während die restlichen Neuronen eine Pulsfrequenz von 40 bis 50 Hz aufweisen. Da der Schwellwert θ den Wert 1 hat, sind für die Modellparameter 2 binäre Vorkommastellen ausreichend. Bei 6 binären Nachkommastellen (2.6) zeigt sich, dass die meisten Neuronen aufgrund der unzureichenden Auflösung nie in den Pulszustand gelangen. Bei 10 binären Nachkommastellen (2.10) wird das gleiche Pulsverhalten aller Neuronen erreicht, aber erst bei 16 binären Nachkommastellen (2.16) wird auch die identische Pulsfrequenz wie bei der Fließkomma-Darstellung erzielt.

Pulszeitpunkte

In Abb. 4-5 sind die Standardabweichungen der Pulszeitabstände von 3 verschiedenen Neuronengruppen à 4 Neuronen für eine Netzwerkzeit von 1000 ms dargestellt. Die Standardabweichung der Pulszeitabstände berechnet sich, indem man nach einem Puls eines präsynaptischen Neurons den Zeitabstand ermittelt bis das postsynaptische Neuron einen Puls generiert. Am Ende der Simulation wird die Summe der Zeitabstände durch die Anzahl der ermittelten Zeitabstände gemittelt. Die 3 Neuronengruppen befinden sich innerhalb einer 20×20 großen Netzwerkschicht mit einer Vierer-Nachbarschaftsverbindung. Für jede Neuronengruppe ergeben sich somit 16 Synapsenverbindungen, bei denen die Standardabweichungen der Pulszeitabstände ermittelt worden ist.

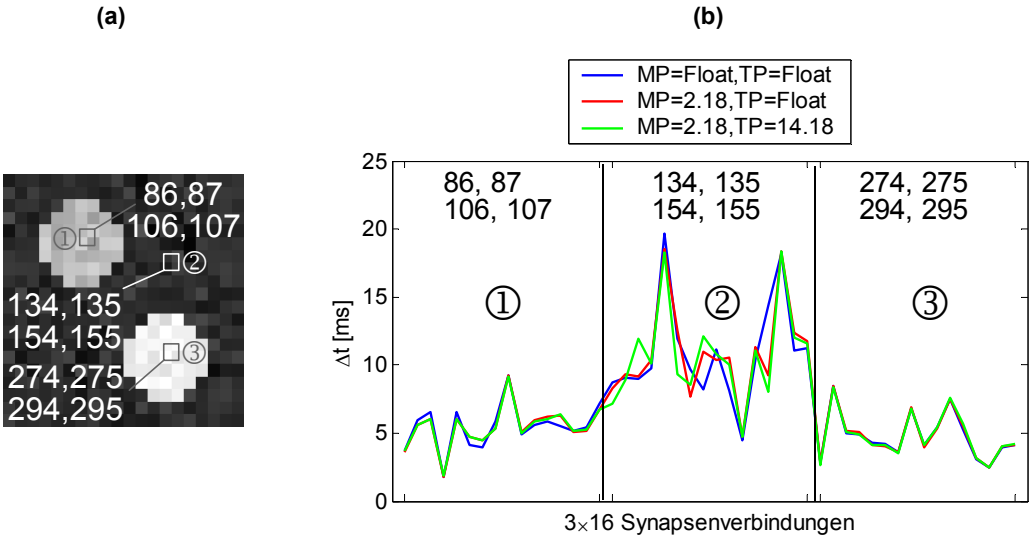


Abb. 4-5: Pulszeitpunkte: (a) Stimulus, (b) Standardabweichung von 3 verschiedenen Neuronengruppen

Die Neuronengruppen ① und ③ befinden sich innerhalb der hellen Kreise des Eingangsbildes (siehe Abb. 4-5a) und werden dadurch mit starken Eingangsreizen stimuliert. Bei diesen beiden Neuronengruppen sind die Pulsabstände zu benachbarten Neuronen gering, da sich im Verlauf der Simulation ein synchronisiertes Pulsverhalten einstellt. Bei der Neuronengruppe ②, die sich im Bildhintergrund befindet, sind die Pulsabstände entsprechend größer, da sich bei diesen Neuronen kein korreliertes Pulsverhalten einstellt. Durchgeführte Untersuchungen haben gezeigt, dass für eine zufriedenstellende Übereinstimmung der Pulszeitabstände 16 binäre Nachkommastellen nicht ausreichend sind, sondern mindestens 18 binäre Nachkommastellen sowohl für die Modell- als auch für die Zeitparameter notwendig sind. In Abb. 4-5b sieht man für die Neuronengruppen ① und ③ die sehr gute Übereinstimmung der Pulszeitabstände, die für die Erhaltung der Netzwerkdynamik essentiell ist, da das synchronisierte Pulsverhalten von benachbarten Neuronen erhalten bleibt.

18 binäre Nachkommastellen

Für eine ausreichende Genauigkeit der Festkomma-Arithmetik sind somit neben den 2 binären Vorkommastellen 18 binäre Nachkommastellen notwendig. Auf der Emulationsplattform werden Modellparameter mit 20 Bit (2.18) und Zeitparameter mit 32 Bit (14.18)

dargestellt. Das Neuronenmodell mit adaptiven Synapsengewichten erfordert eine höhere Genauigkeit als andere pulscodierte Neuronenmodelle, und somit ist es aufgrund der notwendigen Bitbreite von erhöhter Bedeutung, dass eine ausreichend hohe Speicherbandbreite für die PCNN Emulation bereitgestellt wird.

4.1.3 Eigenschaften eines effizienten Emulationsverfahrens

Für ein effizientes Emulationsverfahren auf der Emulationsplattform sind diejenigen Eigenschaften (siehe auch Kapitel 3.1.1 auf Seite 34) auszuwählen, die eine höhere Performanz, eine größere Flexibilität und vor allem eine effizientere Bearbeitung von adaptiven Synapsengewichten versprechen.

5 Eigenschaften

Ein paralleles Emulationsverfahren wird durch die Bereitstellung von mehreren PEs realisiert und führt zu einer höheren Performanz als ein sequentielles Emulationsverfahren. Die Rechenkomplexität bei einem Pulsereignis-basierten Emulationsverfahren steigt lediglich linear mit der Anzahl der zu adaptierenden Synapsengewichte an und führt bei Pulsfrequenzen von bis zu 120 Hz ebenfalls zu einer höheren Performanz als ein Zeitscheiben-basiertes Emulationsverfahren. Für eine ausreichende Flexibilität ist ein numerisches Emulationsverfahren gegenüber einem analytischen Emulationsverfahren vorzuziehen, da Modelländerungen schnell angepasst werden können, und somit die Emulationsplattform eher den Charakter einer Experimentierplattform hat. Ein empfängerorientiertes Emulationsverfahren, bei dem die Neuronen als postsynaptisch betrachtet werden, führt im Vergleich zu einem senderorientierten Emulationsverfahren, bei dem die Neuronen als präsynaptisch angesehen werden, zu einer effizienteren Bearbeitung von adaptiven Synapsengewichten. Für die Adaption eines Synapsengewichts ist der Pulszustand des präsynaptischen Neurons und der Membranpotentialwert des postsynaptischen Neurons erforderlich. Wird ein Neuron als postsynaptisch betrachtet, müssen lediglich die Pulszustände der präsynaptischen Neuronen übertragen werden (Pulszustand eines Neurons kann mit 1 Bit codiert werden), um die Synapsengewichte zu adaptieren. Wird ein Neuron allerdings als präsynaptisch angesehen, müssen alle Membranpotentiale der postsynaptischen Neuronen transferiert werden (Membranpotential eines Neurons wird mit mehreren Bits codiert). Der Kommunikationsaufwand bei einem neuronenparallelen Emulationsverfahren ist im Vergleich zu einem synapsenparallelen Emulationsverfahren erheblich reduziert und führt ebenfalls zu einer effizienteren Bearbeitung von adaptiven Synapsengewichten. Zusammenfassend wird ein paralleles, Pulsereignis-basiertes, numerisches, senderorientiertes und neuronenparalleles Emulationsverfahren für SEE favorisiert.

Nachdem die notwendigen Eigenschaften und die Notwendigkeit einer hohen Speicherbandbreite für die Emulation diskutiert wurden, wird im folgenden das Architekturkonzept der Emulationsplattform vorgestellt.

4.1.4 Architekturkonzept der Emulationsplattform

Das Architekturkonzept von SEE basiert auf 3 FPGAs, die während der Emulation jeweils eigenständige Aufgaben übernehmen, und ist in Abb. 4-6 dargestellt.

PPC2-FPGA

Das obere FPGA ist für die Emulationsablaufsteuerung zuständig und wird mit PPC2-FPGA abgekürzt, da es 2 eingebettete PPCs (PowerPC-Prozessoren) beinhaltet. Die Emulationsablaufsteuerung beinhaltet 3 Aufgaben: 1.) Konfiguration des Netzwerks vor dem Emulationsstart über externe serielle Schnittstellen, 2.) Protokollierung von Netzwerkparametern und Pulsereignissen, und 3.) Verwaltung von Ereignislisten. Während der Emulation müssen 2 Ereignislisten verwaltet werden: die dynamische Ereignisliste (DEL = *Dynamic Event List*) und die Feuer-Ereignisliste (FEL = *Fire Event List*). Die DEL beinhaltet alle erregten Neuronen, die zum betrachteten Emulationszeitpunkt einen Puls oder einen externen Reiz empfangen. Die FEL speichert alle Neuronen, die einen Puls senden und sich somit im Pulszustand befinden, und den dazugehörigen Zeitpunkt, an

dem das Neuron wieder den Pulszustand verlässt. Die maximale Anzahl an Neuronen, die während einer Simulation berücksichtigt werden können, wird von der Speicherkapazität der DEL (2 MB) bestimmt, weil im Extremfall alle Neuronen im Netzwerk erregt sein können. Da ein Neuron mit 4 Bytes codiert wird, beträgt die maximale Anzahl an emulierbaren Neuronen $2^{21}/4 = 2^{19} \approx 0{,}5 \cdot 10^6$. Aufgrund des zusätzlich abzuspeichernden Zeitwertes muss die FEL im Vergleich zur DEL eine zweifache Speicherkapazität bereitstellen.

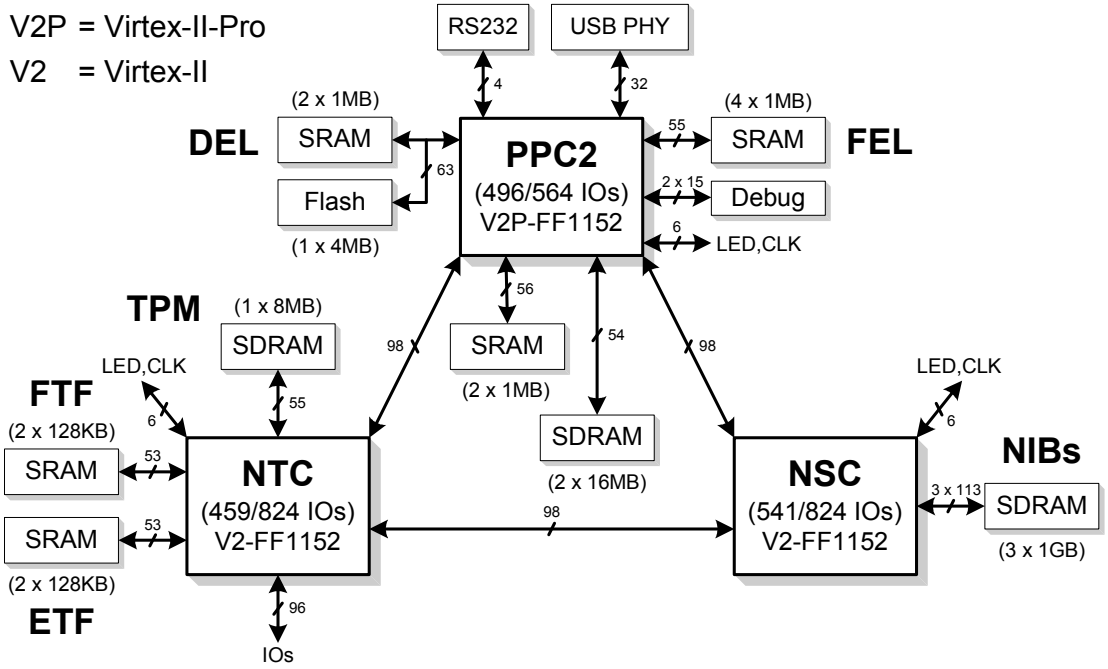


Abb. 4-6: Architekturkonzept der digitalen Emulationsplattform

NTC-FPGA

Das linke FPGA ist für die Netwerktopologieberechnung verantwortlich und wird als NTC-FPGA bezeichnet (NTC = *Network Topology Computation*). Die Netzwerktopologieberechnung besteht aus 2 Aufgaben: 1.) Ermittlung der präsynaptischen Pulsaktivität eines erregten postsynaptischen Neurons, und 2.) Berechnung der postsynaptischen Neuronen, die durch einen neuen Puls eines präsynaptischen Neurons in den Erregungszustand gelangen. Für diese Berechnungsaufgaben sind 2 Markierungsspeicher notwendig: der Feuer-Tag-Speicher (FTF = *Fire Tag-Field*) und der Erregungs-Tag-Speicher (ETF = *Excitation Tag-Field*). Das FTF markiert alle Neuronen, die sich im Pulszustand befinden, und wird für die Berechnung der präsynaptischen Pulsaktivität benötigt. Das ETF kennzeichnet alle Neuronen, die sich sowohl im Pulszustand als auch im Erregungszustand befinden, und wird bei der Berechnung der postsynaptischen Neuronen gebraucht, die durch einen Puls erregt werden. Anhand der Information im ETF wird verhindert, dass Neuronen, die sich bereits im Puls- bzw. Erregungszustand befinden, als neu erregte postsynaptische Neuronen betrachtet werden. Ein zusätzlicher SDRAM-Speicher (8 MB) dient als Topologiespeicher (TPM = *Topology Memory*), um irreguläre Netzwerkverbindungen abzulegen, die keinem regulären Verbindungsschema entsprechen und somit nicht während der Verarbeitung berechnet werden können.

NSC-FPGA

Das rechte FPGA übernimmt die Neuronenzustandsberechnung und wird mit NSC-FPGA abgekürzt (NSC = *Neuron State Computation*). Das NSC-FPGA kann für das numerische Berechnungsverfahren, welches für die Neuronenmodellierung notwendig ist, auf 3 notwendige Informationsquellen parallel zugreifen: 1.) Erregte Neuronen in der DEL, bei denen der Neuronenzustand über das PPC2-FPGA aktualisiert werden muss, 2.) Präsynaptische Pulsaktivität über das NTC-FPGA, und 3.) Neuronen-spezifischer Parameterspeicher (NIB = *Neuron Information Block*), der Informationen wie Membranpotential und Synapsengewichte von verbundenen präsynaptischen Neuronen enthält. Der Parameterspeicher, der die NIBs von jedem Neuron im Netzwerk enthält, verfügt über 3 parallel nutzbare Datenkanäle. Diese Anzahl an parallel nutzbaren Datenkanälen ist mindestens

notwendig, um auch dichtere Vernetzungsstrukturen bei dem verwendeten Integrationsverfahren berücksichtigen zu können, wie im weiteren Verlauf dieses Kapitels noch gezeigt wird. Die maximale Anzahl an adaptiven Synapsen, die während einer Simulation berücksichtigt werden können, wird von den 3 SDRAM-DIMM-Speichern bestimmt, die für die Speicherung der NIBs zur Verfügung stehen. Bei einer Größe von 1 GB ergibt sich für die Synapsen eine maximale Anzahl von $(3 \cdot 2^{30}\text{-}2^{19})/4 \approx 800 \cdot 10^6$.

NTC/NSC-Kommunikation

Im Vergleich zu existierenden Architekturkonzepten findet bei SEE eine echte Trennung zwischen Netzwerktopologieberechnung und Neuronenzustandsberechnung statt. Dies wird durch die Kommunikation von Topologie-Vektoren erreicht, bei der die Netzwerktopologieberechnung lediglich die Pulszustände der präsynaptischen Neuronen an die Neuronenzustandsberechnung übermittelt. Die Netzwerktopologieberechnung ist somit nicht in die Neuronenmodellierung durch die Modifikation von Synapsengewichten involviert. Dies ist aber bei allen existierenden Architekturkonzepten der Fall. Insbesondere für die Adaption von Synapsengewichten ist solch eine Kommunikation vorteilhaft, da für die Aktualisierung der Synapsengewichte die Membranpotentiale der postsynaptischen Neuronen notwendig ist. Durch diese Kommunikation wird zum einen die notwendige Datentransferrate reduziert und zum anderen der Ablauf der Neuronenzustandsberechnung vereinfacht. Würde die Aktualisierung der Synapsengewichte innerhalb der Netzwerktopologieberechnung stattfinden, müssten aktuelle Membranpotentialwerte von der Neuronenzustandsberechnung abgeholt werden, um danach die aktualisierten Synapsengewichte wieder an die Neuronenzustandsberechnung zu transferieren.

Das Architekturkonzept von SEE stellt eine verteilte Speicherarchitektur in den Vordergrund, da nur durch den parallelen Zugriff von Ereignislisten, Markierungsspeicher und Parameterspeicher eine effizientere Parallelverarbeitung realisiert werden kann. Die notwendige hohe Speicherbandbreite wird insbesondere für den Zugriff auf die NIBs bereitgestellt, damit die Neuronenzustandsberechnung entsprechend beschleunigt werden kann.

4.2 Maßnahmen zur Beschleunigung der Emulation

Puls-Start/Stop-Ereignis

Die angewandten Maßnahmen zur Beschleunigung der Emulation werden nun im folgenden zusammengefasst. Bei den Emulationsereignissen wird zwischen Puls-Start-Ereignis (*Fire-Start-Event*) und Puls-Stop-Ereignis (*Fire-Stop-Event*) unterschieden. Ein Puls-Start-Ereignis ist ein Emulationsereignis, an dem ein Neuron in den Pulszustand bzw. Feuerzustand gelangt, und ein Puls-Stop-Ereignis ist ein Emulationsereignis, an dem ein Neuron den Pulszustand verlässt und sich wieder im Ruhe- oder Erregungszustand befindet.

4.2.1 Klassifizierung von pulscodierten Neuronen

Für die Bestimmung des nächsten Puls-Start-Ereignisses ist es während der PCNN Emulation nicht notwendig, alle Neuronen im Netzwerk heranzuziehen. Das Membranpotential a_K nach Gl. 2-13 auf Seite 19 bleibt konstant solange das Neuron N_K keinen Puls eines präsynaptischen Neurons N_L ($X_L = 0$) und keinen Eingangsreiz i_K erhält. Aus diesem Grund können weder Neuronen, die sich im Pulszustand befinden, noch Neuronen, die nicht durch einen Eingangsreiz oder durch einen Puls erregt werden, ein Puls-Start-Ereignis verursachen. Durch eine Klassifizierung der Neuronen in erregt und nicht-erregt nach

Tab. 4-1 wird eine beschleunigte Ermittlung des nächsten Puls-Start-Ereignisses während der Emulation erreicht.

Erregte Neuronen			**Nicht-Erregte Neuronen**	
feuernd	dynamisch	statisch	abklingend	ruhend
$X_K = 1$	$\Sigma X_L > 0$	$\Sigma X_L = 0 \wedge i_K \neq 0$	$\Sigma X_L = 0 \wedge i_K = 0$	$\forall W_{KL} = W_{KLmin}$

Tab. 4-1: Klassifizierung von erregten und nicht-erregten Neuronen

Erregte Neuronen

Erregte Neuronen lassen sich in feuernd, dynamisch erregt und statisch erregt unterteilen. Dynamisch erregte Neuronen empfangen Pulse von präsynaptischen Neuronen. Die Zeitpunkte und Aufeinanderfolge dieser Pulse sind nicht vorhersagbar, und somit lässt sich auch der Feuerzeitpunkt des Neurons nicht genau bestimmen. Das Pulsverhalten des Neurons ist dynamisch. Statisch erregte Neuronen werden nur durch einen externen Eingangsreiz stimuliert. Der Feuerzeitpunkt t_f eines statisch erregten Neurons kann zum Zeitpunkt t_0 exakt berechnet werden (siehe Gl. 4-1), und somit ist das Pulsverhalten des Neurons statisch.

$$t_f = \frac{\theta - a_K(t_0)}{i_K} = \frac{\theta - a_K(t_0)}{I_K \cdot W_0} \qquad \textbf{(4-1)}$$

Nicht-erregte Neuronen

Nicht-erregte Neuronen können keine Pulse generieren, da das Membranpotential sich nicht verändert. Solange die Synapsengewichte noch nicht ihren Minimalwert W_{KLmin} erreicht haben, muss das Abklingverhalten dieser Synapsen bei jeder Aktualisierung des Netzwerks berechnet werden. Erst wenn alle Synapsengewichte ihren Minimalwert erreicht haben, befindet sich das Neuron im Ruhezustand und muss für den weiteren Emulationsverlauf nicht mehr berücksichtigt werden. Diese Klassifizierung der Neuronen führt zu der Einführung der beiden Ereignislisten DEL und FEL, die bereits bei der Vorstellung des Architekturkonzeptes erwähnt wurden. Für die Bestimmung des nächsten Puls-Start-Ereignisses müssen somit nur dynamisch und statisch erregte Neuronen betrachtet werden, die sich alle in der DEL befinden.

4.2.2 Parallelverarbeitung des Pulsereignis-basierten Emulationsablaufs

Anstatt eines sequentiellen Ablaufs von Neuronenzustands- und Netzwerktopologiephase (siehe auch Kapitel 3.1 auf Seite 33) wird der Ablauf der PCNN Emulation durch eine parallele Ausführung dieser beiden Phasen beschleunigt. Der Emulationsablauf von SEE ist in Abb. 4-7 dargestellt.

Topologie-Vektor

Der Ablauf der Netzwerktopologiephase (*NTC-Phase*) ist auf der linken Seite dargestellt und umfasst 3 Zustände: *Topvec-Spike*, *Topvec-Update* und *Update*. In den Zuständen *Topvec-Spike* und *Topvec-Update* wird die präsynaptische Pulsaktivität ermittelt, indem die Pulszustände vom FTF gelesen werden. Die Pulszustände werden dann in Form von Topologie-Vektoren (TV = *Topology Vector*) zusammengefügt und der Neuronenzustandsberechnung zur Verfügung gestellt. Im Zustand *Update* werden die Markierungsspeicher entsprechend dem aufgetretenen Emulationsereignis aktualisiert. Bei einem Puls-Stop-Ereignis wird die entsprechende Markierung des Neurons im FTF gelöscht. Bei einem Puls-Start-Ereignis wird das Neuron im FTF markiert. Im ETF werden die postsynaptischen Neuronen markiert, die durch den Puls des Neurons erregt werden. Die neu erregten postsynaptischen Neuronen werden anschließend der DEL hinzugefügt.

Puls-Stop/Start

Der Ablauf der Neuronenzustandsphase (*NSC-Phase*) ist auf der rechten Seite dargestellt und umfasst 4 Zustände: *Read*, *Next-Spike*, *Next-Event* und *Update*. Im Zustand *Read* wird aus der FEL der nächste Puls-Stop-Zeitpunkt eines Neurons gelesen, welches sich im Pulszustand befindet. Im Zustand *Next-Spike* erfolgt dann die Ermittlung des nächsten Puls-Start-Zeitpunktes bei allen erregten Neuronen, die sich in der DEL befinden. Im darauffolgenden Zustand *Next-Event* wird entschieden, ob zuerst das Puls-Stop- oder das Puls-Start-Ereignis eintritt. Das früher eintretende Ereignis wird als nächstes Emulationsereignis definiert und der Netzwerktopologieberechnung für die Aktualisierung der Mar-

kierungsspeicher mitgeteilt. Im letzten Zustand *Update* werden dann alle Neuronen im Netzwerk, die sich nicht im Ruhezustand befinden, aktualisiert.

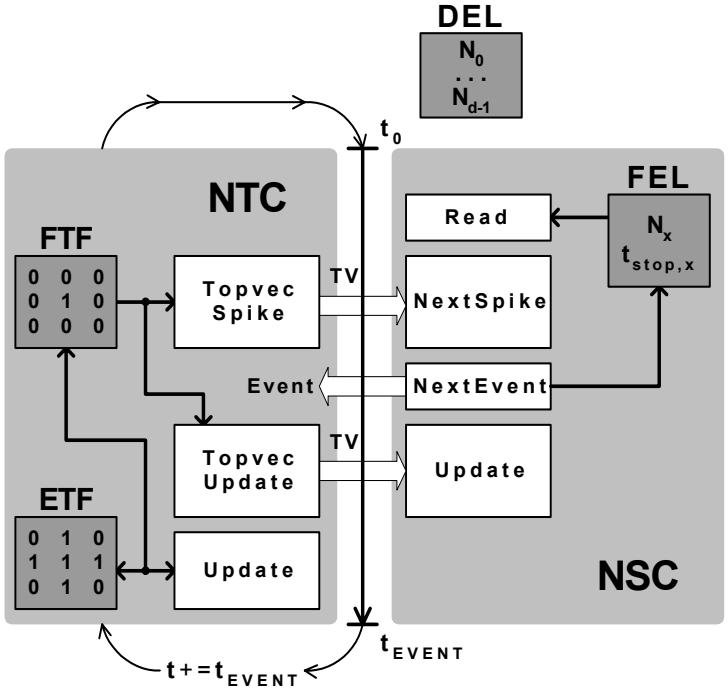


Abb. 4-7: Pulsereignis-basierter Emulationsablauf

Der gesamte Emulationsablauf von SEE ist Pulsereignis-basiert, da die Netzwerkaktualisierung nicht in festen Zeitscheiben erfolgt, sondern nur bei Pulsereignissen wie dem Puls-Stop- oder dem Puls-Start-Ereignis. Durch den beschriebenden Ablauf der PCNN Emulation wird eine Parallelverarbeitung der beiden High-Level Operationen Netzwerktopologie- und Neuronenzustandsberechnung erreicht (siehe auch Kapitel 3.3.2 auf Seite 41).

4.2.3 Optimale Datenkanalauslastung einer verteilten Speicherarchitektur

Aufgrund der verteilten Speicherarchitektur ist SEE in der Lage, insgesamt 10 eigenständige Datenkanäle mit einer Speicherbandbreite von insgesamt 2,6 GB/s zur Verfügung zu stellen.

10 Datenkanäle mit 416 Bit

Das PPC2-FPGA kann 4 Datenkanäle (3×SRAM à 32 Bit und 1×SDRAM à 32 Bit) nutzen, die eine Speicherbusbreite von insgesamt 4·32 Bit = 16 Bytes haben. Das NTC-FPGA verfügt über 3 parallel nutzbare Datenkanäle (2×SRAM à 32 Bit und 1×SDRAM à 32 Bit), die eine Speicherbusbreite von 3·32 Bit = 12 Bytes zur Verfügung stellen. Schließlich verfügt das NSC-FPGA ebenfalls über 3 parallel nutzbare Datenkanäle (3×SDRAM à 64 Bit), die allerdings eine Speicherbusbreite von 3·64 Bit = 24 Bytes zur Verfügung stellen. Bei einer Operationsfrequenz von 50 MHz bietet die Emulationsplattform einen theoretisch maximal möglichen Datendurchsatz von 52 Bytes·50 MHz = 2,6 GB/s.

Eine optimale Datenkanalauslastung versucht Leerlaufzeiten auf dem Datenkanal zu verhindern und möglichst mit jedem Takt verwendbare Daten zu transferieren. Solch eine optimale Datenkanalauslastung muss für den Datentransfer zwischen dem NSC-FPGA und dem Parameterspeicher in Form der 3 SDRAM-DIMM-Speicher sichergestellt werden. Dies ist für eine effiziente Berechnung der Neuronenzustände erforderlich und setzt voraus, dass die Neuronenzustandsberechnung innerhalb des NSC-FPGAs durch eine Parallelverarbeitung mit Pipelining realisiert ist, was Leerlaufzeiten auf dem entsprechenden Datenkanal zum Parameterspeicher vermeidet (siehe auch Kapitel 3.3.2 auf Seite 41). Der theoretisch maximal mögliche Datendurchsatz vom NSC-FPGA zum Parameterspeicher beträgt 24 Bytes·50 MHz = 1,2 GB/s.

Datenkanäle für HW und SW

Vom PPC2-FPGA zu den anderen beiden FPGAs gibt es jeweils einen Hardware- und einen Software-Datenkanal mit einer jeweiligen Datenbusbreite von 32 Bit. Über den SW-Datenkanal kann die jeweilige CPU im PPC2-FPGA auf interne Register oder auf Speicher des NTC- bzw. des NSC-FPGAs zugreifen. Zeitgleich können die Hardware-Module im NTC- bzw. im NSC-FPGA über den HW-Datenkanal erregte Neuronen aus der DEL lesen oder in die DEL schreiben und zusätzlich Netzwerkinformationen oder Emulationsereignisse in den Speichern des PPC2-FPGAs ablegen, die dann von der Software zugänglich sind.

4.2.4 Autarkie der Emulationsplattform

Die Autarkie der Emulationsplattform ist ein wichtiges Kriterium, um die Emulation effizient beschleunigen zu können, und hat zur Folge, dass ein eigenständiger Betrieb (*Stand-Alone*) der Emulationsplattform möglich ist. Bei einem nicht-eigenständigen Betrieb ist während der Emulation die Kommunikation zu einem Host-Rechner notwendig, der die Emulationsablaufsteuerung übernimmt, Pulslisten leert und auffüllt oder sonstigen Datentransfer initiiert, und dadurch unter Umständen die Emulation kurzzeitig anhalten muss. Voraussetzung für die Autarkie der Emulationsplattform ist die Bereitstellung einer ausreichenden Speicherkapazität des Parameter- bzw. Gewichtsspeichers, der alle Membranpotentiale und adaptiven Synapsen enthält. Dadurch ist während der Emulation großer PCNNs keine Intervention des Host-Rechners notwendig. Dieser müsste bei nicht ausreichend großen Parameterspeichern permanent aktualisierte Netzwerkparameter von der Emulationsplattform abholen und zu aktualisierende Netzwerkparameter vom Massenspeicher an die Emulationsplattform übertragen.

4.2.5 Parallelisierung des notwendigen Integrationsverfahrens

Der Hardware-Entwurf einer Emulationsplattform muss sich im wesentlichen auf die Parallelisierung der Rechenoperationen fokussieren, die für die Berechnung der Neuronenzustände notwendig sind. Diese Parallelisierung kann durch Bereitstellung von mehreren PEs realisiert werden, die die Datenparallelität bei PCNNs (Lokalität von notwendigen Netzwerkparametern) ausnutzen. Innerhalb der PEs wird das numerische Integrationsverfahren von Bulirsch-Stoer für die Neuronenzustandsberechnung durch numerische HW-Integratoren ausgeführt. Diese HW-Integratoren realisieren anhand einer Pipeline-Struktur die arithmetischen Operationen, die bei dem numerischen Integrationsverfahren notwendig sind. Jede PE muss insbesondere bei der Bearbeitung von adaptiven Synapsen über einen eigenen Datenkanal verfügen, den es nicht mit einer anderen PE teilen muss. Die Anzahl der HW-Integratoren innerhalb einer PE muss ausreichend hoch gewählt werden, damit, nachdem sich ein HW-Integrator seine Netzwerkparameter aus dem Parameterspeicher geholt hat, sofort der nächste HW-Integrator seine Netzwerkparameter aus dem Parameterspeicher holen bzw. aktualisierte Netzwerkparameter in den Parameterspeicher zurückschreiben kann. Durch die Bereitstellung von PEs und HW-Modulen für die numerische Integration wird eine Parallelisierung der Low-Level Operationen erzielt, welche bei der PCNN Emulation, insbesondere bei der Neuronenzustandsberechnung, auftreten (siehe auch Kapitel 3.3.2).

4.3 Numerisches Integrationsverfahren nach Bulirsch-Stoer

Die Neuronenzustandsberechnung ermittelt die Verläufe der Membranpotential- und Synapsengewichtsfunktionen anhand eines numerischen Integrationsverfahrens. Die Auswahl eines geeigneten Integrationsverfahrens orientiert sich an 3 Eigenschaften: Stabilität, Genauigkeit und notwendiger Rechenaufwand.

Euler-Verfahren

Das Euler-Verfahren ist das bekannteste numerische Integrationsverfahren und erfordert den geringsten Rechenaufwand [Hine03]. Man unterscheidet zwischen einem explizi-

ten Euler-Verfahren (*Forward-Euler*) und einem, im Vergleich dazu stabileren, impliziten Euler-Verfahren (*Backward-Euler*), die in Abb. 4-8 dargestellt sind.

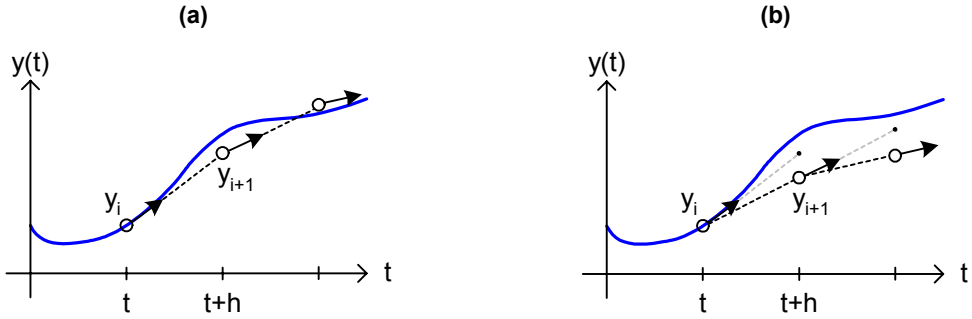


Abb. 4-8: Euler-Verfahren: (a) Forward-Euler, (b) Backward-Euler

Die Ermittlung des nächsten Funktionswertes y_{i+1} beim Forward-Euler-Verfahren erfolgt nach Gl. 4-2 und beim Backward-Euler-Verfahren nach Gl. 4-3,

$$y_{i+1} = y_i + h \cdot f(t, y_i) \qquad \textbf{(4-2)}$$

$$y_{i+1} = y_i + h \cdot f(t + h, y_{i+1}) \qquad \textbf{(4-3)}$$

wobei *h* die Schrittweite darstellt. Da in Gl. 4-2 auf der rechten Seite nur bekannte Größen auftreten, kann der nächste Funktionswert explizit bestimmt werden. In Gl. 4-3 tritt y_{i+1} als unbekannte Größe ebenfalls auf der rechten Seite auf und muss somit implizit durch Lösung der Gleichung ermittelt werden. Dafür wird die Prediktor-Korrektur-Methode (*Predictor-Corrector-Method*) angewendet, bei der man zunächst, wie in Abb. 4-8b dargestellt, nach dem Forward-Euler-Verfahren die Ableitung des nächsten Funktionswertes bestimmt (Prediktion), um mit dieser dann den endgültigen Wert von y_{i+1} festzulegen (Korrektur). Das Euler-Verfahren bietet im Vergleich zu anderen Integrationsverfahren, die mit der gleichen Schrittweite operieren, die geringste Genauigkeit und die geringste Stabilität [Nume05]. Bei PCNN Simulationen ist vor allem die Besonderheit zu beachten, dass jeder Puls eines Neurons die Funktionsverläufe von Membranpotentialen und Synapsengewichten anderer Neuronen beeinflussen kann. Eine zu geringe Genauigkeit des numerischen Integrationsverfahrens führt, auf Kosten der eigentlichen Netzwerkdynamik, zu künstlich erzeugtem synchronen Pulsverhalten von Neuronen [Masc98].

Runge-Kutta-Verfahren

Aufgrund der hohen Genauigkeit und Stabilität wird bei der PCNN Simulation häufig das Runge-Kutta-Verfahren 4. Ordnung als Integrationsverfahren eingesetzt [Bret94]. Bei diesem Verfahren müssen für jede Schrittweite 4 Funktionszwischenwerte berechnet werden, und dieses Verfahren kann zusätzlich zur Effizienzsteigerung mit einer adaptiven Schrittweitensteuerung (*Adaptive Stepsize Control*) versehen werden. Der hohen Robustheit und Genauigkeit des Runge-Kutta-Verfahrens steht ein hoher Rechenaufwand gegenüber, da bei einem Integrationsschritt (Schrittweite) für jeden Funktionszwischenwert eine Ableitungen berechnet werden muss [Nume05].

Bulirsch-Stoer-Verfahren

Einen geeigneten Kompromiss zwischen ausreichender Genauigkeit und Stabilität für das verwendete Neuronenmodell und notwendigem Rechenaufwand bietet das Bulirsch-Stoer-Verfahren [Stoe00] [Oliv71]. Software Simulationen auf Einprozessor-Systemen haben gezeigt, dass bei Verwendung des Bulirsch-Stoer-Verfahrens anstatt des Runge-Kutta-Verfahrens die Simulationszeiten um etwa den Faktor 20 reduziert werden (siehe Simulationszeitverhältnis in Abb. 4-9). Die Simulation bei 30×30 Neuronen beträgt mit dem Bulirsch-Stoer-Verfahren etwa 1,3 Stunden (4659 s), während das Runge-Kutta-Verfahren mehr als 30 Stunden (109142 s) benötigt.

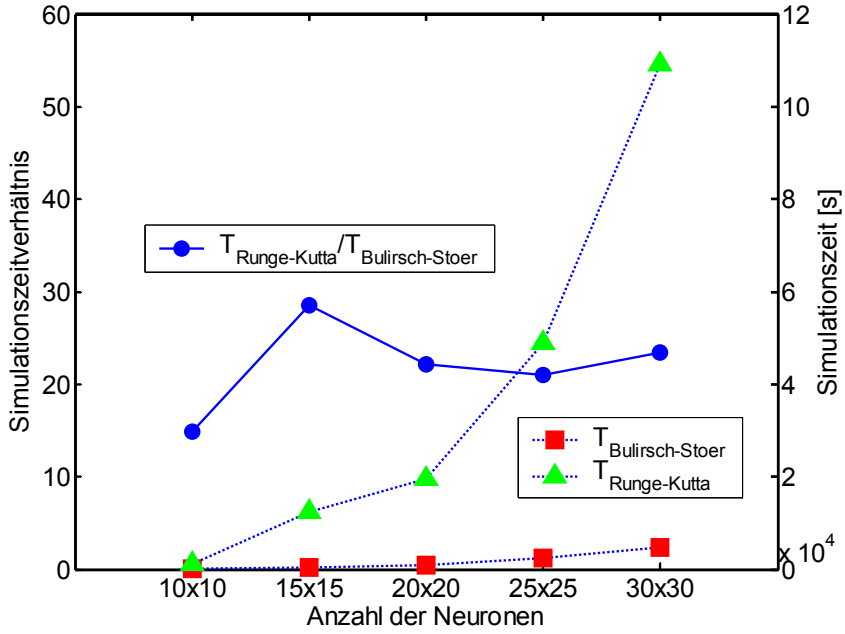


Abb. 4-9: Netzwerksimulationen (t_{END} = 1000 ms) anhand des Runge-Kutta und des Bulirsch-Stoer Integrationsverfahrens auf einem Einprozessor-System (1,2 GHz Prozessor mit 1 GB RAM Arbeitsspeicher), bei denen Neuronen über adaptive Synapsen in einer Vierer-Nachbarschaft untereinander verbunden sind

Aus diesen Voruntersuchungen, die die Überlegenheit des Bulirsch-Stoer-Verfahrens gegenüber dem Runge-Kutta-Verfahren demonstriert haben, entstand der Entschluss, eine geeignete Hardware-Abbildung der notwendigen arithmetischen Operationen beim Bulirsch-Stoer-Verfahren für die Neuronenzustandsberechnung bereitzustellen.

Das numerische Integrationsverfahren nach Bulirsch-Stoer verwendet 2 arithmetische Operationen, die für die Bestimmung des nächsten Funktionswertes iterativ und nacheinander ausgeführt werden müssen [Nume05]: die MMID-Integration (MMID = *Modified Midpoint*) und die Polynomextrapolation, die mit PZEXTR abgekürzt wird. Zusätzlich ist noch ein Nullstellensuchverfahren notwendig, welches durchgeführt werden muss, wenn das Membranpotential den Schwellwert überschritten hat. Diese arithmetischen Operationen werden in den nun folgenden Kapiteln erläutert und die zugehörige Hardware-Abbildung der MMID-Integration und der PZEXTR für das verwendete Neuronenmodell mit adaptiven Synapsengewichten vorgestellt.

4.3.1 Modified Midpoint Integration

Die MMID-Integration ist wie der Name besagt eine modifizierte Midpoint-Integration. Die Midpoint-Integration verwendet die Ableitung des Mittelpunkts eines Intervalls, um den nächsten Funktionswert zu bestimmen. Die Modifikationen im Vergleich zur Midpoint-Integration finden lediglich bei der Bestimmung des ersten Funktionszwischenwertes k_1 und des endgültigen Funktionswertes y_{t+H} statt [Nume05].

Mathematische Formeln

Das Integrationsintervall H, welches den Zeitabschnitt vom aktuellen Funktionswert y_t zum nächsten Funktionswert y_{t+H} definiert, wird bei der MMID-Integration in Subintervalle $h = H/n$ unterteilt, wobei n die Anzahl der Subintervalle darstellt. Der erste Funktionszwischenwert k_1 wird nach Gl. 4-4 analog zum expliziten Euler-Verfahren bestimmt.

$$k_0 = y_t(t)$$
$$k_1 = k_0 + h \cdot f(t, k_0) \qquad \textbf{(4-4)}$$

Die nachfolgenden Funktionszwischenwerte werden nach dem Verfahren der Midpoint-Integration ermittelt. Dabei ist für die Berechnung des nächsten Funktionszwischenwertes k_{i+1} die Ableitung des aktuellen Funktionszwischenwertes k_i und der Wert des vorherigen Funktionszwischenwertes k_{i-1} notwendig (siehe Gl. 4-5).

$$k_{i+1} = k_{i-1} + 2 \cdot h \cdot f(t + i \cdot h, k_i) \qquad i = 1, \ldots, n-1 \tag{4-5}$$

Bei der Bestimmung des endgültigen Funktionswertes y_{t+H} wird eine Mittelung nach Gl. 4-6 durchgeführt, die auf den letzten beiden Funktionszwischenwerten k_n und k_{n-1} basiert.

$$y_{t+H} = \frac{1}{2} \cdot [k_n + k_{n-1} + h \cdot f(t + H, k_n)] \tag{4-6}$$

Die Ermittlung eines Funktionswertes nach der MMID-Integration ist zusammenfassend in Abb. 4-10 illustriert, in der die vorgestellten mathematischen Formeln (Gl. 4-4 bis Gl. 4-6) zum Einsatz kommen.

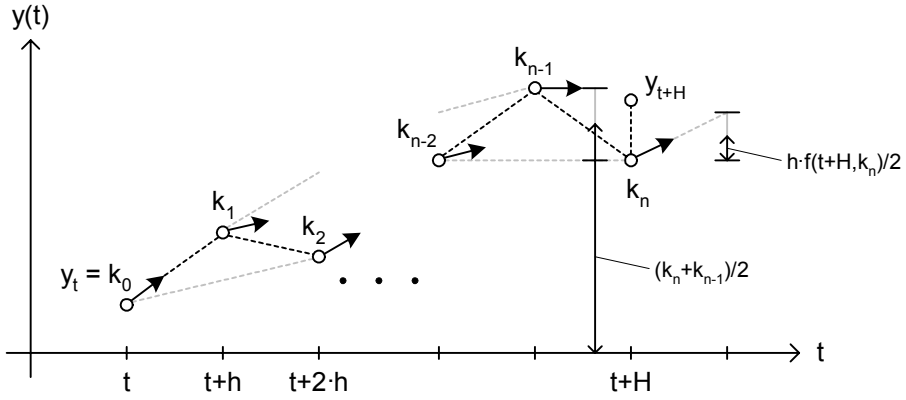


Abb. 4-10: Modified Midpoint Integration

Hardware Mapping

In den Gleichungen Gl. 4-5 und Gl. 4-6 erkennt man, dass jeweils für die Bestimmung eines neuen Funktionswertes stets 2 Funktionswerte gelesen werden müssen. Für die Bestimmung von k_{i+1} in Gl. 4-5 müssen k_{i-1} und k_i und für die Bestimmung von y_{t+H} in Gl. 4-6 müssen k_{n-1} und k_n gelesen werden. Aus diesem Grund sind für eine effiziente Fließbandverarbeitung 3 Datenkanäle notwendig, auf die parallel zugegriffen werden kann.

Ableitung von a_K

Die für die MMID-Integration notwendige Ableitung für das Membranpotential a_K ergibt sich aus der Gl. 2-13 auf Seite 19 und ist in Gl. 4-7 angegeben.

$$a_K(t)' = i_K + \sum_{i \in N_S} W_{KLi}(t) \tag{4-7}$$

Mit N_S in Gl. 4-7 wird die Menge aller präsynaptischen Neuronen bezeichnet, die sich zum Zeitpunkt t im Pulszustand befinden. Somit ist für die Ableitung von a_K neben dem Eingangsreiz i_K die Summe der Synapsengewichte notwendig, über die ein Puls verläuft.

Ableitung von W_{KL}

Die Ableitung für die Synapsengewichte W_{KL} ergibt sich nach der Adaptionsregel nach Gl. 2-15 auf Seite 20. Die Hardware-Abbildung für die Ableitung von W_{KL} nach dieser Adaptionsregel ist in Abb. 4-11 dargestellt.

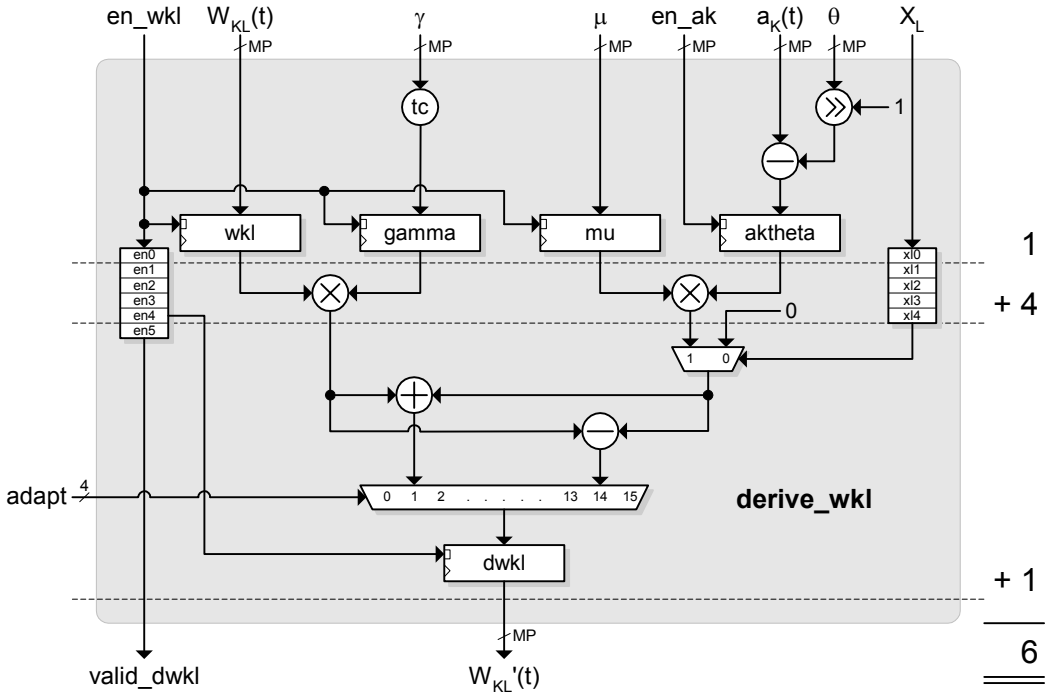


Abb. 4-11: HW-Modul (`derive_wkl`) für die Bestimmung der zeitlichen Ableitung der Synapsengewichte W_{KL}

Im ersten Takt werden das Synapsengewicht (*wkl* in Abb. 4-11), die negative Abklingkonstante γ (*gamma*), der Verstärkungsfaktor μ (*mu*) und die Differenz von Membranpotential und des halben Schwellwertes θ (*aktheta*) zwischengespeichert. Dann erfolgt eine Multiplikation über 4 Takte für den Abklingterm (*wkl* · *gamma*) und den Verstärkungsterm (*mu* · *aktheta*). Anschließend werden diese je nach vorliegender Adaptionsregel entweder addiert oder voneinander subtrahiert und das Ergebnis in einem Register gespeichert. Die Bestimmung der Ableitung eines Synapsengewichts beträgt für diese Adaptionsregel somit 6 Takte. Die Taktanzahl von 4 Takten für die Multiplikation ergibt sich aus dem verwendeten Pipeline-Multiplizierer, der mit dem Werkzeug CORE Generator von Xilinx erzeugt wurde. Mit diesem Werkzeug ist es möglich, parametrisierbare HW-Module für arithmetische Operationen, wie Multiplikation, Addition oder Division, zu generieren, die für die Implementierung von Xilinx FPGAs optimiert sind. Die 3-stufige Pipeline und das zusätzliche Ausgangsregister der Multiplizierer stellen sicher, dass diese mit einer Operationsfrequenz von 50 MHz betrieben werden können. Addition und Subtraktion, sowie die Zweierkomplement-Bildung (tc = *two-complement*) für die Abklingkonstante γ können innerhalb eines Taktes durchgeführt werden (siehe Abb. 4-11). Aufgrund der Fließbandverarbeitung bei der Berechnung der Ableitung der Synapsengewichte kann nach einer Latenzzeit von 6 Takten pro Takt ein neuer Ableitungswert W_{KL}' für ein Synapsengewicht bestimmt werden.

MMID-Integration von W_{KL}

Das HW-Modul für die Bestimmung von W_{KL}' wird bei der Hardware-Abbildung der MMID-Integration mit integriert, wie in Abb. 4-12 dargestellt ist. Bei dem HW-Modul für die MMID-Integration werden pro Takt 2 Funktionswerte ($W_{KL}(t_0+i\cdot h)$ und $W_{KL}(t_0+(i-1)\cdot h)$) einer Synapsengewichtsfunktion eingespeist.

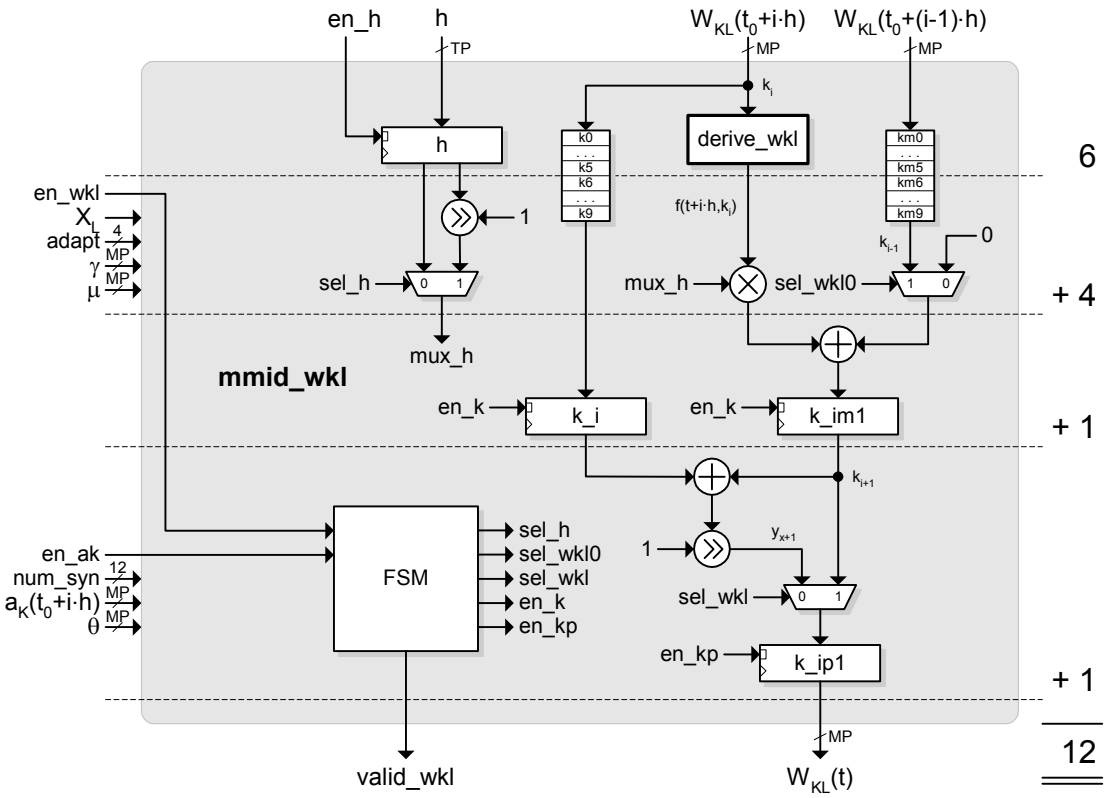


Abb. 4-12: HW-Modul (`mmid_wkl`) für die Modified Midpoint Integration der Synapsengewichte W_{KL}

Nach Gl. 4-5 erfolgt nach den 6 Takten der Ableitungsberechnung des aktuellen Funktionswertes eine Multiplikation (*f(t+i·h,ki)* · *mux_h*) mit dem Integrationsschrittintervall *h* bzw. *2·h*, die weitere 4 Takte benötigt. Beim ersten und letzten Berechnungsschritt erfolgt die Multiplikation mit dem Intervall *h* (siehe Gl. 4-4 und Gl. 4-6), ansonsten mit dem Intervall *2·h*. Im nächsten Takt folgt eine Addition mit dem vorherigen Funktionswert. Die Summe wird im Register *k_im1* zwischengespeichert. Bei der Bestimmung des letzten Funktionswertes (siehe Gl. 4-6) ist dann noch eine weitere Addition mit dem aktuellen Funktionswert notwendig, der im Register *k_i* abgelegt ist. Das Endergebnis eines Berechnungsschrittes der MMID wird schließlich im Ausgangsregister *k_ip1* hinterlegt. Es ergeben sich somit insgesamt 12 Takte, bevor der erste neue Funktionswert eines Synapsengewichts berechnet ist. Dann können mit jedem weiteren Takt die neuen Funktionswerte der anderen Synapsengewichte präsynaptischer Neuronen berechnet werden. Die Latenzzeit t_{MMID} für die MMID-Integration von Synapsengewichten beträgt, wie in Gl. 4-8 dargestellt 12 Takte.

$$t_{MMID} = 12 \qquad \textbf{(4-8)}$$

Das HW-Modul für die MMID-Integration zeigt, dass bei einer geringen Anzahl an Synapsengewichten die neu berechneten Funktionswerte auf dem FPGA in BRAM-Speichern oder FIFOs zwischengespeichert werden können, um direkt wieder die Pipeline des HW-Moduls für die Berechnung des nächsten Funktionswertes zu füllen. Bei einer dichten Vernetzungsstruktur ist diese Zwischenspeicherung nicht möglich. In diesem Fall sind 3 parallel nutzbare Datenkanäle (2 Lese- und ein Schreibkanal) notwendig, um die neu berechneten Funktionswerte direkt im externen SDRAM-Speicher zwischenspeichern zu können.

4.3.2 Polynomextrapolation

Die Anzahl der Subintervalle, die bei der MMID-Integration notwendig sind, ergibt sich nach der Bulirsch-Folge in Gl. 4-9, die den Wert *n* festlegt [Nume05].

$$n = 2, 4, 6, 8, 12, 16, 24, 32, \ldots \quad | \; n_i = (2 \cdot n_{i-2}) \wedge i > 2 \tag{4-9}$$

Dies bedeutet, dass man das Intervall *H* zuerst in 2 Subintervalle unterteilt und dadurch einen neuen Funktionswert erhält. Danach wird *H* in 4 Subintervalle unterteilt, und es wird ein neuer Funktionswert ermittelt. Die Bestimmung eines Subintervalls *h* des Bulirsch-Stoer-Verfahrens ist unter Berücksichtigung von Gl. 4-9 in Gl. 4-10 dargestellt.

$$h_i = \frac{H}{n_i} \;\Big|\; \begin{matrix} n_0 = 2 \\ n_1 = 4 \\ n_2 = 6 \end{matrix} \tag{4-10}$$

Wenn mindestens 2 Funktionswerte vorliegen, wird eine Polynomextrapolation zwischen diesen beiden Funktionswerten durchgeführt, die einen extrapolierten Funktionswert ergibt. Liegt der Fehler ε der durchgeführten Extrapolation unter einem festgelegten Wert, kann das Bulirsch-Stoer-Verfahren für diese Funktion beendet werden.

Mathematische Formeln

In Gl. 4-11 ist ein Polynom $P_{j,n}$ mit dem Grad *n* dargestellt, das *n+1* Stützstellen in Form von Funktionswerten besitzt (y_0, y_1, y_2, …, y_n).

$$\begin{aligned} P_{j,n}(t) &= a_0 + a_1 \cdot t + a_2 \cdot t^2 + \ldots + a_n \cdot t^n \\ y_j &= P_{j,0} \end{aligned} \tag{4-11}$$

Wenn man 2 Stützstellen $y_0 = P_{0,0}$ und $y_1 = P_{1,0}$ hat (siehe auch Abb. 4-13a), ergibt sich das Polynom $P_{1,1}$ mit dem Grad 1 bzw. ein Polynom 1. Ordnung. Kommt nun eine dritte Stützstelle $y_2 = P_{2,0}$ hinzu, wird zunächst das Polynom 1. Ordnung $P_{2,1}$ mit der Stützstelle y_1 erzeugt. Anschließend folgt die Bestimmung des Polynoms 2. Ordnung $P_{2,2}$ anhand einer Subtraktion (siehe auch Abb. 4-13b) der beiden Polynome $P_{2,1}$ und $P_{1,1}$ (j = 2 und k = 2).

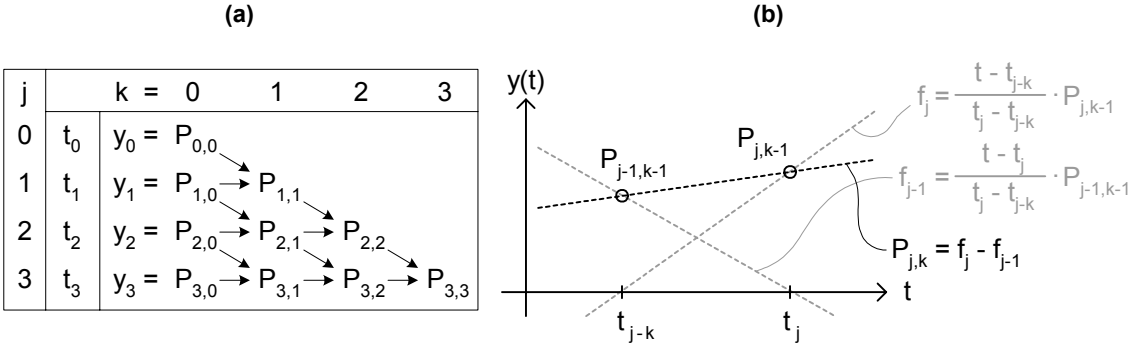


Abb. 4-13: Polynomextrapolation: (a) Polynomerzeugung anhand von 4 Stützstellen, (b) Bestimmung von $P_{j,k}$ durch Subtraktion der beiden Polynome $P_{j,k-1}$ und $P_{j-1,k-1}$

Einmal erzeugte Polynome werden folglich für die Erzeugung eines neuen Polynoms höherer Ordnung verwendet. Dadurch lässt sich ein Polynom mit den Indizes *j* und *k* ($P_{j,k}$) als Subtraktion zweier Polynome mit den Indizes *i* und *k-1* ($P_{j,k-1}$) und mit den Indizes *j-1* und *k-1* ($P_{j-1,k-1}$) ausdrücken (siehe Gl. 4-12).

$$\begin{aligned} P_{j,k} &= \frac{t - t_{j-k}}{t_j - t_{j-k}} \cdot P_{j,k-1} - \frac{t - t_j}{t_j - t_{j-k}} \cdot P_{j-1,k-1} = \frac{(t - t_{j-k}) \cdot P_{j,k-1} - (t - t_j) \cdot P_{j-1,k-1}}{t_j - t_{j-k}} \\ &= \frac{(t - t_{j-k} - t_j + t_j) \cdot P_{j,k-1} - (t - t_j - t_{j-k} + t_{j-k}) \cdot P_{j-1,k-1}}{t_j - t_{j-k}} \\ &= P_{j,k-1} + \frac{t - t_j}{t_j - t_{j-k}} \cdot (P_{j,k-1} - P_{j-1,k-1}) = P_{j-1,k-1} + \frac{t - t_{j-k}}{t_j - t_{j-k}} \cdot (P_{j,k-1} - P_{j-1,k-1}) \end{aligned} \tag{4-12}$$

Wenn in Gl. 4-12 $t = t_{j-k}$ entspricht, erhält man $P_{j,k} = P_{j-1,k-1}$, und wenn $t = t_j$ gilt, vereinfacht sich die Gleichung zu $P_{j,k} = P_{j,k-1}$. Die graphische Darstellung von Gl. 4-12 ist in

Abb. 4-13b zu sehen ($P_{j,k} = f_j - f_{j-1}$). Durch geeignete Umstellungen von Gl. 4-12 lassen sich die Abhängigkeiten der 3 Polynome anhand der beiden Größen $Q_{j,k}$ und $D_{j,k}$ ausdrükken (siehe Gl. 4-13 und Gl. 4-14).

$$P_{j,k} - P_{j,k-1} = \frac{t - t_j}{t_j - t_{j-k}} \cdot (P_{j,k-1} - P_{j-1,k-1}) = Q_{j,k} \tag{4-13}$$

$$P_{j,k} - P_{j-1,k-1} = \frac{t - t_{j-k}}{t_j - t_{j-k}} \cdot (P_{j,k-1} - P_{j-1,k-1}) = D_{j,k} \tag{4-14}$$

$$\begin{aligned}(P_{j,k-1} - P_{j-1,k-1}) &= (P_{j,k-1} - P_{j-1,k-1}) + (P_{j-1,k-2} - P_{j-1,k-2}) \\ &= (P_{j,k-1} - P_{j-1,k-2}) - (P_{j-1,k-1} - P_{j-1,k-2}) \\ &= D_{j,k-1} - Q_{j-1,k-1}\end{aligned} \tag{4-15}$$

Diese beiden Größen können nun durch eine rekursive Darstellung, wie in Gl. 4-16 zu sehen, formuliert werden [Stoe00], bei der $t = 0$ gilt, da beim Bulirsch-Stoer-Verfahren nur Funktionswerte zu extrapolieren sind, die denselben Zeitwert haben. Weiterhin können die Zeitwerte t_j und t_{j-k} durch die Subintervalle h der Bulirsch-Folge (siehe auch Gl. 4-10) ersetzt werden.

$$\begin{aligned}Q_{j,k} &= \frac{h_j}{h_{j-k} - h_j} \cdot (D_{j,k-1} - Q_{j-1,k-1}) \\ D_{j,k} &= \frac{h_{j-k}}{h_{j-k} - h_j} \cdot (D_{j,k-1} - Q_{j-1,k-1})\end{aligned} \tag{4-16}$$

Die Polynomdarstellung in Gl. 4-11 lässt sich nun als Summe von $Q_{n,k}$ Größen definieren, wie in Gl. 4-17 gezeigt.

$$\begin{aligned}P_{n,n} &= \sum_{k=0}^{n} Q_{n,k} \\ y_j &= Q_{j,0} = D_{j,0}\end{aligned} \tag{4-17}$$

Diese mathematische Formel wird für die Polynomextrapolation im Rahmen des numerischen Integrationsverfahrens nach Bulirsch-Stoer angewendet. Die Fehlerabschätzung bei der Polynomextrapolation $\varepsilon_{j,k}$ erfolgt nach Gl. 4-18 und darf den defnierten Fehler ε nicht überschreiten.

$$\varepsilon_{j,k} = |P_{j,k} - P_{j-1,k-1}| < \varepsilon \tag{4-18}$$

Die Funktionsweise der Polynomextrapolation lässt sich anhand eines Zahlenbeispiels mit 3 Funktionswerten (y_0, y_1 und y_2) verdeutlichen. Zuerst werden nach Gl. 4-16 und Gl. 4-17 die Polynomdarstellung für jeden der 3 Funktionswerte angeben. **Zahlenbeispiel**

$$P_{0,0} = Q_{0,0} = y_0 \tag{4-19}$$

$$\begin{aligned}P_{1,1} = Q_{1,0} + Q_{1,1} &= Q_{1,0} + \frac{h_1}{h_0 - h_1} \cdot (D_{1,0} - Q_{0,0}) \\ &= y_1 + \frac{h_1}{h_0 - h_1} \cdot (y_1 - y_0)\end{aligned} \tag{4-20}$$

$$\begin{aligned}P_{2,2} = Q_{2,0} + Q_{2,1} + Q_{2,2} &= Q_{2,0} + \frac{h_2}{h_1 - h_2} \cdot (D_{2,0} - Q_{1,0}) + \frac{h_2}{h_0 - h_2} \cdot (D_{2,1} - Q_{1,1}) \\ &= Q_{2,0} + \frac{h_2}{h_1 - h_2} \cdot (D_{2,0} - Q_{1,0}) + \frac{h_2}{h_0 - h_2} \cdot \left[\left\{ \frac{h_1}{h_1 - h_2} \cdot (D_{2,0} - Q_{1,0}) \right\} - \left\{ \frac{h_1}{h_0 - h_1} \cdot (D_{1,0} - Q_{0,0}) \right\} \right] \\ &= y_2 + \frac{h_2}{h_1 - h_2} \cdot (y_2 - y_1) + \frac{h_2}{h_0 - h_2} \cdot \left[\left\{ \frac{h_1}{h_1 - h_2} \cdot (y_2 - y_1) \right\} - \left\{ \frac{h_1}{h_0 - h_1} \cdot (y_1 - y_0) \right\} \right]\end{aligned} \tag{4-21}$$

Wenn man nun für $y_0 = 0{,}21$, $y_1 = 0{,}2$, $y_2 = 0{,}195$, $\varepsilon = 0{,}01$ und $H = 60$ ms ($\Rightarrow h_0 = 30$ ms, $h_1 = 15$ ms, $h_2 = 10$ ms) annimmt, erhält man nach Gl. 4-19 $P_{0,0} = 0{,}21$. Nach Gl. 4-20 ist $P_{1,1} = 0{,}2+1\cdot(0{,}2-0{,}21) = 0{,}19$ und der Fehler $\varepsilon_{1,1} = |0{,}19-0{,}21| = 0{,}02$ (siehe

Gl. 4-18). Schließlich ergibt $P_{2,2}$ = 0,195+2·(0,195-0,2)+0,5·[{3·(0,195-0,2)}-{1·(0,2-0,21)}] = 0,1825 und der Fehler beträgt $\varepsilon_{2,2}$ = |0,1825-0,19| = 0,0075.

Hardware Mapping

In den Gleichungen Gl. 4-16 und Gl. 4-17 erkennt man, dass für die Polynomextrapolation ein rekursives Berechnungsverfahren notwendig ist. Somit ist eine Fließbandverarbeitung wie bei der MMID-Integration, die nach einer gewissen Latenzzeit immer neue Funktionswerte aufnehmen kann nicht realisierbar.

PZEXTR von W_{KL}

Das HW-Modul für die Durchführung der Polynomextrapolation der Synapsengewichte ist in Abb. 4-14 dargestellt und ist mit dem HW-Modul für die Durchführung der Polynomextrapolation von Membranpotentialen identisch.

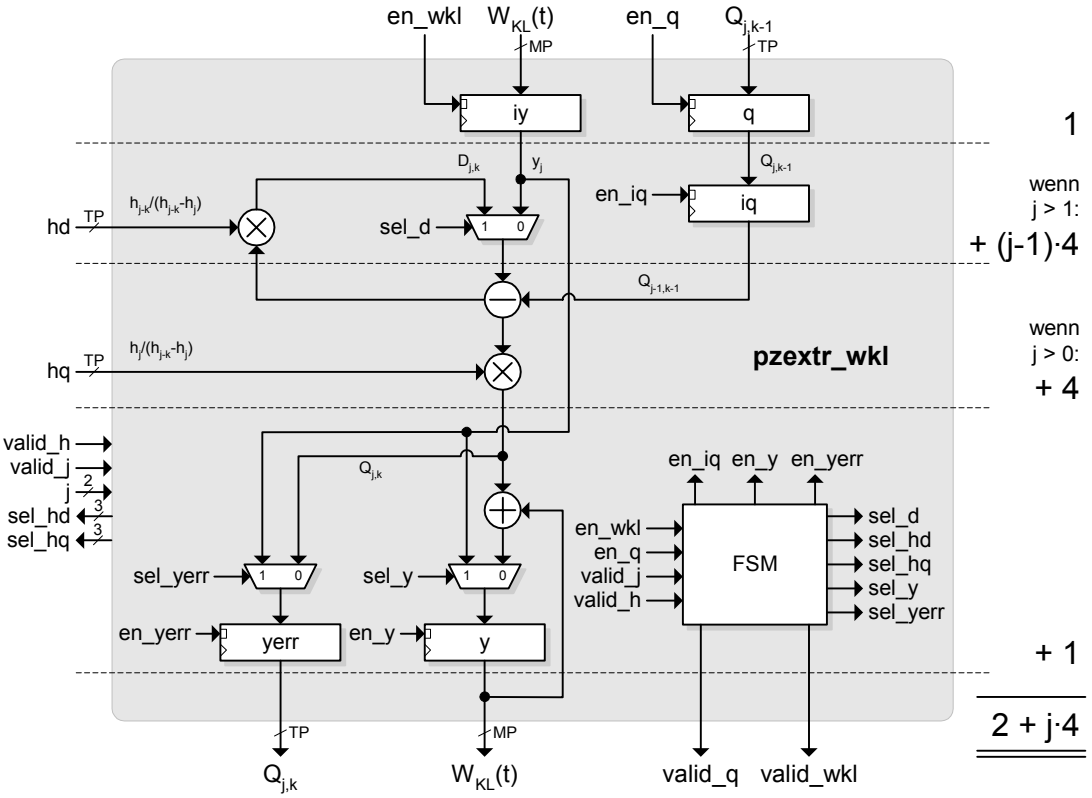


Abb. 4-14: HW-Modul (`pzextr_wkl`) für die Polynomextrapolation der Synapsengewichte W_{KL}

Im ersten Takt wird der durch die MMID-Integration neu berechnete Funktionswert und der Wert $Q_{j,k-1}$, der aus der Berechnung einer vorher durchgeführten Polynomextrapolation resultiert, in den Registern *iy* und *q* zwischengespeichert. Dann erfolgt eine Multiplikation von 4 Takten, wenn vorher schon einmal eine Polynomextrapolation durchgeführt wurde (j > 0), ansonsten wird der Funktionswert direkt in den Ausgangsregistern *yerr* und *y* (mit *sel_yerr* = *sel_y* = 1) abgelegt.

Bei der Extrapolation von 2 Funktionswerten (*j* = 1) wird der neue Funktionswert im Register *iy* mit dem vorherigen Funktionswert im Register *iq* (*sel_d* = 0) subtrahiert, und danach folgt die Multiplikation mit der Variablen $hq = h_1/(h_0-h_1)$ (siehe auch Gl. 4-20). Zeitgleich wird der neue Funktionswert ins Ausgangsregister *y* (*sel_y* = 1) kopiert. Nach der Multiplikation wird abschließend das Ergebnis zum einen ins Register *yerr* und zum anderen mit dem Registerinhalt *y* addiert und wieder ins Ausgangsregister *y* kopiert (*sel_yerr* = *sel_y* = 0).

Bei der Extrapolation von mehr als 2 Funktionswerten (j > 1) müssen insgesamt j-1 Multiplikationen durchgeführt werden. Wenn beispielsweise über 3 Funktionswerte, wie im Abschn. "Zahlenbeispiel" auf Seite 79, extrapoliert wird, stehen als Eingangsdaten neben dem 3. Funktionswert als Synapsengewicht $W_{KL}(t)$, die Variablen $Q_{1,0}$ und $Q_{1,1}$ aus der vorangegangenen Polynomextrapolation zur Verfügung (siehe auch Gl. 4-20). Zuerst wird der 3. Funktionswert in das Ausgangsregister *y* kopiert (*sel_y* = 1) und zeitgleich starten

die Multiplikationen der Variablen $hd = h_2/(h_1-h_2)$ (zur Berechnung von $D_{2,1}$) und $hq = h_1/(h_1-h_2)$ (zur Berechnung von $Q_{2,1}$) jeweils mit der Differenz zwischen Funktionswert (entspricht der Variablen $D_{2,0}$) und $Q_{1,0}$ (siehe auch 2. Zeile in Gl. 4-21). Danach erfolgt eine zweite Multiplikation der Variablen $hq = h_2/(h_0-h_2)$ mit der Differenz zwischen $D_{2,1}$ und $Q_{1,1}$ während $Q_{2,1}$ in das Register *y*, welches ja den 3. Funktionswert beinhaltet, aufaddiert wird (*sel_y* = 0). Das Ergebnis der zweiten Multiplikation wird schließlich noch in das Register *y* aufaddiert, und die Polynomextraploation von 3 Funktionswerten ist abgeschlossen

Die Latenzzeit t_{PZEXTR} für die Polynomextrapolation sowohl für Synapsengewichte als auch für Membranpotentiale beträgt, wie in Gl. 4-22 dargestellt 4 Takte.

$$t_{PZEXTR} = 4 \qquad \textbf{(4-22)}$$

Pipeline-Dividierer

Die durchzuführenden Divisionen der Variablen *hd* und *hq* wird mit Hilfe eines Pipeline-Dividierers, der mit dem Werkzeug CORE Generator von Xilinx generiert wurde, bewerkstelligt und hat eine Latenzzeit von *TP* + 4 Takten. *TP* entspricht der Bitbreite von Zeitparametern (und *MP* entspricht der Bitbreite von Modellparametern) und hat den Wert 32 (siehe auch Abb. 4-5 auf Seite 66). Der Pipeline-Dividierer wird mit der doppelten Systemfrequenz getaktet und benötigt somit (32+4)/2 = 18 Takte. Die Divisionen für *hq* ($h_1/(h_0-h_1)$, $h_2/(h_0-h_2)$, $h_2/(h_1-h_2)$, ...) und *hd* (h_1/h_1-h_2, h_2/h_2-h_1, h_2/h_2-h_3, ...), werden parallel ausgeführt, nur wenn sich das Integrationsintervall *H* ändert, und werden anhand eines Kontrollsignals (siehe *sel_hd und sel_hq* in Abb. 4-14) ausgewählt. Eine Verringerung von *H* resultiert immer dann, wenn im Zustand *Next-Spike* in der NSC-Phase ein Neuron gefunden wird, das innerhalb des Integrationsintervalls *H* zu pulsen beginnt.

4.3.3 Nullstellensuchverfahren nach Newton

Für die Neuronenzustandsberechnung ist weiterhin ein Nullstellensuchverfahren notwendig, wenn nach der numerischen Integration das Membranpotential den Schwellwert überschritten hat. Mit dem Nullstellensuchverfahren nach Newton wird der Zeitpunkt bestimmt, an dem das Membranpotential des Neurons exakt den Schwellwert erreicht hat.

Mathematische Formeln

In Abb. 4-15 ist die Differenz *y(t)* des Membranpotentials $a_K(t)$ und des Schwellwertes θ dargestellt. Wenn für $a_K(t)$ nach der numerischen Integration, welche im Integrationsintervall *H* vom Zeitpunkt t_0 bis zum Zeitpunkt t_i durchgeführt wurde, die Bedingung $a_K(t_i) > \theta$ erfüllt ist, wird durch ein anschließendes Iterationsverfahren die Nullstelle bestimmt, um den exakten Feuerzeitpunkt t_f mit der Bedingung $a_K(t_f) = \theta$ zu ermitteln.

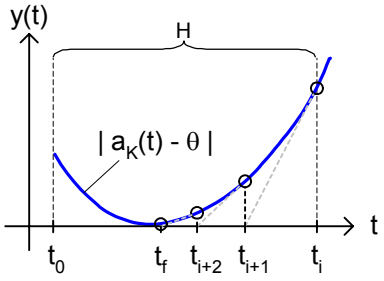


Abb. 4-15: Nullstellensuchverfahren nach Newton

Anhand der Ableitung am Zeitpunkt t_i lässt sich eine Geradengleichung formulieren, um den nächsten Zeitpunkt t_{i+1} (Nulldurchgang der Geradengleichung) zu bestimmen, der näher an der gesuchten Nullstelle liegt. Dadurch ergibt sich die Iterationsvorschrift nach Newton für die Suche der Nullstelle [Nume05]:

$$y'(t_i) = \frac{y(t_i)}{t_i - t_{i+1}} \Leftrightarrow t_{i+1} = t_i - \frac{y(t_i)}{y'(t_i)} \qquad \textbf{(4-23)}$$

Wenn die beiden Zeitpunkte t_{i+1} und t_i nahe genug beieinander liegen, kann das Iterationsverfahren beendet werden, und die Nullstelle ist mit $y(t_{i+1})$ gefunden. Die Abbruchbedingung für die Nullstellensuche lautet,

$$|t_{i+1} - t_i| < 2^{-f} \cdot |t_{i+1}| \qquad \textbf{(4-24)}$$

wobei *f* die Anzahl der binären Nachkommastellen darstellt.

Hardware Mapping

In der Gleichung Gl. 4-23 erkennt man, dass für die Bestimmung des Zeitpunkts t_{i+1} neben der Funktion $y(t_i) = |a_K(t_i)-\theta|$ die Ableitung $y'(t_i)$ notwendig ist, die der Ableitung des Membranpotentials a_K entspricht (siehe auch Gl. 4-7 auf Seite 75).

NEWTON von a_K

Die Hardware-Abbildung zur Bestimmung des Feuerzeitpunkts t_f ist in Abb. 4-16 dargestellt.

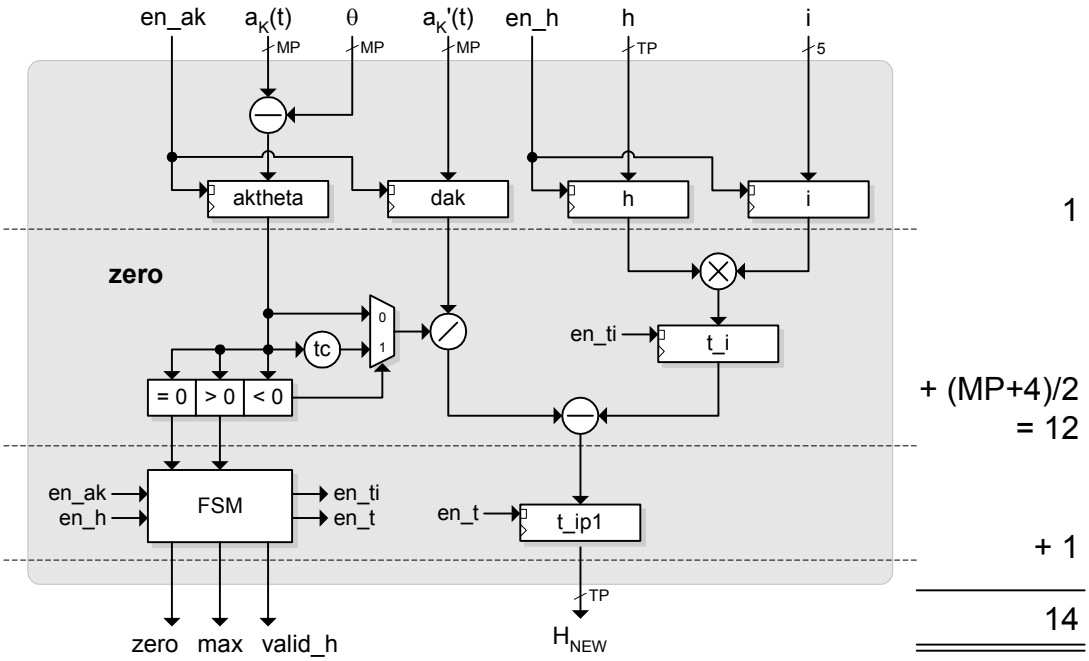


Abb. 4-16: HW-Modul (`zero`) für die Definition eines neuen Integrationsintervalls H_{NEW} zur Bestimmung des Feuerzeitpunkts t_f ($a_K(t_f) = \theta$)

Im ersten Takt wird die Differenz von Membranpotential und des Schwellwertes θ (*aktheta*), die Ableitung des Membranpotentials (*dak*), das Subintervall *h* und der Bearbeitungsindex *i* der MMID (siehe auch Gl. 4-5 auf Seite 75) zwischengespeichert. Dann folgt die Division von *aktheta* und *dak*, die anhand eines Pipeline-Dividieres (siehe auch Abschn. "Pipeline-Dividierer" auf Seite 81) mit einer Latenzzeit von *MP* + 4 Takten durchgeführt wird. *MP* entspricht der Bitbreite von Modellparametern (und *TP* entspricht der Bitbreite von Zeitparametern) und hat den Wert 20 (siehe auch Abb. 4-5 auf Seite 66). Mit doppelter Systemfrequenz getaktet benötigt der Pipeline-Dividierer also (20+4)/2 = 12 Takte. Falls *aktheta* kleiner als Null ist, wird das Zweierkomplement für die Ermittlung des Quotienten von *aktheta* und *dak* verwendet. Zeitgleich erfolgt die Multiplikation von *h* und *i*, die 4 Takte benötigt, um den Ausgangszeitpunkt t_i zu bestimmen (siehe auch Abb. 4-15). Schließlich erfolgt die Subtraktion von t_i mit dem Quotienten zur Bestimmung des neuen durchzuführenden Integrationsintervalls H_{NEW}, welches dem Zeitpunkt t_{i+1} entspricht. Die Bestimmung eines neuen Integrationsintervalls, welches näher am Feuerzeitpunkt des Neurons liegt, beträgt somit 14 Takte, nachdem Membranpotentialwert und Ableitung des Membranpotentials (des Zeitpunkts t_i) zur Verfügung stehen. Anhand dieses neuen Integrationsintervalls muss nun die numerische Integration erneut durchgeführt werden, um einen neuen Membranpotentialwert mit zugehöriger Ableitung (des Zeitpunkts t_{i+1}) zu bestimmen. Wenn der Membranpotentialwert immer noch über dem Schwellwert liegt, wird anhand des HW-Moduls in Abb. 4-16 ein neues Integrationsintervall bestimmt.

Im folgenden wird nun beschrieben, wann die numerische Integration nach Bulirsch-Stoer und das Nullstellensuchverfahren im Rahmen der Neuronenzustandsberechnung zum Einsatz kommen.

4.3.4 Einsatz bei der Neuronenzustandsberechnung

Das vorgestellte numerische Integrationsverfahren kommt in der NSC-Phase in den Zuständen *Next-Spike* und *Update* zum Einsatz (siehe auch Abb. 4-7 auf Seite 71).

Aktualisierung der Neuronenzustände

Im Zustand *Update* werden alle Zustände der Neuronen aktualisiert, die sich nicht im Ruhezustand befinden (siehe auch Tab. 4-1 auf Seite 70). Der nächste Emulationsereigniszeitpunkt t_{EVENT} ergibt sich aus dem früheren Zeitpunkt von einem Puls-Start-Ereignis t_{FIRE} und von einem Puls-Stop-Ereignis zum Zeitpunkt t_{FEL}, dessen Zeitpunkt aus der FEL gelesen wird. Das Integrationsintervall H für die numerische Integration ergibt sich aus der Differenz von t_{EVENT} und dem aktuellen Emulationszeitpunkt t_0. Die numerische Integration wird für alle gültigen Neuronen in der DEL durchgeführt bis der Lesezeiger RP den Wert des Schreibezeigers der DEL WP_{DEL} erreicht hat (siehe Abb. 4-17).

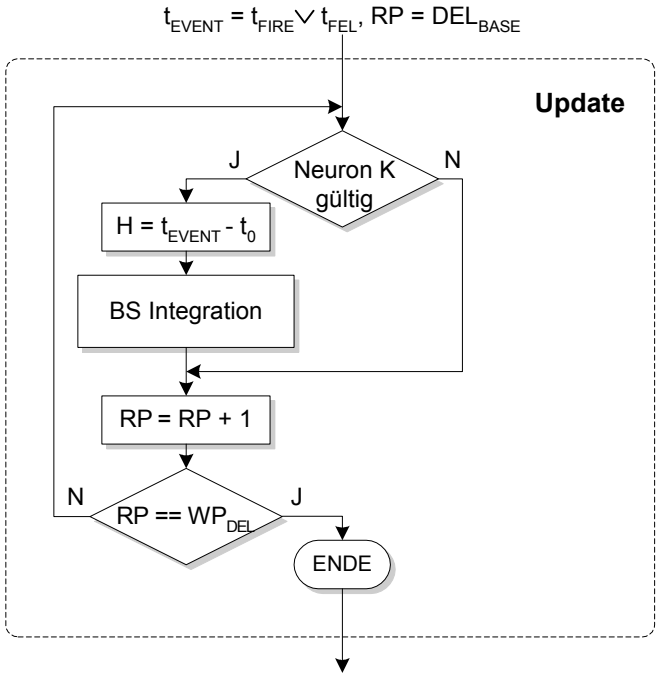


Abb. 4-17: Flussdiagramm des *Update* Zustands in der Neuronenzustandsphase (t_{EVENT} = nächster Emulationsereigniszeitpunkt, DEL_{BASE} = Basisadresse der DEL, RP = Read Pointer, WP_{DEL} = Write Pointer der DEL)

Suche nach Puls-Start-Ereignissen

Im Zustand *Next-Spike* wird mit Hilfe der numerischen Integration nach Puls-Start-Ereignissen gesucht. Dabei wird zunächst t_{FIRE} mit t_{FEL} gleichgesetzt, da zwischen den Zeitpunkten t_0 und t_{FEL} nach einem Puls-Start-Ereignis gesucht werden soll. Wenn das Neuron in der DEL gültig ist, wird die präsynaptische Pulsaktivität in Form eines Topologie-Vektors benötigt, damit die numerische Integration durchgeführt werden kann. Wenn der Topologie-Vektor des Neurons Null ist, das Neuron also keine präsynaptischen Pulse empfängt, kann der Zeitpunkt eines Puls-Start-Ereignisses nach Gl. 4-1 auf Seite 70 ohne numerische Integration bestimmt werden (siehe Pfad ② in Abb. 4-18). Wenn der ermittelte Zeitpunkt kleiner ist als t_{FIRE}, wird dieser Zeitpunkt als aktueller Puls-Start-Ereigniszeitpunkt definiert und die Anzahl der gefundenen Puls-Start-Ereignisse $n_{FIRE} = 1$ gesetzt (siehe Pfad ④). Ist der ermittelte Zeitpunkt mit t_{FIRE} identisch, wird n_{FIRE} um 1 erhöht (siehe Pfad ⑤). Ansonsten wird das nächste Neuron aus der DEL herangezogen (siehe Pfad ⑥). Wenn der Topologie-Vektor allerdings ungleich Null ist, wird die numerische Integration durchgeführt (siehe Pfad ① in Abb. 4-18). Dabei gilt wieder für das Integrationsin-

tervall $H = t_{FIRE} - t_0$. Stellt sich nach der numerischen Integration heraus, dass das Membranpotential a_K des Neurons größer oder gleich dem Schwellwert θ ist, wird das Nullstellensuchverfahren nach Newton durchgeführt, um den genauen Zeitpunkt zu ermitteln, an dem das Membranpotential den Schwellwert überschreitet (siehe Pfad ③). Wenn a_K nach der Integration genau θ entspricht, braucht das Nullstellensuchverfahren nicht durchgeführt zu werden, und der ermittelte Puls-Start-Ereigniszeitpunkt kann mit dem aktuellen Puls-Start-Ereigniszeitpunkt t_{FIRE} verglichen werden. Wenn der Lesezeiger *RP* den Wert des Schreibezeigers der DEL WP_{DEL} erreicht hat, ist der *Next-Spike* Zustand beendet.

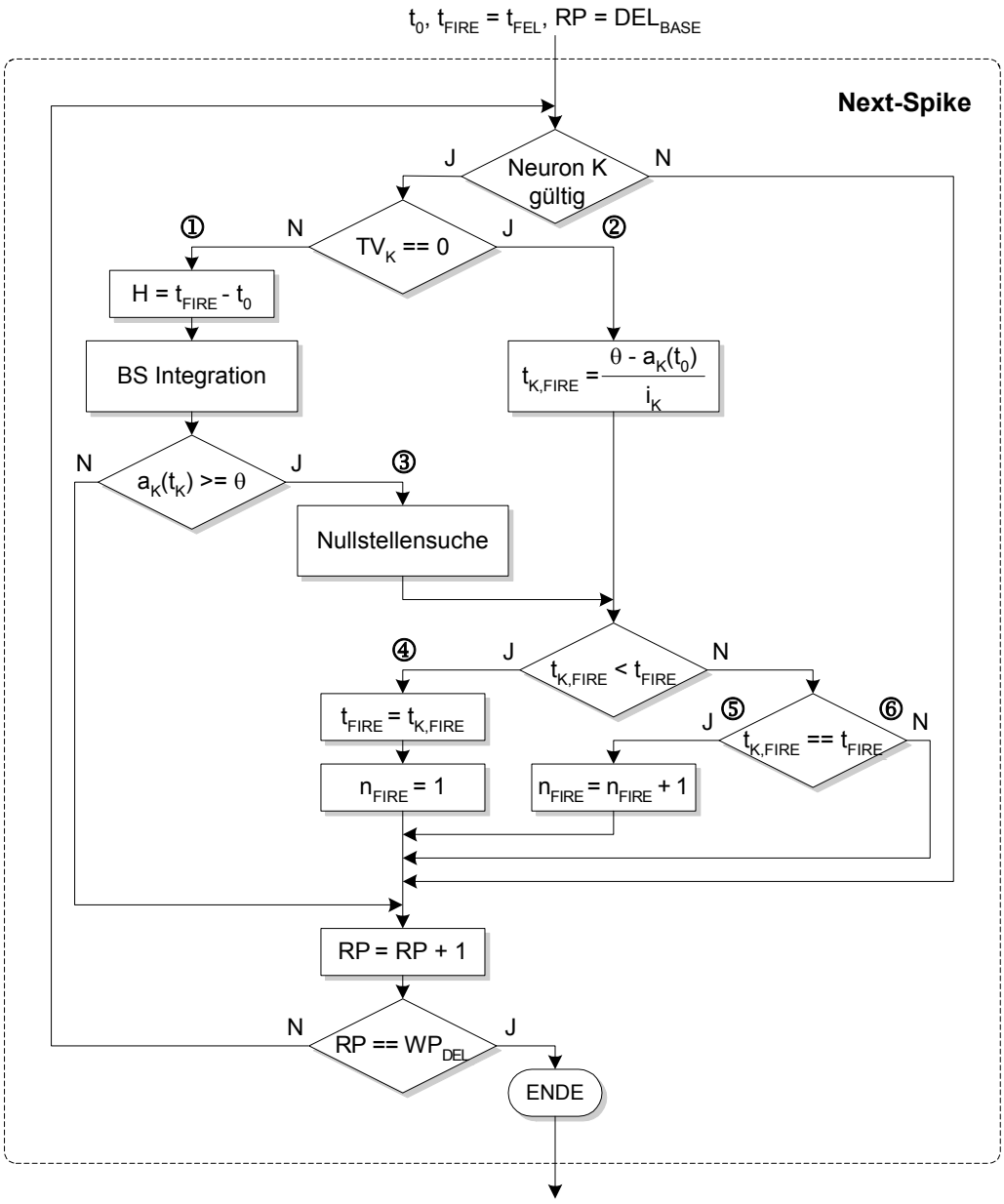


Abb. 4-18: Flussdiagramm des *Next-Spike* Zustands in der Neuronenzustandsphase (t_{FIRE} = aktueller Puls-Start-Ereigniszeitpunkt, t_{FEL} = aktueller Puls-Stop-Ereigniszeitpunkt, n_{FIRE} = Anzahl der Puls-Start-Ereignisse)

Nachdem nun das numerische Integrationsverfahren im Rahmen der Neuronenzustandsberechnung beschrieben wurde, werden im folgenden die Netzwerkarchitekturen vorgestellt, die von der Netzwerktopologieberechnung verwaltet werden müssen.

4.4 Netzwerkarchitekturen für das Neuronenmodell mit adaptiven Synapsengewichten

Die Netzwerktopologieberechnung muss unterschiedliche reguläre und irreguläre Verbindungsstrukturen berücksichtigen können. Für das Neuronenmodell mit adaptiven Synapsengewichten sind für die Flecken- und Gradientendetektion 2 unterschiedliche Netzwerkarchitekturtypen zu beachten: die Kaskaden-Architektur und die Gabor-Architektur. Die Neuronen innerhalb der Netzwerkarchitekturen werden anhand einer Neuronennummer gekennzeichnet und zusätzlich durch Koordinaten beschrieben, da diese Neuronenbeschreibung eine effiziente Ermittlung von den zugehörigen prä- und postsynaptischen Neuronen ermöglicht. Bei der Koordinatendarstellung werden die Neuronen Netzwerkschichten zugeordnet und im Rahmen eines zweidimensionalen kartesischen Koordinatensystems mit x- und y-Koordinate dargestellt. Die Neuronennummer wird anhand der Durchnummerierung der Neuronen, angefangen bei der ersten Netzwerkschicht bis hin zur letzten Netzwerkschicht, vergeben.

4.4.1 Kaskaden-Architektur

Die Kaskaden-Architektur beschreibt eine Netzwerkarchitektur, bei der die Netzwerkschichten, wie bei einer Kaskade in Abb. 4-19, hintereinander angeordnet sind.

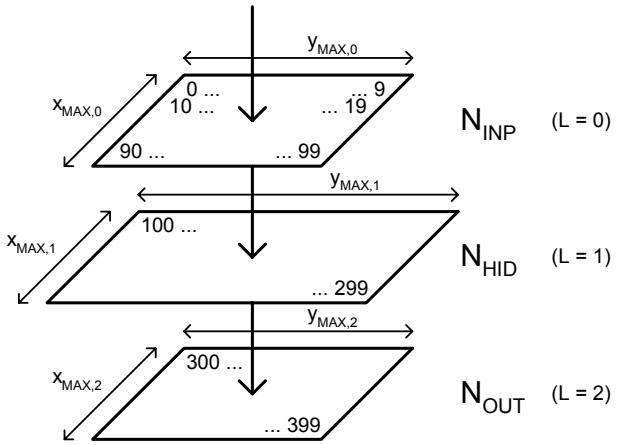


Abb. 4-19: Darstellung einer Kaskaden-Architektur (3 Netzwerkschichten mit insgesamt 400 Neuronen)

Dabei werden ausschließlich reguläre Verbindungsstrukturen verwendet. Reguläre Verbindungsstrukturen, die berücksichtigt werden, sind: Vierer- und Achter-Nachbarschaft, Vollvernetzung, divergente und konvergente Verbindungen und Punkt-zu-Punkt-Verbindungen. Eine Punkt-zu-Punkt-Verbindung liegt vor, wenn ein Neuron der Quellschicht mit einem Neuron einer gleich großen Zielschicht verbunden ist. Dabei haben beide Neuronen die gleiche Position innerhalb der jeweiligen Netzwerkschicht (identische x- und y-Koordinaten). Bei divergenten (auseinandergehenden) Verbindungen wird das Neuron der Quellschicht mit einer Gruppe von Neuronen in der Zielschicht verbunden. Die Größe der Neuronengruppe in der Zielschicht muss dabei als rechteckiges Projektionsfeld mit x- und y-Größe angegeben werden. Bei einer Nicht-Überlappung der Projektionsfelder wird die Zielschicht um den Faktor der x-Größe des Projektionsfeldes in x-Richtung und um den Faktor der y-Größe des Projektionsfeldes in y-Richtung größer. Bei konvergenten (zusammenlaufenden) Verbindungen wird eine Gruppe von Neuronen in der Quellschicht mit einem einzigen Neuron in der Zielschicht verbunden. Die Größe der Neuronengruppe in der Quellschicht muss ebenfalls als rechteckiges Feld mit x- und y-Größe definiert werden. Bei einer Nicht-Überlappung der Felder wird die Zielschicht um den Faktor der x-Größe des Feldes in x-Richtung und um den Faktor der y-Größe des Feldes in y-Richtung kleiner.

Lateral/Extern

Bei Netzwerkverbindungen wird zwischen lateralen und externen Verbindungen unterschieden. Laterale Verbindungen sind Verbindungen zwischen Neuronen derselben Netzwerkschicht (Quell- und Zielschicht sind identisch) und können als Vierer-, Achter-Nachbarschaft oder Vollvernetzung ausgelegt werden. Externe Verbindungen sind Verbindungen zwischen Neuronen unterschiedlicher Netzwerkschichten (Quell- und Zielschicht sind verschieden), wobei zwischen afferenten (hinführenden) externen Verbindungen und efferenten (wegführenden) externen Verbindungen unterschieden wird.

Neuronencodierung

Die Neuronencodierung bei einer Kaskaden-Architektur erfolgt mit 32 Bit und ist in Abb. 4-20 illustriert. Das oberste Bit (*STATE*) gibt an, ob sich das Neuron im Pulszustand befindet oder nicht. Dann folgt eine Typendefinition (*TYPE*) der Netzwerkschicht, bei der zwischen Eingangsschicht (*Input Layer*), Ausgangsschicht (*Output Layer*) und Zwischenschicht (*Hidden Layer*) differenziert wird. Bei einem Neuron der Eingangsschicht (N_{INP}) muss ein externer Eingangsreiz berücksichtigt werden. Bei Neuronen der Ausgangsschicht (N_{OUT}) existieren keine postsynaptischen Neuronen, die durch Pulse von Ausgangsschicht-Neuronen erregt werden können. Neuronen einer Zwischenschicht (N_{HID}) verfügen stets über afferente und efferente externe Netzwerkverbindungen. Eine Schichtnummer von 5 Bit (*LAYER*) erlaubt eine maximale Anzahl von 32 Netzwerkschichten. Die jeweilige Codierung von 12 Bit für x-Koordinate (*X*) und y-Koordinate (*Y*) ermöglicht einen maximalen Koordinatenwert von 2^{12}-1 = 4095.

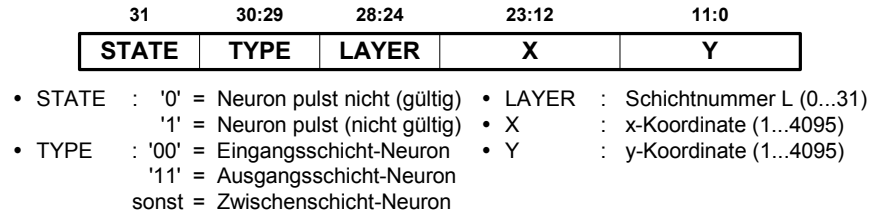


Abb. 4-20: Neuronencodierung bei einer Kaskaden-Architektur

4.4.2 Gabor-Architektur

Die Gabor-Architektur besteht aus einer Eingangsschicht und einem oder mehreren Gabor-Detektoren, die sich jeweils aus 8 Netzwerkschichten zusammensetzen (siehe Abb. 4-21).

RF/Gabor-Masken

Für die Gradientendetektion eines Eingangsbildes werden RF-Masken, die man auch als Gabor-Masken bezeichnet, überlappend auf der Eingangsschicht verteilt. Als RF-Masken werden cortikale RFs mit 2 oder 3 Aktivitätsbereichen (siehe auch Kapitel 2.3.1.2 auf Seite 26) bezeichnet, die Gradienten mit einer bestimmten Winkelausrichtung im Eingangsbild detektieren. Jeweils 2 Neuronen der Eingangsschicht bilden innerhalb einer RF-Maske ein Gradientenpaar, wenn sie sich in unterschiedlichen Aktivitätsbereichen innerhalb der RF-Maske befinden. Für die Definition eines Gradientenpaares werden die relativen x- und y-Koordinaten der beiden Neuronen innerhalb der RF-Maske im Topologiespeicher (TPM) abgelegt. Für eine effiziente Gradientendetektion sind für eine RF-Maske mehrere Gradientenpaare vorzusehen, deren Definitionen wie die Definition von irregulären Netzwerkverbindungen jeweils expilizit im TPM abgelegt werden.

Korrelation

Jedes Gradientenpaar einer RF-Maske wird mit einem Korrelations-Neuron (N_{COR}) verbunden, mit dem es einen Puls-Korrelations-Detektor (siehe Kapitel 2.2.4.1 auf Seite 20) bildet. Die Anzahl der Korrelations-Neuronen entspricht somit der Anzahl der Gradientenpaare, die in einer RF-Maske verwendet werden. Die Korrelations-Neuronen einer RF-Maske werden zu einem rechteckigen RF ($x_{GDP} \times y_{GDP}$) in einer Korrelationsschicht zusammengefasst.

Dekorrelation

Die Korrelations-Neuronen werden dann in der nachfolgenden Dekorrelationsschicht mit Dekorrelationsneuronen (N_{DEC}) verbunden und bilden zusammen mit dem Gradientenpaar in der Eingangsschicht einen Puls-Dekorrelations-Detektor (siehe Kapitel 2.2.4.2

auf Seite 21). Nur bei einer vorliegenden Dekorrelation der Pulsfolgen von 2 Eingangsschicht-Neuronen liegt ein Intensitätsunterschied von Eingangsreizen vor, der zu einer Detektion eines Gradienten im Eingangsbild führt. Die Ausgänge der Puls-Dekorrelations-Detektoren werden in der nächsten Netzwerkschicht durch ein Summen-Neuron (N_{SUM}) aufsummiert. Die Anzahl der Summen-Neuronen entspricht somit der Anzahl der RFs, die in der Korrelations- bzw. Dekorrelationsschicht vorhanden sind.

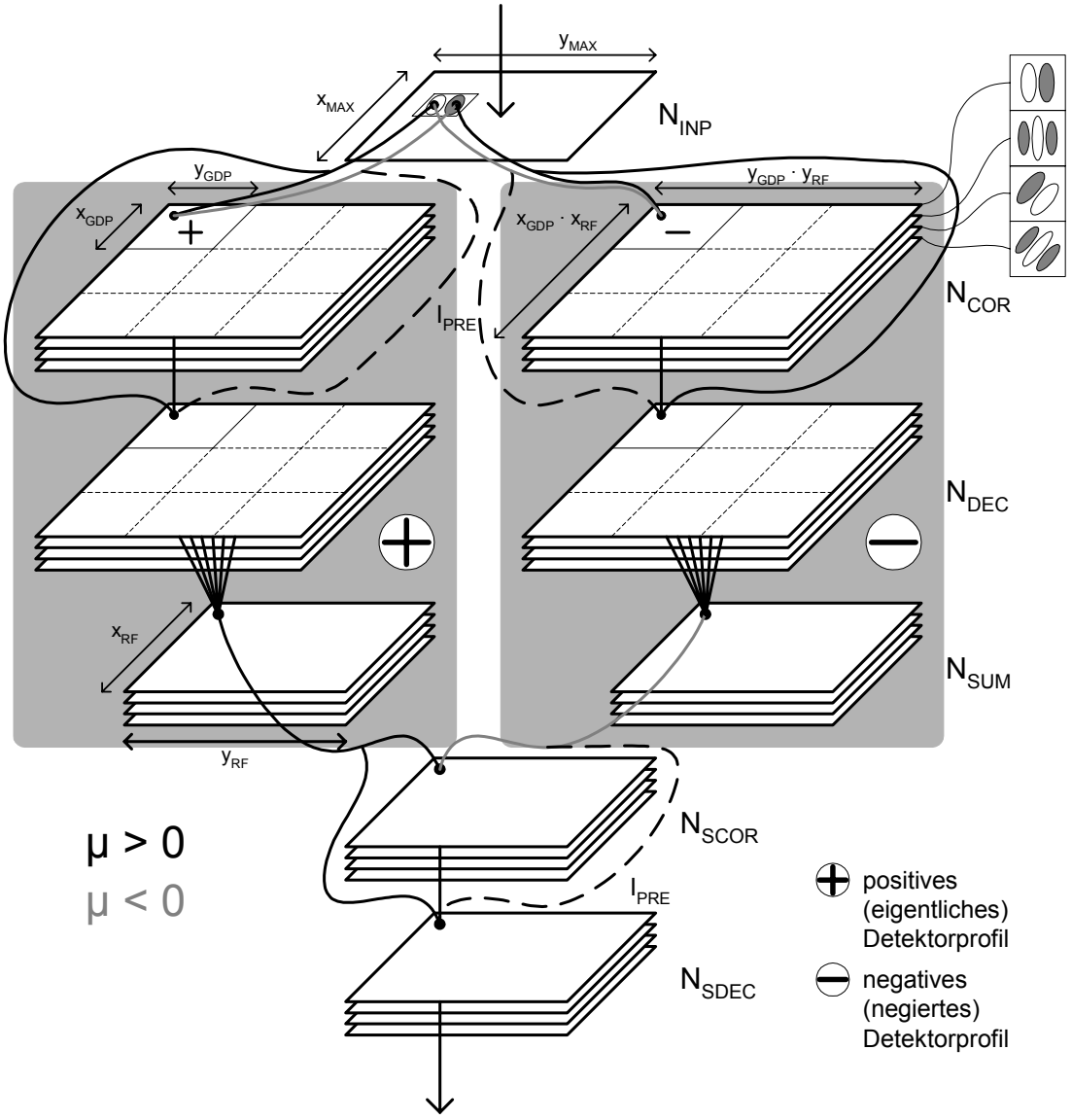


Abb. 4-21: Darstellung einer Gabor-Architektur mit 4 Gabor-Detektoren

2 Detektorprofile

In Abb. 4-22 sind 4 Gradientenpaare eines Gabor-Detektors dargestellt, die sensitiv auf vertikale Kanten reagieren. Die Pulsantworten für die Detektion einer schrägen Kante in Abb. 4-22a und dem Ende einer vertikalen Kante in Abb. 4-22b sind identisch (die Gradientenpaare 3 und 4 generieren jeweils eine Pulsantwort), obwohl die Detektion einer vertikalen Kante eine qualitativ höhere Pulsantwort erzielen sollte. Durch den Einsatz eines negierten Detektorprofils wird die Qualität der Pulsantwort verbessert. Hierfür werden Korrelations-, Dekorrelations- und Summenschicht für die Detektion der negierten Puls-Dekorrelation nochmals ausgelegt (siehe rechte Seite in Abb. 4-21). Durch eine Subtraktion der Pulsantworten beider Detektorprofile wird nun die Pulsantwort des Gabor-Detektors insgesamt bestimmt. Diese Subtraktion wird durch einen weiteren Puls-Dekorrelations-Detektor erreicht, der mit den Summen-Neuronen der beiden Detektorprofile, einem Summenkorrleations-Neuron (N_{SCOR}) und einem Summendekorrelations-Neuron (N_{SDEC}) gebildet wird. Bei der Detektion der schrägen Kante entspricht die

Pulsantwort des negierten Detektorprofils in Abb. 4-22c dem des eigentlichen Detektorprofils, so dass nach der Subtraktion keine Pulsantwort mehr existiert. Bei der Detektion des Endes der vertikalen Kante entsteht beim negierten Detektorprofil in Abb. 4-22d keine Pulsantwort. Die Pulsantwort insgesamt entspricht somit dem des eigentlichen Detektorprofils. Durch den Einsatz von 2 Detektorprofilen wird eine annähernd ideale Filterung nach der Gabor-Funktion erzielt [Heit04].

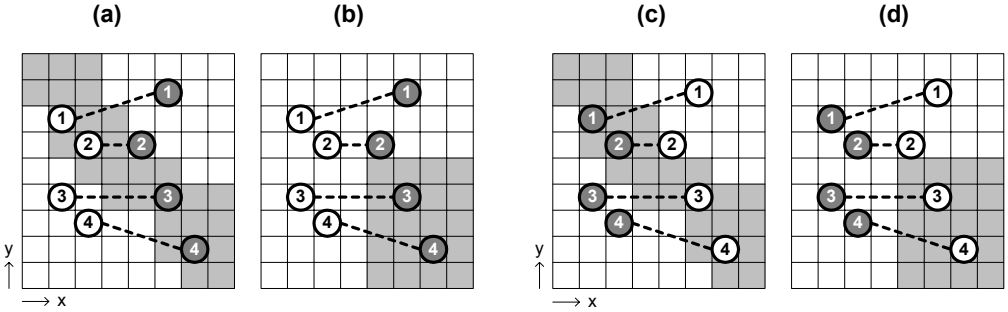


Abb. 4-22: Darstellung von 4 Gradientenpaaren eines Gabor-Detektors:
(a) Detektion einer schrägen Kante, (b) Detektion des Endes einer vertikalen Kante,
(c) negiertes Detektorprofil von (a), (d) negiertes Detektorprofil von (b)

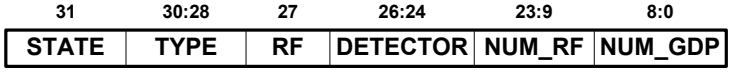
Ein Gabor-Detektor besteht folglich aus insgesamt 8 Netzwerkschichten (2 Korrelationsschichten, 2 Dekorrelationsschichten, 2 Summenschichten, eine Summenkorrelationsschicht und eine Summendekorrelationsschicht). Das eigentliche Detektorprofil eines Gabor-Detektors wird als positives Detektorprofil und das negierte Detektorprofil als negatives Detektorprofil bezeichnet. Die Anzahl an Neuronen N_D und Synapsen S_D eines Gabor-Detektors sind von der Anzahl der Gradientenpaare innerhalb einer RF-Maske n_{GDP} und von der Anzahl der zugehörigen RF-Masken in der Eingangsschicht n_{RF} abhängig und den folgenden beiden Gleichungen zu entnehmen:

$$N_D = 4 \cdot n_{GDP} \cdot n_{RF} + 4 \cdot n_{RF} = n_{RF} \cdot 4 \cdot (n_{GDP} + 1) \qquad \textbf{(4-25)}$$

$$S_D = 2 \cdot (5 \cdot n_{GDP} \cdot n_{RF} + n_{GDP} \cdot n_{RF}) + 5 \cdot n_{RF} = n_{RF} \cdot (12 \cdot n_{GDP} + 5) \qquad \textbf{(4-26)}$$

Neuronencodierung

Die Neuronencodierung bei einer Gabor-Architektur ist in Abb. 4-23 zusammengefasst. Im Vergleich zur Kaskaden-Architektur muss die Typendefinition der Netzwerkschichten um 1 Bit erweitert werden, damit die insgesamt 6 verschiedenen Netzwerkschichttypen berücksichtigt werden können. Das Bit 27 (*RF*) enthält die Information, ob sich ein Korrelations-, Dekorrelations- oder Summen-Neuron in einem RF eines positiven oder negativen Detektorprofils befindet. Die Zuordnung des Neurons zu Gabor-Detektoren, RFs und Gradienten-Paaren erfolgt über Nummern (*DETECTOR*, *NUM_RF* und *NUM_GDP*). Aufgrund der Zuteilung der Datenbits ergibt sich eine maximale Anzahl von 8 Gabor-Detektoren mit jeweils maximal $2^{15} \approx 32 \cdot 10^3$ RFs, die wiederum maximal 512 Gradientenpaare haben können. Die Neuronen der Eingangsschicht verwenden x- und y-Koordinaten und werden wie bei der Kaskaden-Architektur in Abb. 4-20 codiert. Die Typendefinition und die Schichtennummer sind bei diesen Neuronen immer Null.



- STATE : '0' = Neuron pulst nicht (gültig)
  '1' = Neuron pulst (nicht gültig)
- TYPE : '000' = Eingangsschicht-Neuron (N_{INP})
  '001' = Korrelations-Neuron (N_{COR})
  '010' = Dekorrelations-Neuron (N_{DEC})
  '100' = Summen-Neuron (N_{SUM})
  '101' = Summenkorrelations-Neuron (N_{SCOR})
  '111' = Summendekorrelations-Neuron (N_{SDEC})
- RF : '0' = positives RF
  '1' = negatives RF
- DETECTOR : Gabor-Detektor-Nummer D (0...7)
- NUM_RF : RF-Nummer (0...32767)
- NUM_GDP : Gradientenpaar-Nummer (0..511)

Abb. 4-23: Neuronencodierung bei einer Gabor-Architektur

Resümee

Im Rahmen dieses vierten Kapitels wurde das Architekturkonzept der Emulationsplattform SEE vorgestellt, welches auf 3 FPGAs basiert (siehe Abb. 4-6 auf Seite 68). Der

wesentliche Unterschied zu existierenden Architekturkonzepten ist, dass zwischen Netzwerktopologieberechnung und Neuronenzustandsberechnung eine effiziente Kommunikation anhand von Topologie-Vektoren realisiert wird (siehe Abschn. "NTC/NSC-Kommunikation" auf Seite 69). Die Untersuchungen zu den Genauigkeitsanforderungen an die Festkomma-Arithmetik haben ergeben, dass für das Neuronenmodell mit adaptiven Synapsengewichten eine höhere Auflösung der Netzwerkparameter notwendig ist. Andere Emulationsplattformen, wie MASPINN oder SP²INN, berücksichtigen maximal 16 Bit für Dendritenpotentiale, während SEE 20 Bit für Modellparameter und 32 Bit für Zeitparameter vorsieht (siehe Kapitel 4.1.2 auf Seite 64). Es besteht somit der Bedarf einer erhöhten Speicherbandbreite, den die Emulationsplattform durch eine verteilte Speicherarchitektur abdeckt. Dadurch wird der parallele Zugriff über 10 Datenkanäle zu externen Speichern ermöglicht, und für die Aktualisierung der Neuronenzustände wird eine Speicherbandbreite von 1,2 GB/s zur Verfügung gestellt. Auf der Emulationsplattform werden insgesamt 5 Maßnahmen zur Beschleunigung der PCNN Emulation ergriffen: Klassifizierung der Neuronen, Parallelverarbeitung des Pulsereignis-basierten Emulationsablaufs, optimale Datenkanalauslastung des Parameterspeichers, Autarkie der Emulationsplattform und die Parallelisierung des numerischen Integrationsverfahrens (siehe Kapitel 4.2 auf Seite 69). Die arithmetischen Operationen des numerischen Integrationsverfahrens von Bulirsch-Stoer, welches für die Neuronenzustandsberechnung verwendet wird, wurden erläutert und die entsprechenden Hardware-Abbildungen anhand einer Pipeline-Struktur vorgestellt. Schließlich wurden als Netzwerkarchitekturen die Kaskaden- und die Gabor-Architektur diskutiert, die von der Netzwerktopologieberechnung für das Neuronenmodell mit adaptiven Synapsengewichten berücksichtigt werden müssen.

Die Hardware- und Software-Implementierung von SEE wird nun im folgenden Kapitel präsentiert, in dem auch das Zusammenspiel von PPC2-, NTC- und NSC-FPGA während der Emulation erläutert wird.

KAPITEL 5

Implementierung von SEE

In diesem Kapitel wird die Implementierung der Emulationsplattform SEE dargestellt. Zunächst wird der Entwurf der Hardware-Module vorgestellt, die alle mit der Hardwarebeschreibungssprache Verilog beschrieben wurden. Es folgt die detailliertere Darstellung der Hardware-Module im Rahmen der Emulationsablaufkontrolle, der Neuronenzustands- und der Netzwerktopologieberechnung. Dann erfolgt die Vorstellung des Entwurfs der notwendigen Software-Module. Software-Module, die als Steuerprogramme für den Emulationsablauf dienen, wurden mit der Programmiersprache C beschrieben. Software-Module, die als graphische Benutzeroberfläche zur Durchführung einer PCNN Emulation dienen, wurden mit der Skriptsprache Perl/Tk entworfen. Nach der Vorstellung von Hardware- und Software-Modulen, wird der Hardware/Software-Entwurfsfluss (*Designflow*) beschrieben, der für die Implementierung von SEE durchgeführt wurde. Schließlich erfolgt die Vorstellung der Prototypen-Platine, die im Rahmen dieser Arbeit neu entwickelt wurde. Hierbei werden die zu erwartenden Werte für die Verlustleistung angegeben, die für die Bereitstellung ausreichender Versorgungsströme auf der Prototypen-Platine zwingend notwendig sind. Schließlich wird die Durchführung einer Emulation auf der Emulationsplattform SEE beschrieben, welche die Netzwerkbeschreibung, die Netzwerkkonfiguration, die Inbetriebnahme der Prototypen-Platine und die Visualisierung von Pulsereignissen umfasst.

5.1 Beschreibung des Hardware-Entwurfs

Der Entwurf der Hardware-Module erfolgt gesondert für jeden der 3 FPGAs auf der Emulationsplattform. Das PPC2-FPGA beinhaltet die Hardware-Module, welche die beiden eingebetteten PowerPC-Prozessoren (PPC = *PowerPC)* bei der Emulationsablaufkontrolle unterstützen. Die Hardware-Module im NSC-FPGA sind für Hardware-Implementierung des numerischen Integrationsverfahrens nach Bulirsch-Stoer zuständig, welches im Kapitel 4.3 auf Seite 72 erläutert wurde und welches für die Ermittlung von Puls-Start-Ereignissen und für die Aktualisierung der Neuronenzustände eingesetzt wird. Im NTC-FPGA befinden sich die Hardware-Module, die Informationen der Verbindungsstrukturen im PCNN verarbeiten. Durch einen direkten Zugriff auf Markierungsspeicher, die alle Neuronen im Netzwerk kennzeichnen, ob sich diese im Pulszustand oder Erregungszustand befinden, können diese Hardware-Module auf eine effiziente Weise prä- und postsynaptische Verbindungen eines Neurons bestimmen. Schließlich werden noch die Listenzugriffe auf die DEL der Hardware-Module vom NTC- und vom NSC-FPGA beschrieben, die zu der Einführung von insgesamt 8 verschiedenen Datentransfertypen geführt haben, um die jeweiligen Datenkanäle vom NTC- zum PPC2-FPGA und vom NSC- zum PPC2-FPGA effizienter zu nutzen.

5.1.1 Emulationsablaufkontrolle

Neben der Emulationsablaufsteuerung (Konfiguration des Netzwerks, Protokollierung der Emulationsereignissen und Verwaltung der Ereignislisten) stellt die Hardware-Implementierung um die beiden eingebetteten PPCs im PPC2-FPGA eine Systemumgebung dar, die eine flexible Emulationsablaufkontrolle ermöglicht (siehe Abb. 5-1).

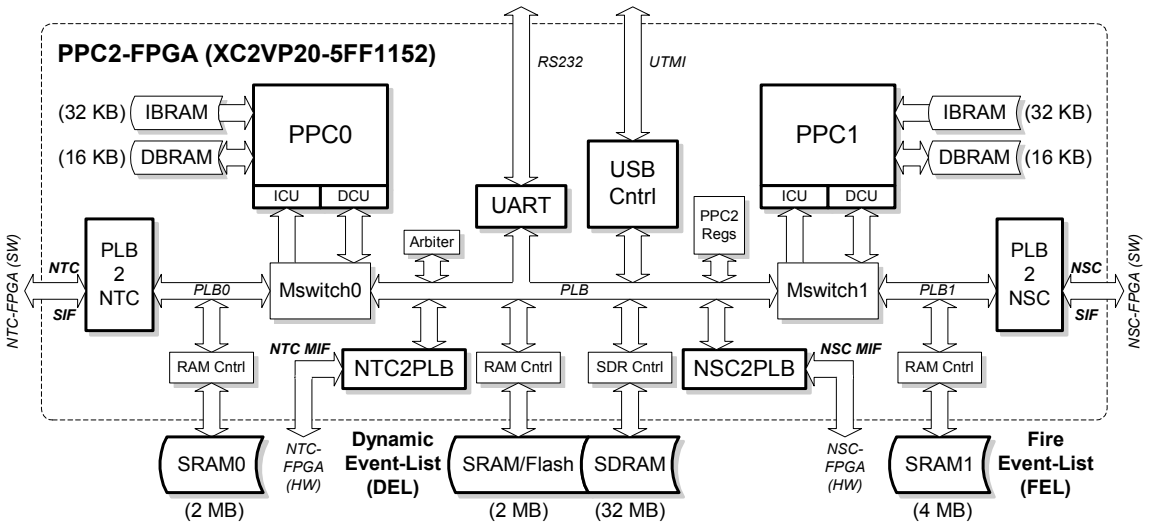


Abb. 5-1: Blockdiagramm des FPGAs für die Emulationsablaufkontrolle (PPC2-FPGA)

HW-Module im PPC2-FPGA

In Abb. 5-1 ist das Blockdiagramm des PPC2-FPGAs dargestellt. Links und rechts oben sind die beiden Prozessoren PPC0 und PPC1 zu erkennen, die aufgrund der Harvard-Architektur über einen Befehls-Cache (IBRAM) und über einen Daten-Cache (DBRAM) verfügen. Die PPCs sind über PLB-Busse (PLB = *Processor Local Bus*) zu den externen Speichern SRAM0, SRAM/Flash, SDRAM und SRAM1 (siehe unten in Abb. 5-1), zu den externen seriellen Schnittstellen über UART und USB Cntrl (siehe oben in Abb. 5-1) und zu den anderen beiden FPGAs über PLB2NTC und PLB2NSC (siehe links und rechts in Abb. 5-1) verbunden. Die Ereignislisten DEL und FEL werden in den externen Speichern (SRAM/Flash bzw. SRAM1) abgelegt. Bei dem PLB-Bus handelt es sich um einen standardisierten On-Chip-Bus, welcher im Rahmen der CoreConnect-Busarchitektur von IBM definiert ist und welcher der Busschnittstelle der PPCs entspricht [Core01]. Die Besonderheit bei diesem voll-synchronen Bus ist, dass er über einen Lese-Datenbus (*Read Data Bus*) und einen Schreib-Datenbus (*Write Data Bus*) verfügt, die zeitgleich genutzt werden können. Dadurch ist es möglich pro Takt 2 Datentransfers durchzuführen (PPC liest Befehl und schreibt zeitgleich ein Datum). Im PPC2-FPGA kommen insgesamt 3 PLB-Busse zum Einsatz. Über 2 Multiplexer-Schaltungen (Mswitch0 und Mswitch1) können die PPCs entweder zu ihrem eigenen PLB-Bus (PLB0 und PLB1 in Abb. 5-1) wechseln oder den gemeinsamen PLB-Bus (PLB in Abb. 5-1) ansteuern, bei dem insgesamt 4 Master arbitriert werden müssen. Neben den beiden PPCs können auf diesem Bus HW-Module des NTC-FPGAs (NTC2PLB) und HW-Module des NSC-FPGAs (NSC2PLB) als Master agieren, um unabhängig von der Software auf die DEL zugreifen zu können. Beide PPCs haben über die PLB-Busse PLB0 und PLB1 jeweils einen eigenen Datenkanal zu den externen Speichern SRAM0 und SRAM1. Somit können umfangreichere Software-Programme, die sich nicht in den Cache-Speichern unterbringen lassen, aus diesen Speichern parallel ausgeführt werden. Das PPC2-FPGA stellt jeweils 2 unabhängige Datenkanäle zum NTC- und zum NSC-FPGA zur Verfügung, die bereits im Kapitel 4.2.3 auf Seite 71 als HW- und SW-Datenkanäle klassifiziert wurden. Über die HW-Module PLB2NTC und PLB2NSC können die Prozessoren per Software auf das NTC- bzw. NSC-FPGA zugreifen (SW-Datenkanäle). Die Hardware im NTC- und NSC-FPGA kann unabhängig von Transfers auf diesen SW-Datenkanälen über die HW-Module NTC2PLB und NSC2PLB auf die externen Speicher zugreifen, die an dem zu arbitrierenden PLB-Bus im PPC2-FPGA angeschlossen sind (HW-Datenkanäle). Da vom PPC2-FPGA zu den anderen beide FPGAs nur jeweils 98 Signalleitungen zur Verfügung gestellt werden konnten, werden Adressen und Daten über einen gemeinsamen 32 Bit-breiten bidirektionalen Bus übertragen.

Emulationsparameter

Die Emulationsablaufkontrolle des PPC2-FPGAs umfasst die Überprüfung von Registerinhalten im NTC- und NSC-FPGA, die den Status von einzelnen HW-Modulen in

diesen FPGAs wiedergeben, und die Überprüfung der Speicherinhalte von Ereignislisten (DEL und FEL), von Markierungsspeichern (ETF und FTF) und vom Parameterspeicher (NIBs). Diese Überprüfungen sind während der Emulation möglich, wenn über die Software allen HW-Modulen mitgeteilt wird, im Einzelschrittbetrieb (*Single Step Mode*) zu operieren. In diesem Betrieb können für jeden Emulationsereigniszeitpunkt die einzelnen Zustände der NTC- und der NSC-Phase ausgeführt werden (NTC: *Topvec-Spike*, *Topvec-Update* und *Update*; NSC: *Read*, *Next-Spike*, *Next-Event* und *Update*), die bereits im Kapitel 4.2.2 auf Seite 70 beschrieben wurden. Die Überprüfung der Richtigkeit von aktualisierten Emulationsparametern ist ebenfalls im Rahmen der Emulationsablaufkontrolle möglich. Diese Emulationsparameter sind in einem BRAM-Speicher im PPC2-FPGA abgelegt, der über den zu arbitrierenden PLB-Bus zugänglich ist (siehe PPC2-Regs in Abb. 5-1), und in Tab. 5-1 zusammengefasst. Die Emulationsparameter im Adressbereich `0x0000` bis `0x005f`, sind dynamisch, da sie während der PCNN Emulation aktualisiert werden. Die Quellen für die Aktualisierung der Parameter sind in der letzten Spalte der Tab. 5-1 angezeigt. Die restlichen Emulationsparameter sind statisch und werden während der PCNN Emulation nicht mehr verändert.

	Adresse	Name	Kommentar	
①	0x0000	SEE_CNTRL	Kontrollinformationen	PPC0 + PPC1
	0x0004	SEE_STATUS	Statusinformationen	
	0x0008	SEE_UART	UART Kommunikation zwischen PPC0 und PPC1	
②	0x0010	DEL_NUM	Anzahl der Neuronen in der DEL	NTC-FPGA + NSC-FPGA
	0x0014	DEL_HIDDEN	Anzahl der erregten Neuronen, die keinen externen Reiz empfangen	
	0x0018	DEL_WP	Schreibzeiger (Write-Pointer) der DEL	
	0x001C	DEL_STOP	Anzahl der Neuronen, die aufgrund eines endenden Pulsereignisses der DEL hinzugefügt wurden	
③	0x0020	DEL_VALID	Zeiger auf das erste Neuron, das aufgrund eines Pulsereignisses der DEL hinzugefügt wurde	NTC-FPGA
	0x0024	DEL_INVALID	Zeiger auf das erste unzulässige Neuron in der DEL	
	0x0028	DEL_START	Anzahl der Neuronen, die aufgrund eines beginnenden Pulsereignisses der DEL hinzugefügt wurden	
④	0x0030	SEE_NUM_START	Anzahl der Neuronen, die anfangen zu pulsen	NSC-FPGA
	0x0034	SEE_NUM_STOP	Anzahl der Neuronen, die aufhören zu pulsen	
	0x0038	SEE_NUM_EVENT	Anzahl der Neuronen, die ein Emulationsereignis im aktuellen Emulationsdurchlauf generieren	
	0x003C	SEE_DT	Aktueller Zeitschritt der Emulation	
⑤	0x0040	SEE_NUM_DT	Anzahl der Emulationsdurchläufe	
	0x0044	SEE_TIME	Aktuelle Emulationszeit	
	0x0048	SEE_EVENT	Anzahl der Emulationsereignisse	
	0x004C	SEE_WP	Schreibzeiger (Write-Pointer) zur Protokollierung der Emulationsereignisse	
⑥	0x0050	FEL_RP	Lesezeiger (Read-Pointer) der FEL	PPC1
	0x0054	FEL_WP	Schreibzeiger (Write-Pointer) der FEL	
	0x0058	FEL_NUM	Anzahl der Neuronen in der FEL	
⑦	0x0060	SEE_BASE_DEL	Basisadresse der DEL	
	0x0064	SEE_BASE_FEL	Basisadresse der FEL	
	0x0068	SEE_BASE_SIM	Basisadresse der Emulationsereignisse	

Tab. 5-1: Emulationsparameter im PPC2-FPGA

	Adresse	Name	Kommentar
⑧	0x0070	SEE_END	Maximale Emulationszeit
	0x0074	SEE_START	Emulationsstartzeit
	0x0078	SEE_TP_MP	Festkomma-Darstellung von Zeit- und Modellparametern
	0x007C	SEE_ARCH	Netzwerkarchitektur: CSC/GBR
	0x0080	SEE_NOL_NOD	Gesamtanzahl der Lagen/Detektoren
	0x0084	SEE_NON	Gesamtanzahl der Neuronen
	0x0088	SEE_NOIN	Gesamtanzahl der Eingangsschichtneuronen
	0x008C	SEE_NOW	Gesamtanzahl der Synapsen
⑨	0x0090	SEE_LD_XY_0	XY-Darstellung für die Eingangsschicht
	0x0094	SEE_LD_BNUM_0	Basis der Neuronennummer für die Eingangsschicht
	0x00D0	SEE_LD_XY_8	XY-Darstellung für Schicht 8/Detektor 8
	0x00D4	SEE_LD_BNUM_8	Basis der Neuronennummer für Schicht 8/Detektor 8
	0x0188	SEE_LD_XY_31	XY-Darstellung für Schicht 31
	0x018C	SEE_LD_BNUM_31	Basis der Neuronennummer für Schicht 31

Tab. 5-1: Emulationsparameter im PPC2-FPGA

Der erste Parametersatz ① ist nur für die Software bestimmt, um Kontrollinformationen zu hinterlegen oder Statusinformationen abzufragen, ob gültige Konfigurationen für die Durchführung einer PCNN Emulation vorliegen. Das Register SEE_UART dient zur wechselseitigen Ansteuerung der seriellen Schnittstelle, da ansonsten bei einer zeitgleichen Ansteuerung der beiden PPCs keine fehlerfreie Kommunikation zur seriellen Schnittstelle gewährleistet werden kann. Der Parametersatz ② wird sowohl vom NTC- als auch vom NSC-FPGA modifiziert. Nachdem Neuronen der DEL hinzugefügt wurden, wird das Register DEL_NUM und der Schreibzeiger DEL_WP entsprechend erhöht. Im *Update* Zustand der NTC-Phase wird der Parametersatz ③ vom NTC-FPGA aktualisiert. Die Anzahl der ermittelten postsynaptischen Neuronen wird dann im Register DEL_START hinterlegt. Der Parametersatz ④ wird im *Next-Event* Zustand der NSC-Phase vom NSC-FPGA modifiziert. Im Register SEE_NUM_EVENT wird hierbei die Anzahl der ermittelten Puls-Start- und Puls-Stop-Ereignisse abgespeichert, und im Register SEE_DT wird der aktuell berechnete Zeitschritt hinterlegt. Der Parametersatz ⑤ wird im *Update* Zustand der NSC-Phase ebenfalls vom NSC-FPGA modifiziert. Dabei wird die Emulationszeit um den aktuellen Zeitschritt erhöht (SEE_EVENT) und das Register SEE_NUM_DT, welches die Anzahl der Emulationsdurchläufe anzeigt, um Eins erhöht. Die Parametersätze ⑦, ⑧ und ⑨ werden während der PCNN Emulation nicht verändert. Hier werden die Basisadressen der Ereignislisten, die Basisadresse für die Abspeicherung von Emulationsereignissen und Angaben zur Netzwerkbeschreibung, wie die maximale Emulationszeit (SEE_END), die Gesamtanzahl der Neuronen (SEE_NON) und der Synapsen (SEE_NOW) abgelegt. Schließlich wird noch die Gesamtanzahl der Neuronen einer Netzwerkschicht bzw. eines Gabor-Detektors (SEE_LD_BNUM) definiert, die für die Durchnummerierung der Neuronen im Netzwerk und für die Konvertierung von einer Koordinatendarstellung der Neuronen zu einer Neuronennummer notwendig ist (siehe auch Anhang F auf Seite 171).

5.1.1.1 Verwaltung der Listen pulscodierter Neuronen

Während der Prozessor PPC0 für die Emulationsablaufkontrolle zuständig ist, wird während der Emulation die FEL vom PPC1 verwaltet. Dafür wird im *Read* Zustand in der NSC-Phase, wenn der Emulationsparameter FEL_NUM (0x0058) nicht Null ist, die Zeit des nächsten Puls-Stop-Ereignisses aus der FEL gelesen und dem NSC-FPGA übergeben. Wenn das NSC-FPGA im *Next-Spike* Zustand Puls-Start-Ereignisse ermittelt hat, die vor dem Puls-Stop-Ereignis auftreten, werden diese vom PPC1 aus dem NSC-FPGA

(über den SW-Datenkanal) in die FEL übertragen und der Emulationsparameter FEL_WP erhöht. Bei einem früheren Eintritt des Puls-Stop-Ereignisses wird der Emulationsparameter FEL_RP erhöht (siehe Parametersatz ⑥ in Tab. 5-1).

DEL/FEL-Codierung

Die Neuronen werden, abhängig von der Netzwerkarchitektur, nach den Codierungen in Abb. 4-20 auf Seite 86 (Kaskaden-Architektur) bzw. in Abb. 4-23 auf Seite 88 (Gabor-Architektur) in der DEL als 32 Bit Datum abgelegt. Dabei signalisiert das oberste Bit (Bit 31 = STATE) der Neuronencodierung, ob das Neuron gültig und somit für den aktuellen Emulationsdurchlauf berücksichtigt werden muss oder ob es ungültig und somit nicht berücksichtigt werden muss. Die Neuronencodierung in der FEL entspricht der in der DEL mit dem Unterschied, dass zu jedem Neuron ein Zeitwert (ebenfalls als 32 Bit Datum) abgelegt wird, an dem das Neuron den Pulszustand wieder verlässt.

5.1.1.2 Adressraum der Emulationsplattform

Für den Adressraum der Emulationsplattform wird der bei 32 Bit-breiten Adressen maximal physikalisch adressierbare Speicherbereich von `0x00000000` bis `0xffffffff` ausgenutzt. Am Anfang dieses Speicherbereichs sind die Daten des Daten-Caches (PPC0/PPC1-DBRAM) abgelegt und am Ende befinden sich die Befehle, welche im Befehls-Cache (PPC0/PPC1-IBRAM) abgespeichert sind (siehe Tab. 5-2).

	Adresse	Speicher	Größe	Adressende	Zugriff
①	`0x00000000`	PPC0/PPC1-DBRAM	16 KB	`0x00003fff`	PPC2-FPGA
	`0x01000000`	PPC0-SRAM0 PPC1-SRAM1	2 MB 4 MB	`0x011fffff` `0x013fffff`	
②	`0x10000000`	PPC2-SRAM	2 MB	`0x101fffff`	PPC2/NSC/NTC-FPGA
	`0x11000000`	PPC2-REGS	1 KB	`0x110003ff`	
	`0x12000000`	PPC2-SDRAM	32 MB	`0x13ffffff`	
	`0x1a000000`	PPC2-UART	100 B	`0x1a000063`	
	`0x1a100000`	PPC2-USB	100 B	`0x1a100063`	
	`0x1f000000`	PPC2-Flash	4 MB	`0x1f3fffff`	
③	`0x20000000`	NSC-SDRAM0	1 GB	`0x5fffffff`	NSC-FPGA
	`0x60000000`	NSC-SDRAM1	1 GB	`0x9fffffff`	
	`0xa0000000`	NSC-SDRAM2	1 GB	`0xdfffffff`	
④	`0xe0000000`	NSC/NTC-REGS	1 KB	`0xe000ffff`	NSC/NTC-FPGA
⑤	`0xe0010000`	NSC-PE0	1 KB	`0xe001ffff`	NSC-FPGA
	`0xe0020000`	NSC-PE1	1 KB	`0xe002ffff`	
	`0xe0030000`	NSC-PE2	1 KB	`0xe003ffff`	
	`0xe0040000`	NSC-TVU	1 KB	`0xe004ffff`	
⑥	`0xe0100000`	NTC-SRAM-ETF	256 KB	`0xe013ffff`	NTC-FPGA
	`0xe0140000`	NTC-SRAM-FTF	256 KB	`0xe017ffff`	
	`0xe0180000`	NTC-BRAM-TPC	64 KB	`0xe018ffff`	
	`0xe0200000`	NTC-SDRAM-TPM	8 MB	`0xe09fffff`	
⑦	`0xffff8000`	PPC0/PPC1-IBRAM	32 KB	`0xffffffff`	PPC2-FPGA

Tab. 5-2: Adressraum der digitalen Emulationsplattform

In der letzten Spalte der Tab. 5-2 ist zusätzlich angegeben, welche FPGAs auf die jeweiligen Speicher im Adressraum Zugriff haben. Die Adressbereiche ① und ⑦ sind ausschließlich vom PPC2-FPGA aus erreichbar. Der Adressbereich ② umfasst alle HW-Module, die an dem zentralen PLB-Bus im PPC2-FPGA angeschlossen sind (siehe Abb. 5-1 auf Seite 92) und ist somit von allen 3 FPGAs zugänglich. Die Adressbereiche ③

und ⑤ werden nur vom NSC-FPGA verwendet und beinhalten die SDRAM-Speicher, in denen die NIBs der Neuronen abgelegt sind, und die Register der PEs und der TVU (*Topology Vector Unit*). Der Adressbereich ⑥ wird allein vom NTC-FPGA benutzt und umfasst sowohl die Markierungsspeicher (ETF und FTF) als auch die Topologiespeicher (TPC und TPM). Im Adressbereich ④ befinden sich die Kontrollregister von NTC- und NSC-FPGA, die per Software über die Prozessoren PPC0 bzw. PPC1 adressierbar sind.

5.1.2 Neuronenzustandsberechnung

Die Neuronenzustandsberechnung umfasst die Durchführung der numerischen Integration, die in den Zuständen *Next-Spike* (Suche nach Puls-Start-Ereignissen) und *Update* (Aktualisierung der Neuronenzustände) in der NSC-Phase vom NSC-FPGA durchgeführt wird (siehe auch Kapitel 4.3.4 auf Seite 83). Das Blockdiagramm mit den HW-Modulen, die sich im NSC-FPGA befinden, ist in Abb. 5-2 dargestellt.

HW-Module im NSC-FPGA

Während der numerischen Integration werden die erregten Neuronen über den HW-Datenkanal von der zentralen Kontrolleinheit (NSC-Cntrl) über das HW-Modul NSC-MIF (MIF = *Master Interface*, rechts oben in Abb. 5-2), welches mit dem HW-Modul NSC2PLB im PPC2-FPGA (siehe auch Abb. 5-1 auf Seite 92) verbunden ist, ins FPGA geladen. Die Kontrolleinheit NSC-Cntrl aktiviert in Zusammenarbeit mit der Topologie-Vektor-Einheit (TVU) die entsprechende PE, die Zugriff auf den NIB des erregten Neurons hat und in dem Parameter, wie Membranpotential und präsynaptische Synapsengewichte, abgelegt sind. Die PEs erhalten von der TVU die Topologie-Vektoren, die vom NTC-FPGA berechnet werden und die präsynaptische Pulsaktivität beinhalten. Die TVU erhält die Topologie-Vektoren über das HW-Modul NTC-SIF (SIF = *Slave Interface*, links unten in Abb. 5-2) vom NTC-FPGA. Wenn Neuron und zugehöriger Topolgie-Vektor verfügbar sind, wird einer der inaktiven HW-Integratoren BSI (*Bulirsch-Stoer Integrator*) aktiviert, um auf den NIB des Neurons zu zugreifen und die numerische Integration durchzuführen. Die notwendigen SDRAM-Kontrollmodule, welche die SDRAM-DIMM-Speicher ansteuern, wurden mit dem Werkzeug CoreConsultant von Synopsys als Hardware-Makros generiert (SDRAM-Cntrl in Abb. 5-2). Diese SDRAM-Kontrollmodule verwenden als interne Busschnittstelle den standardisierten On-Chip-Bus AHB (*Advanced High-Performance Bus*), der im Rahmen der AMBA-Busarchitektur (AMBA = *Advanced Microcontroller Bus Architecture*) von ARM definiert ist [AMBA99].

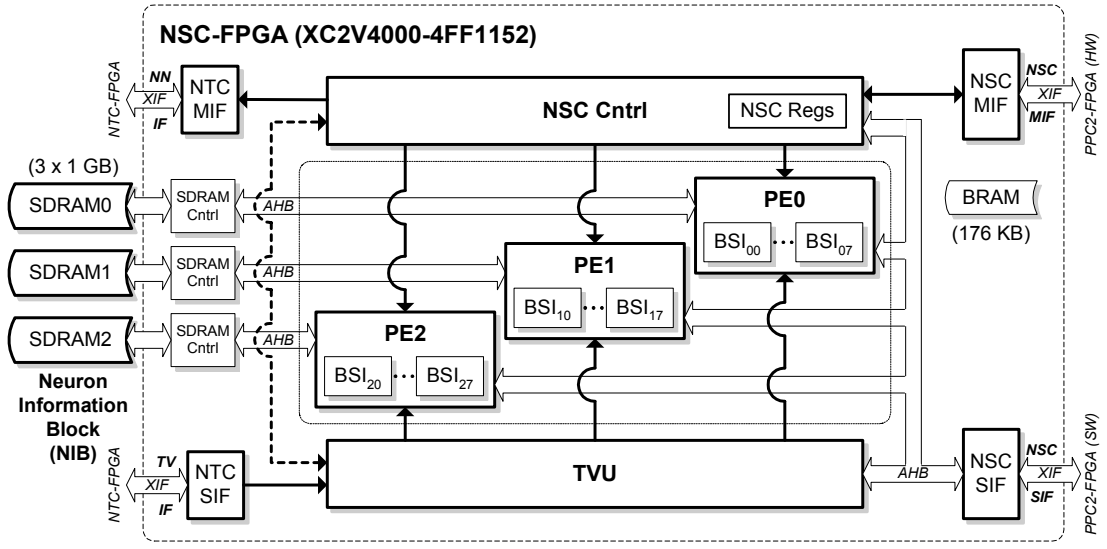


Abb. 5-2: Blockdiagramm des FPGAs für die Neuronenzustandsberechnung (NSC-FPGA)

Im Zustand *Next-Event* der NSC-Phase werden Puls-Start-Ereignisse über das HW-Modul NTC-MIF (links oben in Abb. 5-2) an das NTC-FPGA gesendet während der PPC1 über das HW-Modul NSC-SIF (rechts unten in Abb. 5-2) die Neuronen, die ein Puls-Start-

Ereignis generieren, in die FEL überträgt. Zusätzlich wird dabei zu jedem Neuron in der FEL ein Zeitwert abgelegt, an dem das Neuron den Pulszustand wieder verlässt. Dieser Zeitwert ergibt sich aus der Summe von aktueller Emulationszeit und der Pulsdauer t_d des Neurons.

Die Codierung eines NIBs ist in Abb. 5-3 dargestellt. **NIB-Codierung**

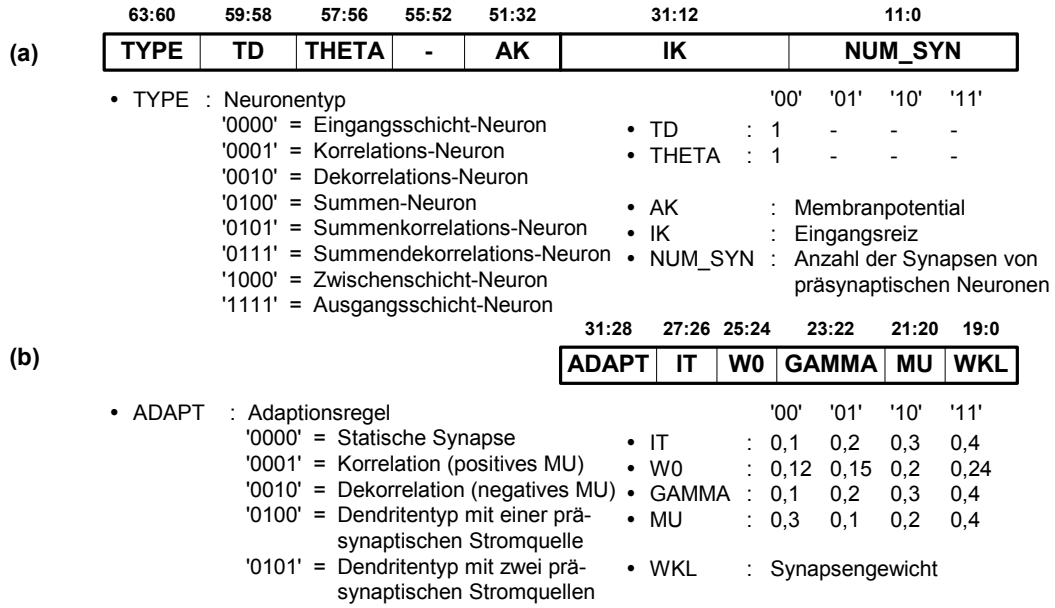


Abb. 5-3: Neuroneninformationsblock (64 Bit für jedes Neuron, 32 Bit für jedes Synapsengewicht)

Für jedes Neuron existiert ein 64 Bit-breite Header-Information, die neben dem Membranpotentialwert (AK = 20 Bit), die Anzahl der Synapsen von präsynaptischen Neuronen (NUM_SYN = 12 Bit) und bei Eingangsschicht-Neuronen den Wert des externen Eingangsreizes (IK = 20 Bit) enthält. Außerdem werden für jedes Neuron 2-Bit-Codierungen für Pulsdauer (IK) und Schwellwert (THETA) abgelegt. Die Synapsengewichtswerte werden mit 20 Bit codiert (WKL) und weitere notwendige Synapsenparameter, wie Schwellstrom (IT), statischer Gewichtswert (W0), Abklingkonstante (GAMMA) und Verstärkungsfaktor (MU), werden in 12 Bit codiert, um die Werte dieser Parameter auf dem NSC-FPGA per Multiplexer auszuwählen.

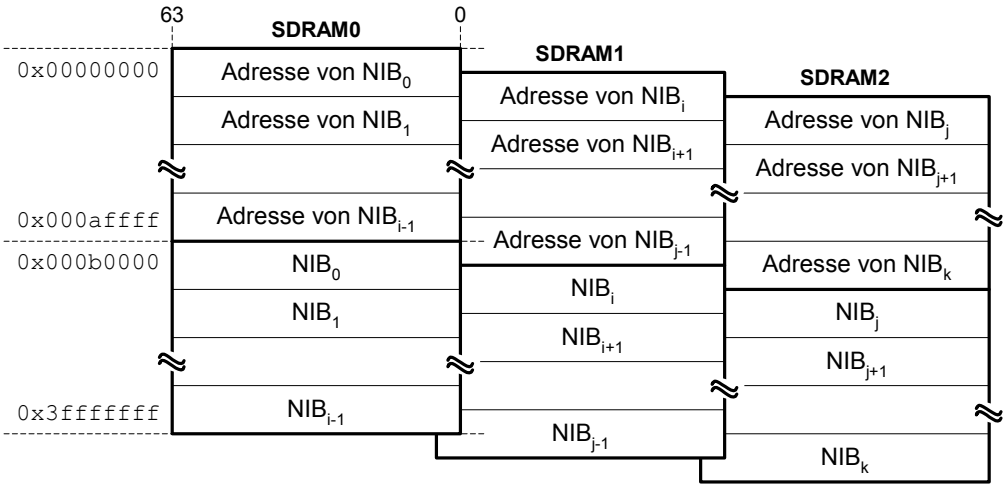


Abb. 5-4: NIB-Speichernutzung

Die Speichernutzung der 3 SDRAM-DIMM-Speicher, in denen die NIBs der Neuronen abgelegt sind, ist in Abb. 5-4 dargestellt. In den oberen Speicherbereichen der SDRAMs

sind die Sprungadressen zu den NIBs der Neuronen hinterlegt, deren Zugriff sich aus der Neuronennummer ergibt.

NIB-Zugriff bei Kaskaden-Architektur

Wenn das NSC-FPGA ein erregtes Neuron aus der DEL liest, wird anhand der Neuronencodierung (nach Abb. 4-20 auf Seite 86) die Neuronennummer NUM_{NEURON} des Neurons ermittelt (siehe auch Anhang F auf Seite 171). Die Neuronennummer stellt die Adresse dar, an der die Sprungadresse zum NIB des Neurons hinterlegt ist. Bei der in Abb. 4-19 auf Seite 85 dargestellten Kaskaden-Architektur erhält man beispielsweise für das Neuron ($BNUM_0 = 0$, $x_{MAX,0} = y_{MAX,0} = 10$) mit der Codierung `0x0000a001` ($\Rightarrow$ L = 0, x = 10, y = 1) nach der Gl. A-8 im Anhang auf Seite 171 die Neuronennummer $NUM_{NEURON} = 0+10\cdot(10-1)+(10-1) = 99$. Dem Neuron ($BNUM_2 = 300$, $x_{MAX,2} = y_{MAX,2} = 10$) mit der Codierung `0x6200100a` ($\Rightarrow$ L = 2, x = 1, y = 10) wird die Neuronennummer $NUM_{NEURON} = 300+1\cdot(10-10)+(1-1) = 300$ zugewiesen.

NIB-Zugriff bei Gabor-Architektur

Bei einer Gabor-Architektur, wie in Abb. 4-21 auf Seite 87 dargestellt, gilt für Eingangsschicht-Neuronen die Neuronencodierung einer Kaskaden-Architektur. Für die übrigen Neuronentypen gilt die Neuronencodierung nach Abb. 4-23 auf Seite 88. Es ergeben sich folgende Neuronennummern für eine Gabor-Architektur mit den Parametern $x_{MAX} = y_{MAX} = 10$ und $x_{RF} \cdot y_{RF} = x_{GDP} \cdot y_{GDP} = 16$:

- `0x00001001` ($\Rightarrow$ Eingangsschicht-Neuron N_{INP}, x = 1, y = 1)
  $NUM_{NEURON,INP} = 10\cdot(10-1)+(1-1) = 90$ (siehe Gl. A-10 im Anhang auf Seite 171)
- `0x10000000` ($\Rightarrow$ Korrelations-Neuron N_{COR}, RF = 0, $NUM_{RF} = NUM_{GDP} = 0$)
  $NUM_{NEURON,COR} = 100+0\cdot(16\cdot16)+0\cdot16+0 = 100$ (siehe Gl. A-11 und Gl. A-14)
  `0x18000000` ($\Rightarrow$ Korrelations-Neuron N_{COR}, RF = 1, $NUM_{RF} = NUM_{GDP} = 0$)
  $NUM_{NEURON,COR} = 100+1\cdot(16\cdot16)+0\cdot16+0 = 356$
- `0x20001e0f` ($\Rightarrow$ Dekorrelations-Neuron N_{DEC}, RF = 0, $NUM_{RF} = NUM_{GDP} = 15$)
  $NUM_{NEURON,COR} = 612+0\cdot(16\cdot16)+15\cdot16+15 = 867$ (siehe Gl. A-11 und Gl. A-15)
  `0x28000000` ($\Rightarrow$ Dekorrelations-Neuron N_{DEC}, RF = 1, $NUM_{RF} = NUM_{GDP} = 0$)
  $NUM_{NEURON,COR} = 612+1\cdot(16\cdot16)+0\cdot16+0 = 868$
- `0x40001e00` ($\Rightarrow$ Summen-Neuron N_{SUM}, RF = 0, $NUM_{RF} = 15$)
  $NUM_{NEURON,SUM} = 1124+0\cdot16+15 = 1139$ (siehe Gl. A-12 und Gl. A-15)
  `0x48000000` ($\Rightarrow$ Summen-Neuron N_{SUM}, RF = 1, $NUM_{RF} = 0$)
  $NUM_{NEURON,SUM} = 1124+1\cdot16+0 = 1140$
- `0x50001e00` ($\Rightarrow$ Summenkorrelations-Neuron N_{SCOR}, $NUM_{RF} = 15$)
  $NUM_{NEURON,SCOR} = 1156+15 = 1171$ (siehe Gl. A-13 und Gl. A-15)
- `0x70001e00` ($\Rightarrow$ Summendekorrelations-Neuron N_{SDEC}, $NUM_{RF} = 15$)
  $NUM_{NEURON,SDEC} = 1172+15 = 1187$ (siehe Gl. A-13 und Gl. A-15)

Schließlich werden auch die Topologie-Vektoren mit zugehöriger Neuronennummer vom NTC-FPGA an das NSC-FPGA übermittelt, damit die präsynaptische Pulsaktivität vom NSC-FPGA dem jeweiligen Neuron zugeordnet werden kann.

5.1.2.1 Bulirsch-Stoer Hardware Integrator

In einem Prozessor-Element des NSC-FPGAs können bis zu 8 HW-Integratoren zum Einsatz kommen (siehe auch BSI_{x0} ... BSI_{x7} in Abb. 5-2 auf Seite 96). In einem HW-Integrator werden die HW-Module MMID und PZEXTR verwendet, die im Kapitel 4.3 auf Seite 72 vorgestellt wurden. Ein HW-Integrator besteht aus den folgenden 4 HW-Modulen: Speicherschnittstelle (BSI-IF), Synapsengewicht-Berechnungseinheit (W_{KL}), Membranpotential-Berechnungseinheit (a_K) und eine Steuereinheit (BSI-Cntrl). Das Blockdiagramm des HW-Integrators ist in Abb. 5-5 dargestellt.

Das HW-Modul BSI-IF dient als Schnittstelle zu den SDRAM-Speichern. Je nach Vernetzungsdichte (siehe hierzu auch das nächste Kapitel 5.1.2.2 auf Seite 100) werden vom BSI-IF 3 SDRAM-Speicher gleichzeitig oder nur ein einziger SDRAM-Speicher angesteuert. Das BSI-IF versorgt über 2 Multiplexer (MUX0 und MUX1) die HW-Module zur Berechnung der Synapsengewichte und des Membranpotentials (a_K und W_{KL}). Die vom BSI-IF

gelesenen Daten werden somit über einen zwei-kanaligen Lesepfad an die HW-Module (*Read Data Path*) gesendet. Die Ergebnisse von MMID-Integration ($W_{KL,i+1}$) und Polynomextrapolation ($W_{KLj,n}$ und $Q_{j,k}$ bzw. a_K) werden dann vom BSI-IF über einen Multiplexer (MUX2) eingesammelt und entweder wieder in den SDRAM-Speicher zurückgeschrieben oder intern in BRAM-Blöcken zwischengespeichert (*Write Data Path*).

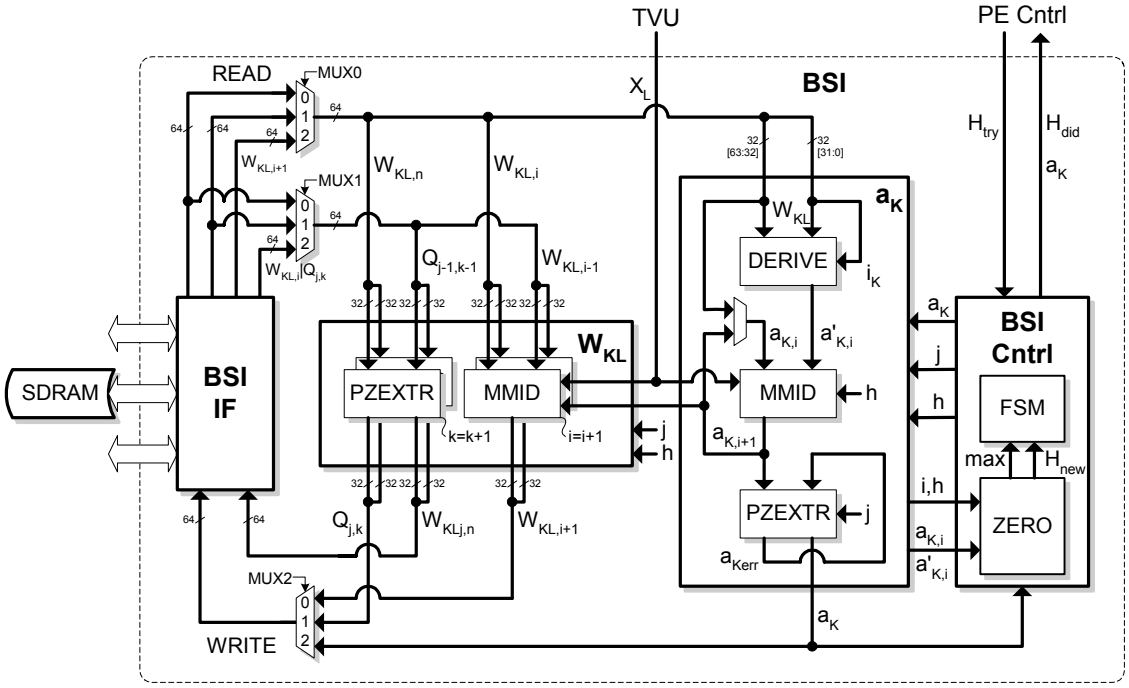


Abb. 5-5: Blockdiagramm des Bulirsch-Stoer HW Integrators (BSI)

Zu Beginn eines Integrationsdurchlaufs wird dem HW-Integrator ein Integrationsintervall H_{try} von der Prozessorelement-Steuerung vorgegeben (siehe Abb. 5-5 oben rechts).

Die numerische Integration beginnt dann mit dem Lesen des Neuroneninformationsblocks bzw. NIBs eines Neurons aus einem SDRAM-Speicher (siehe auch Abschn. "NIB-Zugriff" auf Seite 98). Die ersten 64 Bits eines NIBs enthalten in den oberen 32 Bits das Membranpotential a_K und in den unteren 32 Bits gegebenenfalls einen Eingangsreiz i_K (siehe auch Abschn. "NIB-Codierung" auf Seite 97), die über den Multiplexer (MUX0 = 0) an die Membranpotential-Berechnungseinheit weitergereicht werden. Die nachfolgenden Synapsengewichte, die im NIB hinterlegt sind, werden dann zeitgleich an die Synapsengewicht-Berechnungseinheit über den Lesepfad $W_{KL,i}$ für die MMID-Integration (siehe Kapitel 4.3.1 auf Seite 74) und an die Membranpotential-Berechnungseinheit für die Bestimmung der Ableitung des Membranpotentials (siehe HW-Modul DERIVE in Abb. 5-5 und Gl. 4-7 auf Seite 75) weitergeleitet. Die ermittelten Funktionszwischenwerte der MMID-Integration werden dann anhand des Schreibpfads $W_{KL,i+1}$ über den Multiplexer (MUX2 = 0) an das HW-Modul BSI-IF zurückgeschrieben. Die für die Ableitungsberechnung der Synapsengewichte notwendige präsynaptische Pulsaktivität X_L (siehe auch Gl. 2-15 auf Seite 20) wird von der TVU (siehe auch Abb. 5-2 auf Seite 96) in Form eines Topologie-Vektors für die Dauer der numerischen Integration bereitgestellt. Für die nachfolgenden Berechnungsschritte der MMID-Integration werden nun immer 2 Funktionszwischenwerte gelesen (siehe auch Gl. 4-5 auf Seite 75). Dabei wird der obere Lesepfad $W_{KL,i}$ in Abb. 5-5 (MUX0 = 2), der die vorher zurückgeschriebenen Funktionswerte enthält, und der untere Lesepfad $W_{KL,i-1}$ (MUX1 = 2), der die vorher über den Lesepfad $W_{KL,i}$ gelesenen Funktionswerte enthält, aktiviert. Nachdem der Laufindex i der MMID-Integration den Wert n erreicht hat (siehe auch Gl. 4-9 auf Seite 78), ist die MMID-Integration abgeschlossen, und es folgt die Polynomextrapolation PZEXTR nach Kapitel 4.3.2 auf Seite 78.

Bei der Polynomextrapolation werden über den oberen Lesepfad $W_{KL,n}$ die durch die MMID-Integration ermittelten Synapsengewichte und über den unteren Lesepfad $Q_{j-1,k-1}$ die für die Polynomextrapolation notwendigen Zwischengrößen (siehe auch Gl. 4-16 auf Seite 79) an die Synapsengewicht-Berechnungseinheit übertragen. Die Anzahl der Zwischengrößen bei der Polynomextrapolation entspricht der Variablen *j*, die vom HW-Modul BSI-Cntrl entsprechend gesetzt wird. Die Variable j beginnt bei Null und wird bei jeder Iteration der numerischen Integration (MMID und PZEXTR) um Eins erhöht. Wenn der Laufindex *k* der Polynomextrapolation den Wert *j* erreicht hat, ist die Polynomextrapolation abgeschlossen (siehe auch Gl. 4-17 auf Seite 79). Es folgt die Überprüfung, ob das Membranpotential den Schwellwert überschritten hat anhand des HW-Moduls ZERO in der Steuereinheit. Ist dies der Fall, wird ein neues Integrationsintervall H_{new} bestimmt, mit dem die numerische Integration wiederholt wird. Wenn das Membranpotential den Schwellwert nicht erreicht hat, wird das aktuelle Membranpotential des Neurons und das für die weitere Integration zu verwendende Integrationsintervall H_{did} an die Prozessorelement-Steuerung übertragen.

Bei spärlicher Vernetzung, wie bei der Vierer- oder Achter-Nachbarschaft, haben Neuronen nur eine geringe Anzahl an präsynaptischen Verbindungen bzw. Synapsengewichten. In diesem Fall können alle Funktionszwischenwerte, die bei der numerischen Integration anfallen, im BSI-IF in internen BRAM-Speichern zwischengespeichert werden. Bei dichten Vernetzungsstrukturen, wie der Vollvernetzung, ist die Zwischenspeicherung im FPGA nicht mehr möglich, und die Zwischenspeicherung muss auf den externen SDRAM-Speichern erfolgen. Aus diesem Grund verfügt das BSI-IF über 3 separate Datenkanäle (siehe Abb. 5-5 links) zu jeweils einem der insgesamt 3 SDRAM-Speicher, über die die Emulationsplattform SEE verfügt. Somit können 2 Lesekanäle über die Multiplexer mit MUX0 = 0 und MUX1 = 1 bzw. MUX0 = 1 und MUX1 = 0 aktiviert werden, die Funktionszwischenwerte aus jeweils einem SDRAM-Speicher liefern. Die zurückzuschreibenden Werte $W_{KL,i+1}$ bei der MMID-Integration (MUX2 = 0) bzw. $Q_{j,k}$ bei der Polynomextrapolation (MUX2 = 1) können parallel über den dritten noch zur Verfügung stehenden Datenkanal in den dritten SDRAM-Speicher abgelegt werden. Bei einer Vollvernetzung beispielsweise wird in jedem SDRAM-Speicher ein Speicherbereich vorgesehen, der für die Zwischenspeicherung reserviert ist. Bei der Polynomextrapolation besteht allerdings die Problematik, dass ab dem Laufindex $j = 1$ neben den Zwischengrößen $Q_{j,k}$ auch die extrapolierten Synapsengewichte $W_{KLj,n}$ zeitgleich zwischengespeichert werden müssen ($j = 1$: $Q_{1,0}$, $Q_{1,1}$ und $W_{KL1,n}$; $j = 2$: $Q_{2,0}$, $Q_{2,1}$, $Q_{2,2}$ und $W_{KL2,n}$; ... usw.; siehe auch Abschn. "Zahlenbeispiel" auf Seite 79). Die Zwischenspeicherung der extrapolierten Synapsengewichte erfolgt in dieser Situation über denjenigen Lesekanal, über den die Synapsengewichte nach der MMID-Integration ($W_{KL,n}$) dem HW-Modul PZEXTR in der Synapsengewicht-Berechnungseinheit bereitgestellt werden. Dieser Lesekanal ist im Vergleich zum anderen Lesekanal, der die Zwischengrößen $Q_{j-1,k-1}$ liefert, nicht voll ausgelastet. Bei $j = 1$ müssen zum Synapsengewicht $W_{KL1,n}$ die Zwischengrößen $Q_{1,0}$ und $Q_{1,1}$ gelesen werden, und bei $j = 2$ müssen zu $W_{KL2,n}$ die Zwischengrößen $Q_{2,0}$, $Q_{2,1}$ und $Q_{2,2}$ bereitgestellt werden. Somit ergeben sich beim Lesekanal, der das HW-Modul PZEXTR mit Synapsengewichten versorgt, Zeitschlitze, in denen die extrapolierten Synapsengewichte in den gleichen SDRAM-Speicher abgelegt werden können.

5.1.2.2 Prozessor-Elemente für unterschiedliche Vernetzungsdichten

Die Prozessor-Elemente im NSC-FPGA haben alle einen eigenständigen Datentransferkanal, um auf die NIBs jedes Neurons zuzugreifen. Hauptaufgabe der PEs besteht darin, den Datentransferkanal zum entsprechenden NSC-SDRAM immer effektiv auszulasten. Dies geschieht, indem nacheinander den integrierten BSI-Modulen der Zugriff auf das NSC-SDRAM gewährt wird. Bei einer dichten Vernetzungsstruktur, wird nur eine einzige PE bereitgestellt, die alle 3 Datenkanäle ansteuert (siehe innere Umrandung in Abb. 5-2 auf Seite 96), um zeitgleich 2 Daten lesen und ein Datum schreiben zu können.

Dies ist für eine effiziente Datenkanalauslastung bei der MMID-Integration notwendig, da somit die Fließbandverarbeitung der BSI-Module nicht unterbrochen werden muss.

Spärliche Vernetzung

Bei einer spärlichen Vernetzungsstruktur, wie bei einer Vierer- oder Achter-Nachbarschaft, muss darauf geachtet werden, dass alle 3 PEs möglichst gleichmäßig ausgelastet sind. Der Ansatz für eine gleichmäßige Lastverteilung (siehe hierzu auch Kapitel 3.2.3 auf Seite 37), der bei der Emulationsplattform SEE verfolgt wird, ist, die Gesamtanzahl der Neuronen gleichmäßig auf alle 3 Prozessor-Elemente zu verteilen. Das bedeutet, dass die NIBs nicht nur in einem SDRAM-Speicher abgelegt werden, sondern gleichmäßig in den 3 Speichern SDRAM0, SDRAM1 und SDRAM2 in Abb. 5-2 auf Seite 96 verteilt werden. Hierfür werden die unteren 5 Bits der Neuronennummern ausgewertet. Bei dieser Auswertung werden die Neuronen, deren unteren 2 Bits der Neuronennummer `00`, `01` bzw. `10` entsprechen, den Prozessor-Elementen PE0, PE1 bzw. PE2 zugeordnet. Die Neuronen, deren unteren 2 Bits der Neuronennummer `11` sind, werden auf die 3 PEs verteilt. Das Neuron mit der Neuronennummer 3 wird der PE0 zugeordnet, das Neuron mit der Neuronennummer 7 wird der PE1 zugeordnet usw. (siehe Tab. 5-3 links).

Neuronennummer		PE	PE0	PE1	PE2
3	`00011`	⇒ PE0	0	1	2
7	`00111`	⇒ PE1	3,4	5	6
11	`01011`	⇒ PE2	8	7,9	10
15	`01111`	⇒ PE0	12	13	11,14
19	`10011`	⇒ PE1	15,16	17	18
23	`10111`	⇒ PE2	20	19,21	22
27	`11011`	⇒ PE0	24	25	23,26
31	`11111`	⇒ PE1	27,28	29	30
			32	31,33	34
			35,36	37	38
			40	39,41	42
			...	...	...

Tab. 5-3: Neuronen-PE Zuordnung in Abhängigkeit der Neuronennummer

Bei einer Gesamtanzahl von 32 Neuronen (Neuronennummer 0 bis 31) werden somit je 11 von 32 Neuronen der PE0 und PE1 zugeordnet (11/32 ≈ 34,4 %), und 10 von 32 Neuronen werden der PE2 zugewiesen (10/32 ≈ 31,2 %, siehe auch Tab. 5-3 rechts).

Bei der spärlichen Vernetzungsstruktur werden Membranpotential und Synapsengewichte, wie in Abb. 5-3 auf Seite 97 angegeben, sequentiell hintereinander von einem BSI-Modul eingelesen (*Burst Mode*). Dann beginnt dieses BSI-Modul mit der numerischen Integration, bei der Funktionszwischenwerte aufgrund der geringen Anzahl an Synapsengewichten im FPGA gehalten werden können. Während dieser Berechnung beginnt das nächste BSI-Modul der PE mit dem Einlesen von Membranpotential und Synapsengewichten des nächsten Neurons. Wenn das erste BSI-Modul noch nicht in der Lage ist, das aktualisierte Membranpotential und die neu berechneten Synapsengewichte in den SDRAM-Speicher zurückzuschreiben, wird das nächste betriebsbereite BSI-Modul aktiviert. Aufgrund von Logik- und Speicher-Ressourcen des NSC-FPGAs können bis zu 8 BSI-Module innerhalb einer PE parallel betrieben werden.

Vollvernetzung

Bei einer dichten Vernetzungsstruktur, wie der lateralen Vollvernetzung, bei der ein Neuron mit jedem Neuron in derselben Netzwerkschicht verbunden ist, können aufgrund der Vielzahl an zu berechnenden Synapsengewichten, die Funktionszwischenwerte nicht im FPGA zwischengespeichert werden. In diesem Fall wird nur ein PE verwendet, welches die Kontrolle zu allen 3 Datenkanälen der 3 SDRAM-Speicher hat. Im ersten Berechnungsschritt der numerischen Integration werden das Membranpotential und alle Synapsengewichte vom SDRAM-Speicher gelesen, in dem sich der NIB des Neurons befindet. Zeitgleich werden die Funktionszwischenwerte, die sich nach Gl. 4-4 auf Seite 74 aus der Summe von Funktionswert und Ableitung zusammensetzen, über einen zweiten Datenkanal in einen für Funktionszwischenwerte reservierten Speicherbereich eines anderen SDRAM-Speichers zurückgeschrieben. In den nachfolgenden Berechnungsschritten der

numerischen Integration werden dann nach Gl. 4-5 auf Seite 75 stets 2 Funktionswerte benötigt, um den nächsten Funktionszwischenwert zu berechnen. Es müssen folglich 3 separate Datenkanäle zur Verfügung stehen, damit von 2 Datenkanälen zeitgleich gelesen und über den 3. Datenkanal parallel geschrieben werden kann. Um die beschriebene effiziente Datenkanalauslastung bei einer dichten Vernetzungsstruktur realisieren zu können, sind bei der SEE Emulationsplattform 3 separate Datenkanäle für das NSC-FPGA vorgesehen worden.

5.1.3 Netzwerktopologieberechnung

Das NTC-FPGA ist für die Netzwerktopologieberechnung verantwortlich und berechnet die präsynaptische Pulsaktivität in den Zuständen *Topvec-Spike* und *Topvec-Update* der NTC-Phase. Im Zustand *Update* der NTC-Phase werden diejenigen postsynaptischen Neuronen bestimmt, die aufgrund eines Puls-Start-Ereignisses in den Erregungszustand gelangen. Das Blockdiagramm mit den HW-Modulen, die sich im NTC-FPGA befinden, ist in Abb. 5-6 dargestellt.

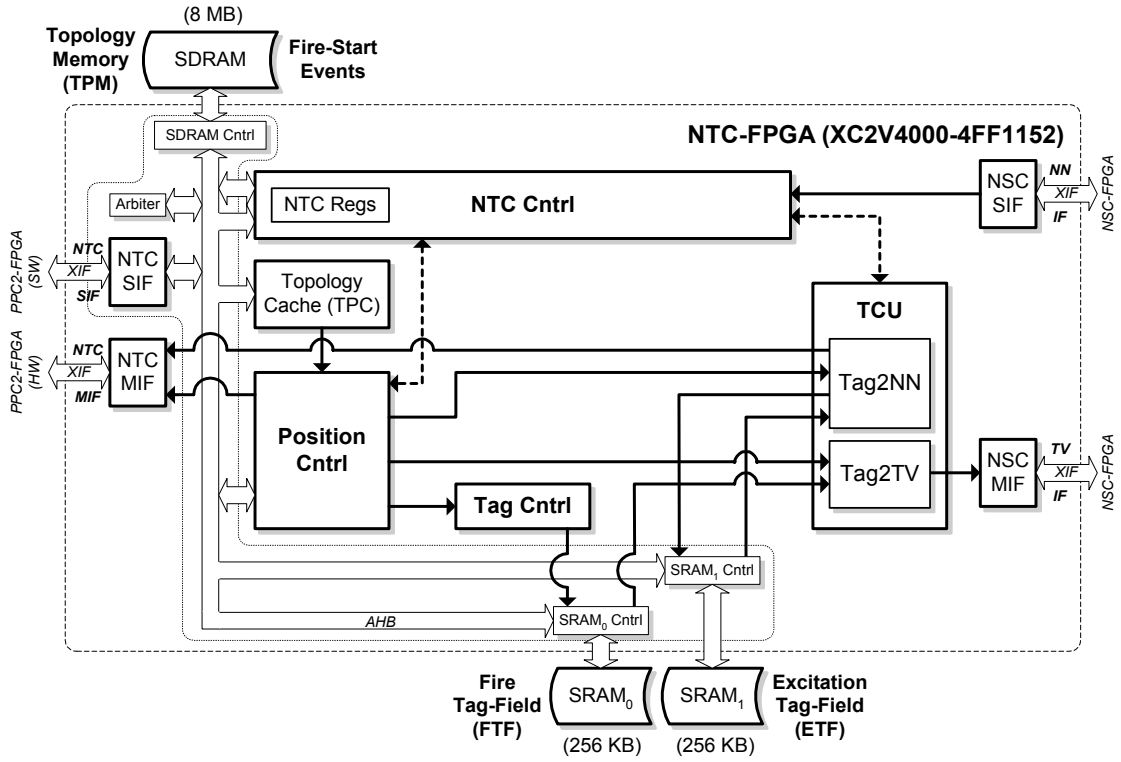


Abb. 5-6: Blockdiagramm des FPGAs für die Netzwerktopologieberechnung (NTC-FPGA)

HW-Module im NTC-FPGA

Der SW- und der HW-Datenkanal werden durch die HW-Module NTC-SIF und NTC-MIF realisiert (siehe Abb. 5-6 links). Über den SW-Datenkanal erfolgt die Initialisierung der Speicher und die Konfiguration der HW-Module über die Software, die vom Prozessor PPC0 auf dem PPC2-FPGA (siehe auch Abb. 5-1 auf Seite 92) ausgeführt wird. Über den HW-Datenkanal werden die erregten Neuronen aus der DEL in die Positionsbestimmungseinheit (Position-Cntrl) geladen. Anhand der Neuronencodierung und den zugehörigen Verbindungsinformationen, die sich im TPC (*Topology Cache*) befinden, werden durch Leseoperationen auf das FTF die Pulszustände in Form von Tags (1 Bit-Markierungen) der präsynaptischen Neuronen bestimmt. Diese Tags werden im HW-Modul TCU (*Topology Computation Unit*) zu einem Topologie-Vektor zusammengeschoben (Tag2TV) und über den Datenkanal für Topologie-Vektoren an das NSC-FPGA über das HW-Modul NSC-MIF versendet (rechts unten in Abb. 5-6).

Neuronen, die an einem Puls-Start-Ereignis beteiligt sind, werden zwischen den Zuständen *Topvec-Spike* und *Topvec-Update* vom NSC-FPGA über das HW-Modul NTC-

SIF (rechts oben in Abb. 5-6) gesendet und im TPM (*Topology Memory*) zwischengespeichert. Die Positionsbestimmungseinheit holt im Zustand *Update* der NTC-Phase diese Neuronen aus dem TPM-Speicher und teilt der TCU die postsynaptischen Verbindungen als auch die Codierung der postsynaptischen Neuronen mit. Anhand dieser Informationen wird durch Lesen des ETFs überprüft (Tag2NN), welche postsynaptische Neuronen noch nicht erregt sind. Nicht erregte Neuronen werden dann im ETF als erregt markiert und von der TCU über das HW-Modul NTC-MIF in die DEL geschrieben.

Neuronen, die an einem Puls-Stop-Ereignis beteiligt sind, werden über den HW-Datenkanal aus der DEL von der Positionsbestimmungseinheit gelesen. Diese versorgt die Markierungskontrolleinheit (Tag-Cntrl) mit Neuronenadressen und Tag-Positionen, damit diese dann die Feuerzustandsmarkierungen im FTF der Neuronen zurücksetzen kann (siehe auch Anhang E auf Seite 169).

Eine detaillierte Darstellung der Funktionsweise der HW-Module im NTC-FPGA sind den Arbeiten in [Geik07] zu entnehmen.

5.1.3.1 Berechnung von Topologie-Vektoren

Für die Umwandlung von gelesenen Tags aus dem FTF in Topologie-Vektoren sind die Verbindungsdefinitionen in Form von Topologie-Parametern à 32 Bit der Netzwerkschichten eines Neurons notwendig. Diese befinden sich im TPC-Speicher und sind für eine Kaskaden- oder eine Gabor-Architektur unterschiedlich.

Kaskaden-Architektur

In Abb. 5-7 sind die Topologie-Parameter für eine Kaskaden-Architektur (siehe auch Kapitel 4.4.1 auf Seite 85) dargestellt. Für jede Netzwerkschicht existiert eine laterale Verbindungsdefinition (siehe Abb. 5-7a), welche den lateralen Verbindungstyp, den Schichttyp (TYPE) und die Schichtgröße (X_MAX, Y_MAX) beschreibt. Weiterhin kann eine Netzwerkschicht bis zu 32 Quell- und bis zu 32 Zielverbindungen haben, die in einer externen Verbindungsdefinition beschrieben sind (siehe Abb. 5-7b).

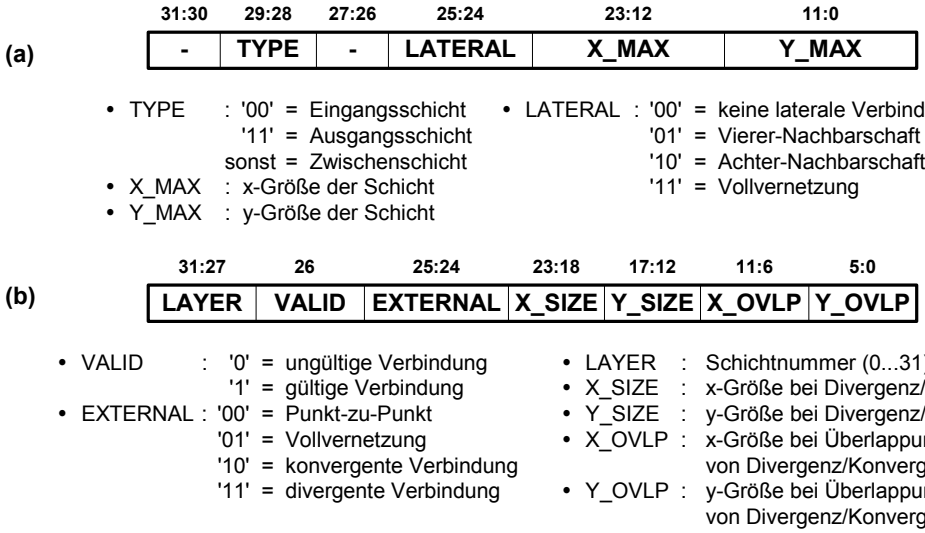


Abb. 5-7: Topologie-Parameter bei einer Kaskaden-Architektur: (a) Laterale Verbindungsdefinition, (b) Externe Verbindungsdefinition

Bei externen Verbindungen wird neben Punkt-zu-Punkt Verbindung und Vollvernetzung auch zwischen konvergenter (zusammenlaufender) und divergenter (auseinanderlaufender) Verbindung unterschieden. Bei diesen beiden Verbindungsarten sind zusätzliche Angaben über Projektionsgröße (X_SIZE, Y_SIZE) und mögliche Überlappungen (X_OVLP, Y_OVLP) erforderlich. In Abb. 5-8 ist ein Beispiel für externe konvergente und divergente Verbindungen skizziert. Aus Sicht der Netzwerkschicht 1 (L = 1) besteht

zur Netzwerkschicht 0 eine konvergente Quellverbindung und zur Netzwerkschicht 2 eine divergente Zielverbindung.

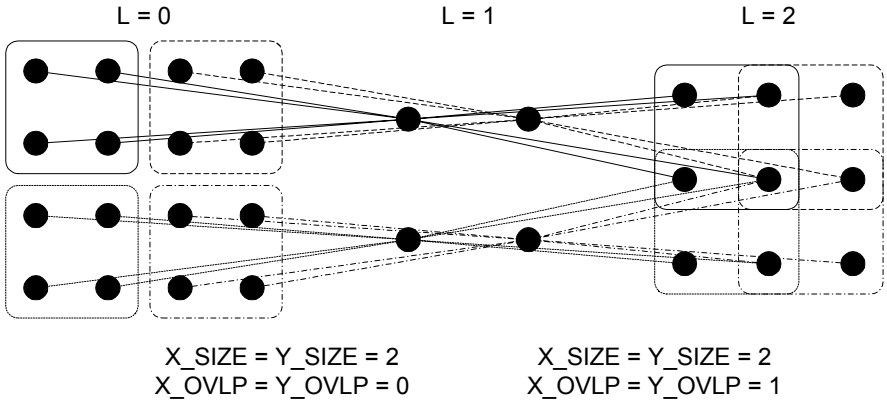


Abb. 5-8: Konvergente und divergente Verbindungen

Wenn das NTC-FPGA ein erregtes Neuron aus der DEL liest, wird anhand der Neuronencodierung (nach Abb. 4-20 auf Seite 86) die Neuronenadresse $ADDR_{NEURON}$ und die Tag-Position POS_{NEURON} des Neurons in den Markierungsspeichern ermittelt (siehe auch Anhang E auf Seite 169). Bei der in Abb. 4-19 auf Seite 85 dargestellten Kaskaden-Architektur erhält man beispielsweise für das Neuron ($BADDR_0 = 0$, $x_{MAX,0} = y_{MAX,0} = 10$) mit der Codierung `0x0000a001` ($\Rightarrow$ L = 0, x = 10, y = 1) nach der Gl. A-1 im Anhang auf Seite 170 die Neuronenadresse $ADDR_{NEURON} = (1\text{-}1)\cdot 1+1\text{-}1+0 = 0$ und die Tag-Position $POS_{NEURON} = (10\text{-}1)\%32 = 9$. Dem Neuron ($BADDR_2 = 20$, $x_{MAX,2} = y_{MAX,2} = 10$) mit der Codierung `0x6200100a` ($\Rightarrow$ L = 2, x = 1, y = 10) wird die Neuronenadresse $ADDR_{NEURON} = (10\text{-}1)\cdot 1+1\text{-}1+20 = 29$ mit $POS_{NEURON} = (1\text{-}1)\%32 = 0$ zugewiesen. Anhand der Neuronenadresse, der Tag-Position und der Verbindungsdefinitionen werden die Tags der präsynaptischen Neuronen von der TCU aus dem Markierungsspeicher (FTF) gelesen. Beim Zusammensetzen des Topologie-Vektors werden die Tags der präsynaptischen Neuronen nach absteigender Neuronenadresse und aufsteigender Tag-Position sortiert. Dies entspricht einer Sortierung nach aufsteigenden Neuronennummern.

Gabor-Architektur

Bei einer Gabor-Architektur (siehe auch Kapitel 4.4.2 auf Seite 86) werden für die Gabor-Detektordefinition 2 Topolgieparameter verwendet.

(a)

31:20	19:16	15:8	7:0
X_INP	RF	X_PIXD	Y_PIXD

31:16	15:0
GDP_MAX	RF_MAX

- X_INP : x-Größe der Eingangsschicht
- RF : Anzahl (2^{RF}) der RF-Masken (in horizontaler Richtung)
- X_PIXD : Pixelabstand zwischen 2 RF-Masken (horizontal)
- Y_PIXD : Pixelabstand zwischen 2 RF-Masken (vertikal)
- GDP_MAX : Anzahl der Gradientenpaare
- RF_MAX : Anzahl der RF-Masken

(b)

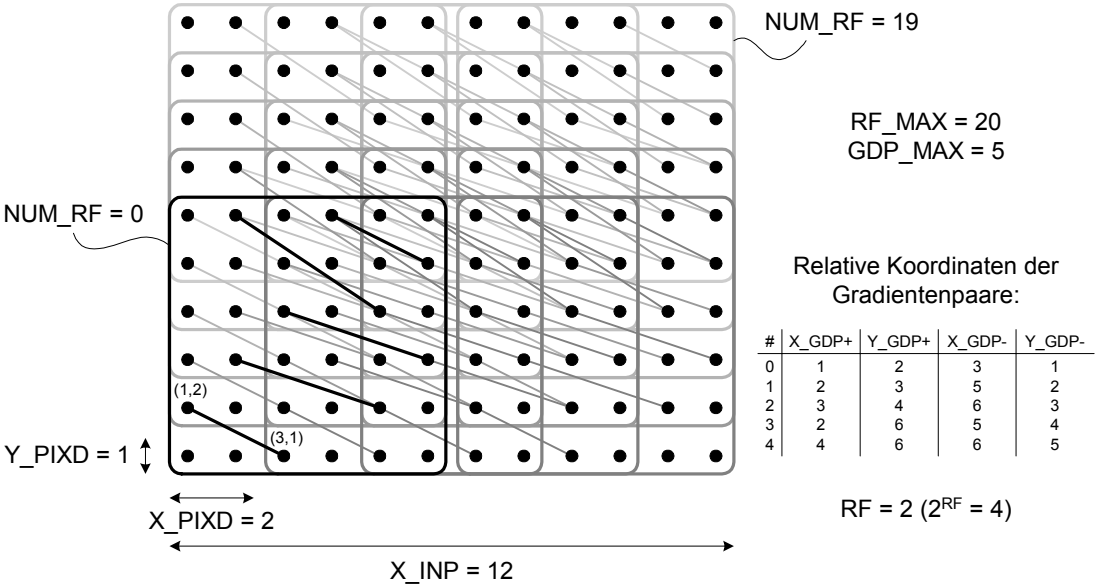
31:24	23:16	15:8	7:0
X_GDP+	Y_GDP+	X_GDP-	Y_GDP-

- X_GDP+ : x-Koordinate des ersten Gradientenneurons
- Y_GDP+ : y-Koordinate des ersten Gradientenneurons
- X_GDP- : x-Koordinate des zweiten Gradientenneurons
- Y_GDP- : y-Koordinate des zweiten Gradientenneurons

Abb. 5-9: Topologie-Parameter bei einer Gabor-Architektur: (a) Gabor-Detektordefinition, (b) relative Koordinaten der Gradientenpaare einer RF-Maske

Diese Parameter beschreiben zum einen die Anordnung der RF-Masken in horizontaler Richtung (RF und X_PIXD) und in vertikaler Richtung (Y_PIXD) auf der Eingangsschicht

und beinhalten zum anderen die Gesamtanzahl der RF-Masken (RF_MAX) und die Anzahl der Gradientenpaare pro RF-Maske (GDP_MAX, siehe Abb. 5-9a). Weiterhin werden die relativen Koordinaten der Gradientenpaare einer RF-Maske benötigt, um die präsynaptischen Neuronen von Korrelations- und Dekorrelations-Neuronen bestimmen zu können (siehe Abb. 5-9b).



#	X_GDP+	Y_GDP+	X_GDP-	Y_GDP-
0	1	2	3	1
1	2	3	5	2
2	3	4	6	3
3	2	6	5	4
4	4	6	6	5

Abb. 5-10: Anordnung von 20 RF-Masken mit je 5 Gradientenpaaren auf der Eingangsschicht

In Abb. 5-10 ist ein Beispiel für die Anordnung von 20 RF-Masken auf einer 12×10 großen Eingangsschicht skizziert. Die Durchnummerierung der RFs erfolgt von links nach rechts und von unten nach oben. Die RF-Maske 0 (RF_NUM = 0) liegt somit in Abb. 5-10 links unten und die RF-Maske 19 (RF_NUM = 19) rechts oben. Die Gesamtanzahl der RF-Masken ergibt sich aus den Parametern X_PIXD, Y_PIXD, X_INP und RF. Die 5 Gradientenpaare einer RF-Maske sind als verbundene Neuronenpaare dargestellt, und die relativen Koordinaten innerhalb einer RF-Maske sind rechts in Abb. 5-10 in einer Tabelle angegeben.

Bei der Berechnung der Topologie-Vektoren von Korrelations- und Dekorrelations-Neuronen wird anhand der Gradientenpaar-Nummer NUM_GDP (siehe Abb. 4-23 auf Seite 88) auf die relativen Koordinaten der Gradientenpaare zugegriffen. Mit der RF-Nummer NUM_RF wird die entsprechende RF-Maske ausgewählt. Das Auslesen der Pulszustände aus dem FTF erfolgt wieder durch die Umwandlung der Neuronencodierung in Neuronenadresse und Tag-Position. Für eine Gabor-Architektur nach Abb. 5-10 mit den Parametern $x_{MAX} = 12$, $y_{MAX} = 10$, $RF_MAX = x_{RF} \cdot y_{RF} = 20$ und $GDP_MAX = x_{GDP} \cdot y_{GDP} = 5$ ergeben sich für die angegebenen Neuronencodierungen folgende Neuronenadressen und Tag-Positionen:

- `0x18002604` ($\Rightarrow$ Korrelations-Neuron N_{COR}, RF = 1, $NUM_{RF} = 19$, $NUM_{GDP} = 4$)
  $ADDR_{NEURON,COR} = 10+(1 \cdot 20+19) \cdot 1+0 = 49$
  $POS_{NEURON,COR} = 4\%32 = 4$ (siehe Gl. A-4 und Gl. A-6 im Anhang auf Seite 170)
- `0x20000000` ($\Rightarrow$ Dekorrelations-Neuron N_{DEC}, RF = 0, $NUM_{RF} = NUM_{GDP} = 0$)
  $ADDR_{NEURON,DEC} = 50+(0 \cdot 20+0) \cdot 1+0 = 50$
  $POS_{NEURON,DEC} = 0\%32 = 0$ (siehe Gl. A-4 und Gl. A-7)
- `0x48002600` ($\Rightarrow$ Summen-Neuron N_{SUM}, RF = 1, $NUM_{RF} = 19$)
  $ADDR_{NEURON,SUM} = 90+2 = 92$
  $POS_{NEURON,SUM} = 4 \cdot (19\%8)+1 = 13$ (siehe Gl. A-5 und Gl. A-7)
- `0x50000000` ($\Rightarrow$ Summenkorrelations-Neuron N_{SCOR}, $NUM_{RF} = 0$)
  $ADDR_{NEURON,SCOR} = 90+0 = 90$
  $POS_{NEURON,SCOR} = 4 \cdot (0\%8)+2 = 2$ (siehe Gl. A-5 und Gl. A-7)

- `0x70002600` (⇒ Summendekorrelations-Neuron N_{SDEC}, NUM_{RF} = 19)
  $ADDR_{NEURON,SDEC}$ = 90+2 = 92
  $POS_{NEURON,SDEC}$ = 4·(19%8)+3 = 15 (siehe Gl. A-5 und Gl. A-7)

TPC

Die Abspeicherung der Topologie-Parameter, die in Abb. 5-7 auf Seite 103 und in Abb. 5-9 auf Seite 104 dargestellt wurden, erfolgt im TPC-Speicher. Diese ist für Kaskaden- und Gabor-Architektur unterschiedlich und in Abb. 5-11 illustriert.

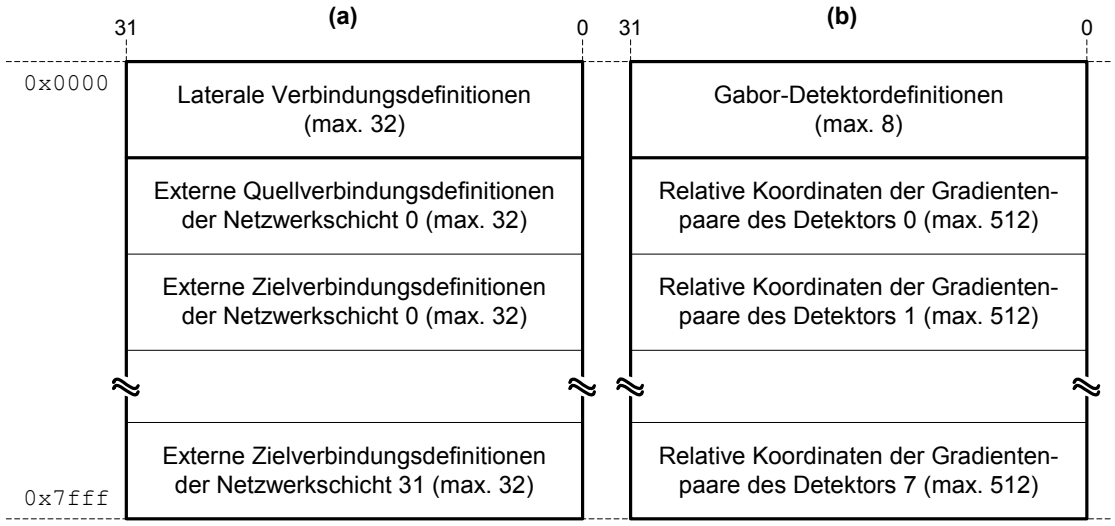


Abb. 5-11: TPC-Speichernutzung (32 KB): (a) Kaskaden-Architektur: laterale Verbindungsdefinition (Abb. 5-7a auf Seite 103), externe Verbindungsdefinition (Abb. 5-7b auf Seite 103), (b) Gabor-Architektur: Gabor-Detektordefinition (Abb. 5-9a auf Seite 104), relative Koordinaten der Gradientenpaare (Abb. 5-9b auf Seite 104)

5.1.3.2 Berechnung von postsynaptischen Neuronen

Für die Berechnung der postsynaptischen Neuronen müssen die Puls-Start-Ereignisse verarbeitet werden, die zwischen den Zuständen *Topvec-Spike* und *Topvec-Update* der NTC-Phase vom NSC-FPGA gesendet werden. Die Neuronen, die ein Puls-Start-Ereignis verursachen, werden temporär im TPM-Speicher abgelegt. Im Zustand *Update* der NTC-Phase werden diese dann von der Positionsbestimmungseinheit zur weiteren Verarbeitung vom TPM-Speicher gelesen.

Kaskaden-Architektur

Ob sich ein Neuron bereits in der DEL befindet, wird anhand der Tags im ETF festgestellt, und bei einer Kaskaden-Architektur können die postsynaptischen Neuronen durch die laterale Verbindungsdefinitionen und der externen Zielverbindungsdefinitionen, die sich im TPC-Speicher befinden (siehe Abb. 5-11) ermittelt werden. Nachdem die Positionsbestimmungseinheit ein Neuron aus dem TPM-Speicher gelesen hat, wird anhand der Neuronencodierung zuerst kontrolliert, ob es sich um ein Eingangsschicht- oder Zwischenschicht-Neuron handelt (TYPE in Abb. 4-20 auf Seite 86). Dann wird anhand der Netzwerkschichtnummer (LAYER in Abb. 4-20) überprüft, ob für diese Netzwerkschicht im TPC-Speicher eine laterale Verbindung (LATERAL Abb. 5-7a auf Seite 103) und gültige externe Zielverbindungen (VALID in Abb. 5-7b) existiert. Anhand der Neuronenadresse und Tag-Position des Neurons werden dann die Adressen und Tag-Positionen aller postsynaptischen Neuronen an das HW-Modul Tag2NN in der TCU weitergeleitet, damit die Tags dieser Neuronen aus dem ETF gelesen werden können. Zeitgleich werden die Neuronencodierungen aller postsynaptischen Neuronen von der Positionsbestimmungseinheit in ein FIFO der TCU geschrieben. Die Überprüfung der Tags im ETF erfolgt durch Lese-Modifizier-Schreibe Operationen (*Read-Modify-Write*). Durch diese Operationen werden die Tags der postsynaptischen Neuronen in jedem Fall gesetzt und somit die Neuronen als erregt markiert. Wird im Lesepfad einer Lese-Modifizier-Schreibe Operation festgestellt, dass das Tag eines postsynaptischen Neurons nicht markiert ist (1 Bit-Markierung = 0), ist ein nicht-erregtes postsynaptisches Neuron identifiziert. Jetzt wird mit Hilfe eines Lesezeigers der von der TCU gesteuert wird, die entsprechende Neuronenco-

dierung dieses Neurons aus dem FIFO gelesen, in dem die Neuronencodierungen von der Positionsbestimmungseinheit abgelegt wurden. Dieser Lesezeiger wird ansonsten von der TCU immer um die Anzahl der Tags erhöht, die bei einer Lese-Modifizier-Schreibe Operation gesetzt werden. Beim dem FIFO zwischen Positionsbestimmungseinheit und TCU wird also nicht jede Neuronencodierung ausgelesen, sondern es können Neuronencodierungen beim Auslesen je nach Erhöhung des Lesezeigers übersprungen werden. Die Neuronencodierung des nicht-erregten Neurons wird nun vom HW-Modul Tag2NN über den HW-Datenkanal in die DEL geschrieben. Durch die Lese-Modifizier-Schreibe Operation im ETF und durch den Eintrag in die DEL gilt dieses Neuron nun als erregt und wird beim nächsten Emulationsdurchlauf mit berücksichtigt.

Gabor-Architektur

Bei der Gabor-Architektur erfolgt die Ermittlung der postsynaptischen Neuronen mit Ausnahme der Eingangsschicht-Neuronen wie bei einer Kaskaden-Architektur. Aufgrund der regulären Verbindungsstruktur eines Gabordetektors (siehe auch Abb. 4-21 auf Seite 87) können anhand bestimmter Variablen in der Neuronencodierung die postsynaptischen Neuronen ermittelt werden. Das postsynaptische Neuron eines Korrelations-Neurons ist immer ein Dekorrelations-Neuron und ist mit den Variablen RF, NUM_RF und NUM_GDP der Neuronencodierung identifizierbar. Der Puls eines Dekorrelations-Neuron trifft stets ein Summen-Neuron mit den identischen Variablen RF und NUM_RF. Die postsynaptischen Neuronen eines Summen-Neurons sind Summenkorrelations- und Summendekorrelations-Neuron und sind anhand der Variablen RF und NUM_RF bestimmbar. Ein Summenkorrelations-Neuron feuert immer auf ein Summendekorrelations-Neuron mit gleicher RF-Nummer (NUM_RF). Summendekorrelations-Neuronen entsprechen Ausgangsschicht-Neuronen eines Gabor-Detektors und besitzen somit keine postsynaptischen Neuronen.

Die postsynaptischen Neuronen von Eingangsschicht-Neuronen werden aufgrund der irregulären Verbindungsstrukturen und der unterschiedlichen Überlappungsdichte von RF-Masken (siehe Abb. 5-10 auf Seite 105) bei der Gabor-Architektur explizit im TPM-Speicher abgelegt. Für jedes Eingangsschicht-Neuron ist in Abhängigkeit der Neuronennummer eine Header-Informationen mit Sprungadresse und Anzahl der postsynaptischen Neuronen hinterlegt. An der Sprungadresse sind alle postsynaptischen Neuronen (Korrelations- und Dekorrelations-Neuronen der Gabor-Detektoren) aufgelistet, die von der Positionsbestimmungseinheit gelesen werden, wenn ein Eingangsschicht-Neuron ein Puls-Start-Ereignis verursacht.

TPM

Die Speichernutzung des TPM-Speichers ist in Abb. 5-12 dargestellt.

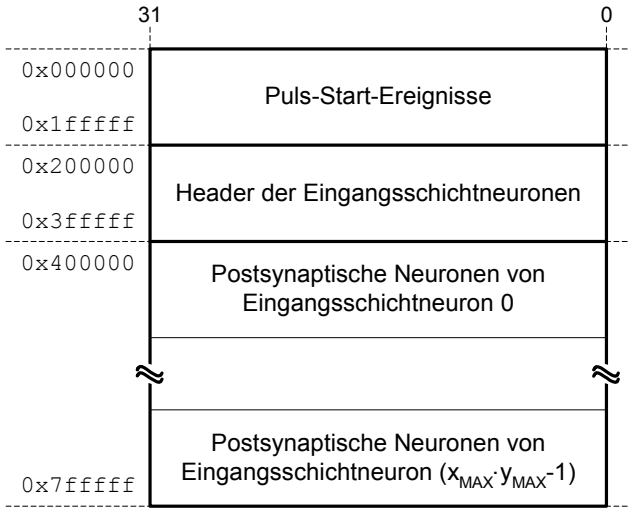


Abb. 5-12: TPM-Speichernutzung (8 MB)

Da im extremsten Fall alle Neuronen gleichzeitig pulsen können, muss für die Puls-Start-Ereignisse ein Speicherplatz von 2 MB vorgesehen werden, was der Speicherkapazität der DEL entspricht. Nur bei der Gabor-Architektur sind weiterhin noch Header-Infor-

mationen der Eingangsschichtneuronen und die Liste der postsynaptischen Neuronen für jedes einzelne Eingangsschicht-Neuron notwendig.

Die Zugriffe von erregten Neuronen in der DEL wird während der PCNN Emulation zeitgleich sowohl für die Neuronenzustands- als auch für die Netzwerktopologieberechnung durchgeführt, und wird im folgenden näher erläutert.

5.1.4 Listenzugriffe von erregten pulscodierten Neuronen

Die Listenzugriffe auf die DEL, die wechselweise vom NTC- und vom NSC-FPGA vorgenommen werden, sind in Abb. 5-13 dargestellt. Dabei werden verschiedene Szenarien, wie Puls-Start- und Puls-Stop-Ereignisse von Neuronen der Eingangsschichten (IN1 und IN0), Puls-Start- und Puls-Stop-Ereignisse von Neuronen der Zwischenschichten (N1 und N0), sowie Kombinationen davon illustriert.

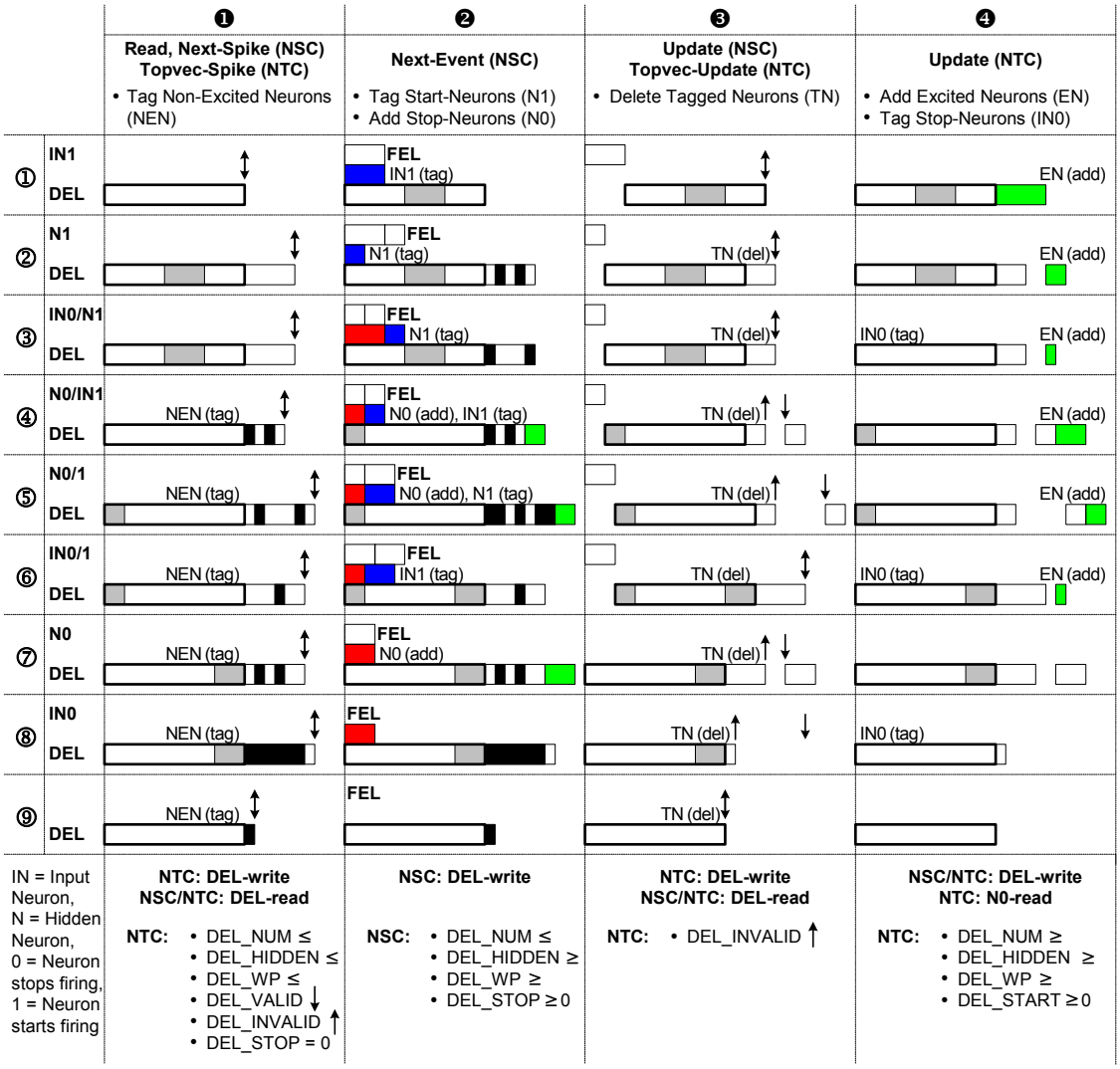


Abb. 5-13: Listenzugriffe von NSC- und NTC-FPGA auf die DEL

In den 4 Spalten ❶ bis ❹ sind die Zustände der NSC- und der NTC-Phase dargestellt, die für eine Emulationsereigniszeit durchlaufen werden müssen (siehe auch Abb. 4-7 auf Seite 71). In den Reihen ① bis ⑨ sind insgesamt 9 Durchläufe für die Aktualisierung einer Emulationsereigniszeit illustriert. Der Emulationsereignistyp (IN0, IN1, N0 oder N1) ist dabei am Anfang jeder Reihe angegeben. Die obere Zeile einer Reihe zeigt den Füllstand der FEL, und die untere Zeile einer Reihe demonstriert den Füllstand der DEL in einer horizontalen Balkenform. Die unterste Reihe kennzeichnet, in welchen Zuständen die DEL

vom NSC- bzw. NTC-FPGA beschrieben bzw. gelesen wird und welche Emulationsparameter der Tab. 5-1 auf Seite 93 vom NSC- bzw. NTC-FPGA modifiziert werden.

DEL-Zugriffe

Grundsätzlich enthält die DEL alle erregten Neuronen. Die Neuronen der Eingangsschichten stehen immer in der DEL (fettumrahmter Balken in Abb. 5-13), während Neuronen aus Zwischenschichten nur in der DEL stehen, wenn sie entweder erregt sind, da sie momentan einen Puls empfangen, oder wenn sie aufhören zu pulsen (Puls-Stop-Ereignis). Im initialisierten Zustand sind in der FEL keine Neuronen abgelegt, und die DEL enthält alle Neuronen der Eingangsschichten (siehe ❶①).

Im *Next-Spike* Zustand werden vom NTC-FPGA die Topologie-Vektoren von gültig markierten (erregten) Neuronen in der DEL berechnet. Wenn ein Neuron keinen Puls (Topologie-Vektor dieses Neurons ist Null) und keinen externen Reiz empfängt, wird das Neuron als ungültig markiert, da es unerregt ist (siehe schwarze Markierungen in ❶④ bis ❶⑨). Das NSC-FPGA liest zeitgleich von der DEL und ermittelt den nächsten Zeitpunkt eines Puls-Start-Ereignisses.

Im *Next-Event* Zustand der NSC-Phase werden vom PPC1 alle Puls-Start-Ereignisse an die FEL angehangen, und das NSC-FPGA markiert diese zeitgleich in der DEL als ungültig (siehe schwarze Markierungen für Neuronen der Zwischenschichten und graue Markierungen innerhalb des fettumrahmten Balkens für Neuronen der Eingangsschichten in Spalte ❷). Bei Puls-Stop-Ereignissen werden die Neuronen der Zwischenschichten an die DEL angehangen (siehe grüne Markierung in ❷④, ❷⑤ und ❷⑦), da diese für den nächsten Emulationsdurchlauf berücksichtigt werden müssen. Neuronen, die aufhören zu pulsen, befinden sich möglicherweise in einem erregten Zustand, da sie wieder Pulse oder externe Reize empfangen können.

Im *Topvec-Update* Zustand der NTC-Phase werden vom NTC-FPGA alle ungültig markierten Neuronen von Zwischenschichten aus der DEL gelöscht, indem nur gültige Neuronen von Zwischenschichten wieder in die DEL zurückkopiert werden (siehe Spalte ❸). Dadurch entsteht unter Umständen ein Listenabschnitt von ungültigen Neuronen in der DEL, der sich zwischen 2 gültigen Listenabschnitten befindet (siehe ❸④, ❸⑤ und ❸⑦). Dieser Fall tritt ein, wenn im vorherigen Zustand das NSC-FPGA Neuronen von Zwischenschichten an die DEL angehangen hat, da diese Puls-Stop-Ereignisse generieren. Der Anfang (↑) und das Ende (↓) dieses ungültigen Listenabschnitts wird durch die beiden Emulationsparameter DEL_INVALID und DEL_VALID beschrieben (siehe `0x0020` und `0x0024` in Tab. 5-1 auf Seite 93), die vom NTC-FPGA bei jedem Emulationsdurchlauf aktualisiert werden.

Im *Update* Zustand der NTC-Phase (Spalte ❹) werden die aufgrund der aufgetretenen Puls-Start-Ereignisse neu erregten Neuronen (aus Zwischenschichten) bestimmt und an die DEL angehangen (siehe grüne Markierung in ❹① bis ❹⑥). Zeitgleich werden eventuelle Puls-Stop-Ereignisse der Eingangsschichtneuronen vom NSC-FPGA in der DEL als gültig markiert, da diese für den nächsten Emulationsdurchlauf wieder berücksichtigt werden müssen (siehe ❹③, ❹⑥ und ❹⑧).

Datentransfertypen

Zur Optimierung der Lese- und Schreibzugriffe auf die DEL werden unterschiedliche Datentransfertypen unterschieden, die in Tab. 5-4 aufgelistet sind. Diese Datentransfertypen kommen bei den HW-Datenkanälen, die für Adressen und Daten einen gemeinsamen 32 Bit-breiten Bus nutzen, zwischen NTC-FPGA und PPC2-FPGA und zwischen NSC-FPGA und PPC2-FPGA zum Einsatz. Die Busschnittstelle bei diesen beiden HW-Daten-

kanälen wurde um ein 2 Bit-breites Transferkontrollsignal erweitert, um die 4 verschiedenen Datentransfertypen für Lese- und Schreibzugriffe codieren zu können.

Transferart	Transfertyp		Beschreibung
Schreiben	Normal	00	Regulärer Schreibtransfer
	Tag0	01	Markieren eines Neurons als gültig
	Tag1	10	Markieren eines Neurons als ungültig
	Delete	11	Löschen eines gültigen Neurons
Lesen	Normal	00	Regulärer Lesetransfer
	Init	01	Initialisieren des Kopierzeigers
	Send	10	Kopieren eines Speicherbereichs
	Copy	11	Senden des Kopierzeigers

Tab. 5-4: Datentransfertypen für den Zugriff auf die DEL

Bei Schreibzugriffen wird neben dem normalen Schreibtransfer zwischen den Schreibtransfers *Tag0*, *Tag1* und *Delete* unterschieden. Hierbei werden nur die Adressen der Neuronen übertragen. Bei einem *Tag0*-Transfer bzw. *Tag1*-Transfer wird das Zustandsbit STATE (siehe auch Abb. 4-20 auf Seite 86 und Abb. 4-23 auf Seite 88) invertiert, indem auf dem PLB-Bus im PPC2-FPGA intern ein zusammenhängender Lese- und Schreibzugriff (*Read-Modify-Write*) des Neurons durchgeführt wird. Bei einem *Delete*-Transfer werden Neuronen von Zwischenschichten als ungültig markiert, indem das Datum `0xffffffff` an die vom NTC- bzw. NSC-FPGA übertragene Neuronen-Adresse geschrieben wird. Durch diese 3 Schreibtransfertypen ist es möglich, nur anhand einer Neuronenadresse, die vom NTC- oder NSC-FPGA gesendet wird, Neuronen in der DEL über den PLB-Bus im PPC2-FPGA zu markieren.

Bei Lesezugriffen wird neben dem normalen Lesetransfer zwischen den Lesetransfers *Init*, *Send* und *Copy* unterschieden. Beim *Init*-Transfer wird der Anfang eines Speicherbereichs in der DEL dem PPC2-FPGA mitgeteilt, indem ein Kopierzeiger initialisiert wird. Beim *Copy*-Transfer werden die Neuronen aus der DEL für das NTC-FPGA gelesen. Zeitgleich werden von den gelesenen Neuronen in der DEL nur die gültig markierten Neuronen direkt wieder in den Speicherbereich kopiert, dessen Anfang vom Kopierzeiger markiert ist. Nach jedem kopierten Neuronenblock (es werden Burstlängen von 2, 4, 8 und 16 unterstützt) wird der Kopierzeiger entsprechend inkrementiert. Nach Beendigung eines Kopiervorgangs wird durch den *Send*-Transfer dem NTC-FPGA der aktuelle Wert des Kopierzeigers mitgeteilt, damit das NTC-FPGA ermitteln kann, wieviele gültige Neuronen insgesamt kopiert wurden. Diese 3 Lesetransfertypen werden vom NTC-FPGA im *Topvec-Spike* und im *Topvec* Zustand verwendet, um den ungültigen Listenabschnitt in der DEL (siehe Spalte ❶) und ungültig markierte Neuronen in der DEL (siehe Spalte ❸) zu löschen. Anhand dieser zusätzlichen Lesetransfertypen wird bei einem Lesevorgang von Neuronen in der DEL der notwendige Löschvorgang von ungültigen Neuronen in der DEL zeitgleich durchgeführt.

Eine detaillierte Darstellung der HW-Implementierung der Busschnittstellen zwischen den FPGAs (XIF = *External Bus Interface*), die diese Datentransfertypen realisieren, und zwischen internen HW-Modulen zu den standardisierten On-Chip-Bussen AHB und PLB (IIF = *Internal Bus Interface*) sind den Arbeiten in [Grie07] zu entnehmen.

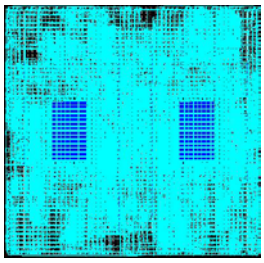
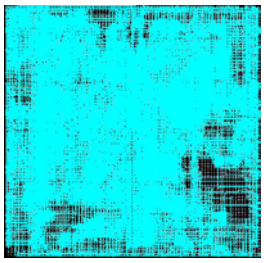
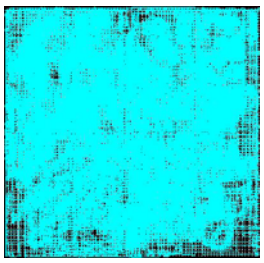
5.1.5 Hardware-Ressourcen

Die Hardware-Ressourcen stellen den Bedarf an internen Logik-, Speicher- und Makro-Blöcken (CPUs und Multiplizierer) dar, der für die Integration der HW-Module in den einzelnen FPGAs notwendig ist. Nach der Platzierung und Verdrahtung der HW-Module im

FPGA wird eine Berichtsdatei (`*.par`) erzeugt, die zusammenfasst, welche Prozentanteile an unterschiedlichen Hardware-Blöcken im FPGA verwendet wurden.

Flächenbedarf

In Tab. 5-5 sind sowohl die absoluten Zahlen als auch die Prozentanteile der verwendeten Hardware-Blöcke aller 3 FPGAs zusammengetragen. Die erste Spalte dieser Tabelle zeigt die interne Ansicht der FPGAs (mit dem Befehl `fpga_editor`), bei der alle Hardware-Blöcke und Signalleitungen dargestellt sind.

FPGA	Device uitilization summary				
PPC2-FPGA 	Number of External IOBs	454	out of	564	80%
	Number of LOCed External IOBs	454	out of	454	100%
	Number of PPC405s	2	out of	2	100%
	Number of RAMB16s	54	out of	88	61%
	Number of SLICEs	4345	out of	9280	46%
	Number of BUFGMUXs	5	out of	16	31%
	Number of DCMs	2	out of	8	25%
	Number of JTAGPPCs	1	out of	1	100%
NTC-FPGA 	Number of External IOBs	398	out of	824	48%
	Number of LOCed External IOBs	398	out of	398	100%
	Number of MULT18X18s	19	out of	120	15%
	Number of RAMB16s	39	out of	120	32%
	Number of SLICEs	11615	out of	23040	50%
	Number of BUFGMUXs	5	out of	16	31%
	Number of DCMs	2	out of	12	16%
NSC-FPGA 	Number of External IOBs	533	out of	824	64%
	Number of LOCed External IOBs	533	out of	533	100%
	Number of MULT18X18s	40	out of	120	33%
	Number of RAMB16s	88	out of	120	73%
	Number of SLICEs	18788	out of	23040	80%
	Number of BUFGMUXs	4	out of	16	25%
	Number of DCMs	2	out of	12	16%

Tab. 5-5: Einzelne FPGA-Ressourcen der Emulationsplattform
(DCM = Digital Clock Manager, BUFGMUX = Global Clock MUX Buffer)

Das PPC2-FPGA verwendet neben den beiden PPCs (PPC405) 4345 Slices und 54 interne BRAM-Speicher (RAMB16), die jeweils eine maximale Speicherkapaziztät von 2048 Bit bereitstellen. Nach [Xili97] entspricht eine Logic Cell im FPGA, von denen 2 in einem Slice enthalten sind (siehe auch Abschn. "Logic Cell" auf Seite 62 und Abb. 4-2a auf Seite 63), etwa 12 ASIC Gattern, so dass sich für die Logik im PPC2-FPGA eine Gatteranzahl von etwa $4345 \cdot 2 \cdot 12 \approx 104$ K ergibt. Die BRAMs stellen für FIFOs und interne Register eine Speicherkapazität von insgesamt $54 \cdot 2$ KB = 108 KB zur Verfügung. Weiterhin wird vom PPC2-FPGA das Hardware-Makro JTAGPPC genutzt, das für eine JTAG-Anbindung (JTAG = *Joint Test Access Group*) zu den beiden Prozessoren notwendig ist.

Das NTC-FPGA benötigt für die Hardware-Implementierung neben den 11615 Slices, und 39 internen BRAM-Speichern zusätzlich 19 Multiplizierer-Blöcke (MULT18X18). Die implementierte Hardware-Logik entspricht etwa einer Anzahl von $11615 \cdot 2 \cdot 12 \approx 279$ K Gattern und die interne Speicherkapazität beläuft sich auf $39 \cdot 2$ KB = 78 KB.

Das NSC-FPGA nutzt für die implementierten HW-Module 18788 Slices, 40 Multiplizierer-Blöcke und 88 interne BRAM-Speicher, die insgesamt eine Speicherkapazität von $88 \cdot 2$ KB = 176 KB bereitstellen. Die Gatteranzahl der verwendeten Logik-Blöcke ergibt einen Wert von etwa $18788 \cdot 2 \cdot 12 \approx 451$ K.

Operations-frequenz

Nach der Platzierung und Verdrahtung der HW-Module in den FPGAs wird eine weitere Berichtsdatei (`*.twr`) erzeugt, welche für den implementierten HW-Entwurf die längsten Logikpfade enthält, die die Verzögerungszeit von Register über kombinatorische Logik zum nächsten Register beschreiben. Dabei stellt sich heraus, dass etwa 30-40 % dieser Verzögerungszeit für die reine Logik notwendig ist (*LOGIC*) und sich 60-70 % der Verzögerungszeit auf die interne Verdrahtung zu anderen Slices oder zu Pads auswirkt (*ROUTE*). Bei dem PPC2-FPGA verwendet der längste Logikpfad 9 Logikstufen (eine Logikstufe entspricht einem Slice, welches für die Realisierung der kombinatorischen Logik verwendet wurde). Beim PPC2-FPGA entspricht der längste Logikpfad 19,63 ns (5,75 ns für Logik und 13,88 ns für Verdrahtung), was einer maximalen Operationsfrequenz von 50,9 MHz entspricht. Beim NTC- und beim NSC-FPGA benötigt der längste Logikpfad 22 Logikstufen bei einer Verzögerungszeit von 20,34 ns (8,07 ns für Logik und 12,27 ns für Verdrahtung). Dies entspricht einer maximalen Operationsfrequenz von 49,2 MHz.

Speedgrade

Diese Operationsfrequenzen ergeben sich, wenn man die leistungsschwächsten FPGAs bezüglich der Verarbeitungsgeschwindigkeit verwendet. Bei den FPGAs von Xilinx wird der gleiche FPGA-Typ in der Regel nach 3 unterschiedlichen Geschwindigkeitsklassen (*Speedgrades*) unterteilt. Würde man für alle 3 FPGAs die leistungsstärksten FPGAs (höchste Geschwindigkeitsklasse) verwenden, würde sich der längste Logikpfad beim PPC2-FPGA (Speedgrade = -7 statt -5) um 3,7 % auf 18,90 ns (52,9 MHz) und bei NTC- und NSC-FPGA (jeweils Speedgrade = -6 statt -4) um 21,1 % auf 16,05 ns (62,3 MHz) reduzieren.

5.2 Beschreibung des Software-Entwurfs

Der Entwurf der Software-Programme erfolgt gesondert für jeden der beiden PPCs, da beide Prozessoren im PPC2-FPGA unabhängig voneinander operieren können. Für jeden PPC stehen jeweils 32 KB große Befehls- und 16 KB große Daten-Cache-Speicher zur Verfügung, die in Form von BRAM-Speichern auf dem FPGA bereitgestellt werden (siehe IBRAM und DBRAM in Abb. 5-1 auf Seite 92). Kurze und entsprechend kompakte SW-Programme können somit direkt in den Cache-Speichern abgelegt werden, ohne den auf der Emulationsplattform vorhandenen Flash-Speicher nutzen zu müssen, in dem ansonsten Befehle und initialisierte Daten der SW-Programme beider PPCs abgelegt werden müssten. Im Rahmen dieser Arbeit diente die Software in erster Linie zur Steuerung und zur Überprüfung des Emulationsablaufs.

5.2.1 Steuerungsprogramme für den Emulationsablauf

Die Software, die vom PPC0 ausgeführt wird, ist für die zentrale Emulationsablaufsteuerung zuständig. Hierzu gehört insbesondere die Steuerung des Einzelschrittbetriebs während der Emulation, der unter anderem die Überprüfung von durchgeführten Datentransfers und Inhalten von Ereignislisten und Markierungsspeicher auf der Emulationsplattform erlaubt. Die Software, die auf dem PPC1 abläuft, ist während der Emulation für die Listenverwaltung der FEL zuständig. Puls-Start-Ereignisse, die vom NSC-FPGA ermittelt worden sind, werden in die Ereignisliste eingetragen und der Schreibzeiger inkrementiert. Bei Puls-Stop-Ereignissen erfolgt lediglich eine Inkrementierung des Lesezeigers der FEL. Zusätzlich übernimmt die Software die Protokollierung der aufgetretenen Emulationsereignisse, indem Neuronennummer und Ereigniszeit eines Puls-Start- bzw. eines Puls-Stop-Ereignisses in den SDRAM des PPC2-FPGAs abgelegt werden.

PPC0-Software

In Abb. 5-14 ist der prinzipielle Aufbau der SW-Programme `ppc0_main.c` und `ppc1_main.c` beider PPCs dargestellt, die aus mehreren SW-Modulen bestehen. Mit dem SW-Modul `ppc0_menu.c` wird vom PPC0 ein Konfigurationsmenü erstellt, dass in einer Endlosschleife realisiert ist. Dieses Konfigurationsmenü beinhaltet eine Menü-Ausgabe mit ASCII-Zeichen (ASCII = *American Standard Code for Information Interchange*)

anhand von `printf`-Anweisungen. Durch diese Textdarstellung ist der Benutzer der Emulationsplattform eher in der Lage durch Tastendruck, unterschiedliche Menüpunkte auszuwählen und somit einzelne SW-Module zu aktivieren. Im wesentlichen stehen die 3 SW-Module `ppc0_write.c`, `ppc0_read.c`, und `ppc0_see.c` zur Auswahl, die mit unterschiedlichen Parametern aktiviert werden können. Mit dem SW-Modul `ppc0_write.c` können Dateien mit Hexadezimal-Werten vom Host-Rechner auf die Emulationsplattform hochgeladen werden (*Upload*), die dann in anzugebene Speicherbereiche abgelegt werden. Diese Funktionalität ist notwendig, um Konfigurationsdateien, die die Netzwerkbeschreibung für eine PCNN Emulation beinhalten, auf die Emulationsplattform zu übertragen. Zusätzlich ermöglicht dieses SW-Modul das Schreiben von einzelnen Datenwörtern (32 Bit), indem der Benutzer über die Tastatur die Speicheradresse und das zu schreibende Datum eingibt. Mit dem SW-Modul `ppc0_read.c` können Speicherinhalte entweder in eine Datei auf dem Host-Rechner heruntergeladen werden (*Download*) oder aber auch direkt in Form von ASCII-Zeichen dem Benutzer angezeigt werden. Die Möglichkeit, Speicherinhalte in eine Datei auf dem Host-Rechner übertragen zu können, ist erforderlich, damit aufgetretene Emulationsereignisse nach einer PCNN Emulation visualisiert werden können. Da der PPC0 eine direkte Datenbusanbindung zum NTC-FPGA besitzt, ist es nur diesem Prozessor möglich, Register der HW-Module im NTC-FPGA sowie die Speicherinhalte der Markierungsspeicher (ETF und FTF) und des Topologiespeichers (TPM) anzuzeigen. Schließlich erfolgt mit dem SW-Modul `ppc0_see.c` die Initialisierung und Aktivierung des NTC-FPGAs. Die Aktivierung des NTC-FPGAs für die PCNN Emulation kann allerdings erst dann erfolgen, wenn zuvor die Aktivierung des NSC-FPGAs vollzogen worden ist.

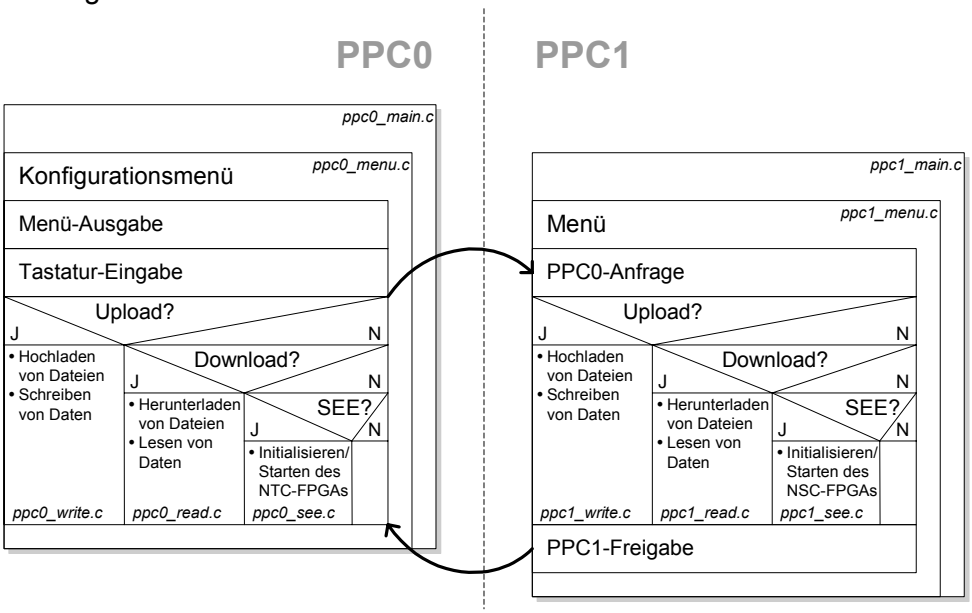


Abb. 5-14: Software-Programme der beiden PowerPCs PPC0 und PPC1

PPC1-Software

Der Aufbau des SW-Programms `ppc1_main.c` vom PPC1 entspricht dem Aufbau des SW-Programms vom PPC0. Wenn über das Konfigurationsmenü vom PPC0 eine Anfrage gestellt wird, werden entsprechende SW-Module vom PPC1 (`ppc1_write.c`, `ppc1_read.c` und `ppc1_see.c`) ausgeführt und das Ende der Ausführung anhand einer Freigabe dem PPC0 mitgeteilt. Anfragen vom PPC0 werden immer dann gestellt, wenn Speicherbereiche, wie die FEL oder der Parameterspeicher (NIB) gelesen oder beschrieben werden sollen, die nur für den PPC1 zugänglich sind. Die Register der HW-Module im NSC-FPGA sind ebenfalls nur für den PPC1 zugänglich, da nur dieser Prozessor eine direkte Datenbusanbindung zum NSC-FPGA hat. Die Ausführung der SW-Module für das Schreiben und Lesen von Speicherinhalten wird von beiden Prozessoren rein sequentiell durchgeführt. Entweder werden SW-Module vom PPC0 ausgeführt während der PPC1 unbeschäftigt ist, oder aber bei einer Anfrage vom PPC0 sind SW-Module vom PPC1 aktiv, und in dieser Zeit ist der PPC0 untätig. Erst nach der Initialisierung und Aktivierung

des NSC-FPGAs durch das SW-Modul `ppc1_see.c` erfolgt sowohl die parallele Ausführung der Software (Emulationsablaufkontrolle vom PPC0 und Listenverwaltung vom PPC1) als auch die parallele Verarbeitung der Hardware im NTC- und im NSC-FPGA.

5.2.2 Software-Ressourcen

Die Software-Ressourcen stellen den Speicherplatz dar, der für die SW-Programme der beiden PPCs notwendig ist. Die notwendige Speichergrößen der SW-Programme lassen sich mit dem Befehl `powerpc-eabi-size ppc0.elf ppc1.elf` als Zahlenangaben in Bytes ermitteln und sind in Tab. 5-6 für die jeweiligen PPCs zusammengefasst.

	TEXT	DATA	BSS	DEC	HEX
PPC0	31660	7504	4320	43484	a9dc
PPC1	22892	4040	4472	31404	7aac

Tab. 5-6: Software-Ressourcen von PPC0/PPC1 (TEXT = Code-Segment, DATA/BSS = Daten-Segmente)

Bei SW-Programmen wird zwischen Code-Segment und Daten-Segmenten unterschieden. Das Code-Segment ist les- und ausführbar (RX = *Read and Executable*) und beinhaltet alle notwendigen Befehle eines SW-Programms. Der Speicherbedarf des Code-Segments (TEXT) vom PPC0 beträgt 31660 Bytes und vom PPC1 22892 Bytes. Da in den SW-Modulen vom PPC0 mehrere Testroutinen für den Speichertest und für den Funktionstest von HW-Modulen eingebunden sind, ergibt sich der größere Speicherbedarf des Code-Segments, der aber noch nicht die Größe des Befehls-Caches von 32 KB überschreitet. Die Daten-Segmente bei einem SW-Programm sind les- und beschreibbar (RW = *Read and Writable*). Man unterscheidet bei den Daten-Segmenten zwischen einem Segment, das initialisierte Daten enthält (DATA), und einem Segment, welches uninitialisierte Daten beinhaltet und somit eine Speicherreservierung für Daten vorsieht, denen im Laufe des Programms Werte zugewiesen werden (BSS = *Block Started by Symbol*). Der fast doppelt so große Speicherbedarf an initialisierten Daten von 7504 Bytes vom PPC0 im Vergleich zu 4040 Bytes vom PPC1, liegt darin begründet, dass für die Menü-Ausgabe vom PPC0 eine Vielzahl von ASCII-Zeichen abgespeichert werden müssen. Die letzten beiden Spalten in Tab. 5-6 stellen den Gesamtspeicherbedarf in dezimaler (DEC) und hexadezimaler Form (HEX) dar.

Die Software vom PPC0 benötigt zusammenfassend etwa 31,7 KB für Befehle und 11,8 KB für Daten, und die Software, die auf dem PPC1 abläuft, braucht einen Speicherplatz von 22,9 KB für Befehle und 8,5 KB für die Abspeicherung von Daten.

5.3 Darstellung des Hardware/Software-Entwurfsfluss

Im folgenden wird der Entwurfsfluss (*Design Flow*) der Hardware- und Software-Module beschrieben, welcher in Abb. 5-15 dargestellt ist und bei der Implementierung von SEE verwendet wurde.

HW-Entwurf

Beim Entwurf der integrierten Hardware-Module findet zuallererst eine Beschreibung der Funktionalität durch eine Hardwarebeschreibungssprache (Verilog) auf der RTL (*Register Transfer Level*) Ebene statt. Der entstandene Hardware-Code (`*.v`) wird mit Hilfe des Simulators ModelSim von Mentor Graphics verifiziert (RTL Simulation). Hierfür muss eine Testumgebung (*Testbench*, `tb_*.v`) erstellt werden, die ebenfalls mit Verilog beschrieben wird und die gegebenenfalls Modelle von SRAM- und SDRAM-Speichern sowie von PPC-Prozessoren einbezieht. Neben neu erstellten Hardware-Modulen können für die Verifikation auch vorgefertigte Hardware-Makros (IP = *Intellectual Property*) eingebunden werden, die als Gatter-Netzliste vorliegen. Bei der Implementierung von SEE standen die SDRAM-Controller für das NTC- und das NSC-FPGA als Hardware-Makros (Designware von Synopsys) zur Verfügung. Mit Hilfe der Synthesewerkzeuge Precision-Synthesis von Mentor Graphics und Design-Compiler von Synopsys wird die RTL-

Beschreibung in eine Gatter-Netzliste übersetzt (siehe auch Skript für Synthese der Hardware im Anhang G auf Seite 172). Dabei wird eine simulierbare Gatter-Netzliste in Verilog (`*_synnet.v`), die mit ModelSim verifiziert werden kann (Post-Synthese Simulation), und eine EDIF-Netzliste erzeugt (EDIF = *Electronic Design Interchange Format*, `*.edf`).

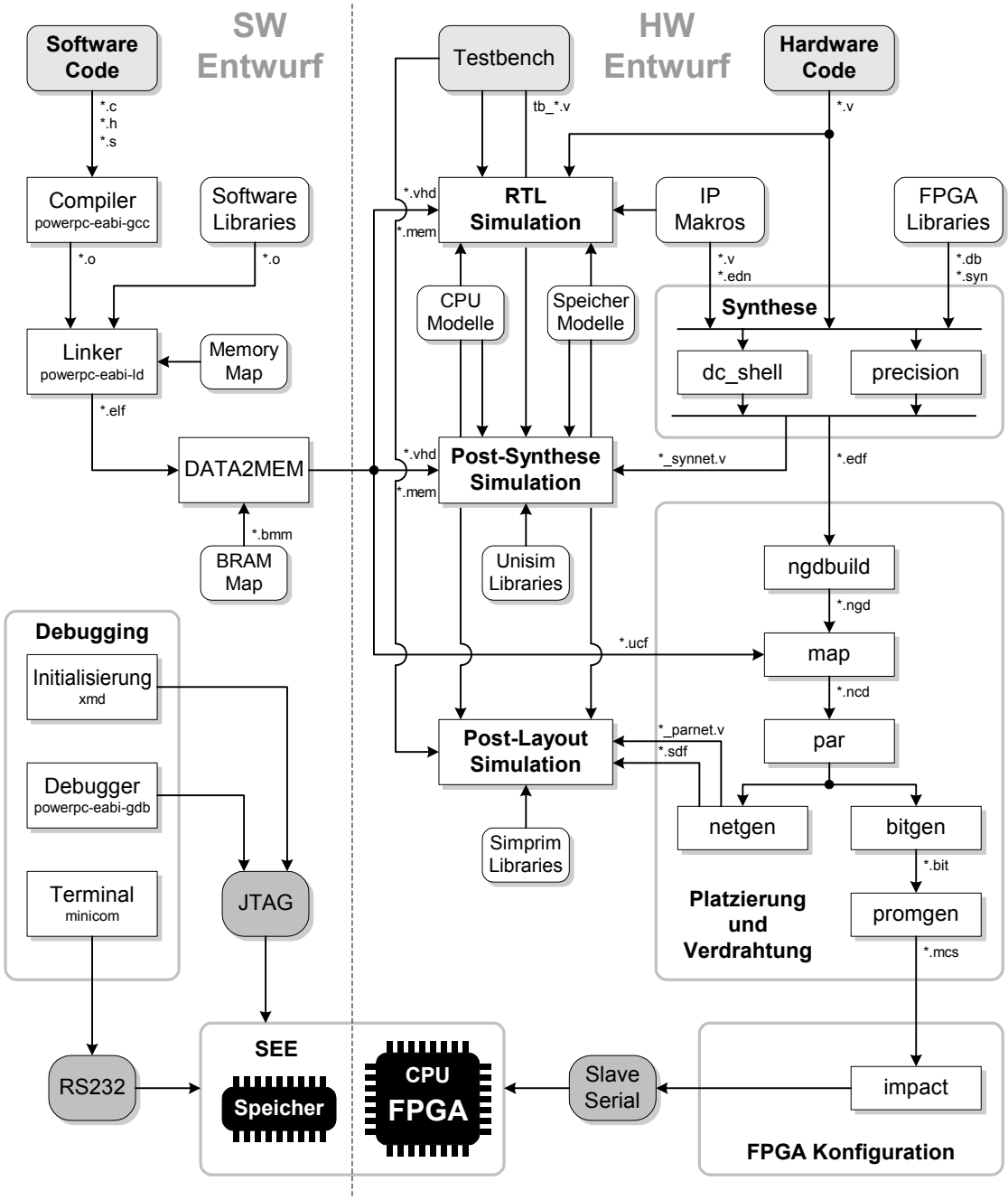


Abb. 5-15: SEE Hardware/Software-Entwurfsfluss

Bei der Post-Synthese-Simulation müssen die Gatter-Modelle der Unisim-Bibliothek (*Unified Library Primitives*) verwendet werden. Anschließend findet mit dem Werkzeug ISE (*Integrated Software Environment*) von Xilinx die Platzierung und Verdrahtung für jedes einzelne FPGA statt. Zuerst wird dabei die EDIF-Netzliste mit dem Befehl `ngdbuild` (siehe Abb. 5-15) in ein Datenformat zur internen Weiterverarbeitung übersetzt (*Native Generic Database*, `*.ngd`). Mit dem Befehl `map` wird das erzeugte Datenformat in eine Netzliste umgewandelt, die der abgezielten FPGA-Technologie entspricht (*Native Circuit Description*, `*.ncd`). Der anschließende Befehl `par` führt dann die Plazierung, die Anordnung von Schaltungsteilen (*Floorplanning*) und die Verdrahtung durch. Danach wird mit dem Befehl `netgen` eine simulierbare Gatter-Netzliste (`*_parnet.v`) mit einer SDF-

Datei erzeugt (SDF = *Standard Delay Format*, `*_parnet.sdf`). Die SDF-Datei ermöglicht die Berücksichtigung der Leitungsverzögerungen, die durch die Platzierung und Verdrahtung entstanden sind (Post-Layout Simulation). Bei dieser Post-Layout-Simulation werden die Gatter-Modelle der Simprim-Bibliothek (*Simulation Library Primitives*) verwendet. Schließlich wird mit dem Befehl `bitgen` für jedes der 3 FPGAs eine binäre Bit-Datei generiert (`*.bit`), mit dem ein SRAM-basiertes FPGA konfiguriert werden kann. Die Bit-Dateien vom PPC2-, NTC- und NSC-FPGA müssen nun mit dem Befehl `promgen` zu einer PROM-Datei (PROM = *Programmable Read Only Memory*) im MCS86-Format von Intel (`*.mcs`) verkettet werden. Mit dem Werkzeug iMPACT (*intelligent Multi-Purpose And Configuration Tool*) von Xilinx kann mit dieser PROM-Datei die Konfiguration der 3 FPGAs auf der Emulationsplattform erfolgen. Die Skripte und notwendigen Befehle für die Platzierung und Verdrahtung sowie der Konfiguration der FPGAs sind im Anhang H auf Seite 173 und im Anhang I auf Seite 175 zusammengefasst.

SW-Entwurf

Der Entwurf der Software-Module erfolgt über die Programmiersprache C. Der entstandene Software-Code (`*.c`) wird mit Hilfe eines Compilers in einen Maschinen-Code übersetzt, und es entsteht eine Objekt-Datei (`*.o`). Bei dieser Übersetzung wird vom Compiler der Software-Code zunächst in einen Zwischen-Code umgewandelt, der dann von einem impliziten Assembler in den Maschinen-Code übersetzt wird. Der Maschinen-Code repräsentiert einen Befehlssatz in binärer Darstellung, der nur von einem bestimmten Prozessor ausgeführt werden kann. Bei der Implementierung von SEE wurde der GNU C-Compiler für den PPC (`powerpc-eabi-gcc`) verwendet, der die Compilierung nach dem EABI-Standard (EABI = *Embedded Application Binary Interface*) durchführt. Durch die Verwendung des EABI-Standards wird versucht, die Speichernutzung für den Maschinen-Code zu reduzieren und die Befehlsausführungszeiten zu optimieren. Bei dem verwendeten Compiler handelt es sich um einen Cross-Compiler. Ein Cross-Compiler ist ein Compiler, der auf einer Computerplattform läuft und Objekt-Dateien für eine andere Computerplattform erzeugt. Die vom Compiler für jedes Software-Modul generierten Objekt-Dateien werden mittels eines Linkers (`powerpc-eabi-ld`) verknüpft und zu einer ELF-Datei zusammengefasst (ELF = *Executable and Linkable Format*). Bei dieser Verknüpfung wird zusätzlich eine Datei benötigt, die das Speicherabbild (*Memory Map*) beschreibt, damit der Linker die einzelnen Software-Module bestimmten Speicherbereichen mit festgelegter Speichergröße zuordnen kann (siehe auch Makefile zum Compilieren/Linken der Software im Anhang J auf Seite 175). Die nun entstandene ELF-Datei kann mit dem Werkzeug ISE von Xilinx und dem Befehl `data2mem` in BRAM-Speicher der FPGAs geladen werden. Es entstehen dabei Dateien mit Hexadezimal-Werten im Text-Format (`*.mem`) oder im VHDL-Format (`*.vhd`), die die BRAMs vor dem Simulationsstart initialisieren. Bei der Platzierung und Verdrahtung der FPGAs wird die ELF-Datei als UCF-Datei (UCF = *User Constraint Format*, `*.ucf`) bei der Netzlistenumwandlung mit berücksichtigt und in die Bit-Datei mit integriert (siehe Abb. 5-15).

Debugging

Die Fehlersuche und -behebung (*Debugging*) auf der Emulationsplattform kann über die beiden Schnittstellen JTAG und RS232 erfolgen. Über die JTAG-Schnittstelle lässt sich ein Debugger (`powerpc-eabi-gdb`) an jeden PPC anschließen, mit dem man Unterbrechungspunkte (*Breakpoints*) im Software-Code definieren kann. Weiterhin ermöglicht der Debugger die Anzeige von Registerwerten des PPC-Prozessors und von Daten in den SRAM- und SDRAM-Speichern. Bevor man jedoch einen Debugger mit einem PPC verbinden kann, muss eine Initialisierung mit dem Werkzeug XMD (*Xilinx Microprocessor Debugger*) erfolgen, welches Bestandteil des Softwarepakets EDK (*Embedded Development Kit*) von Xilinx ist. Über die RS232-Schnittstelle können in Software-Module integrierte Testprozeduren, wie die Speicherinitialisierung oder das Lesen und Schreiben von Registern der Hardware-Module, von den PPC-Prozessoren ausgeführt werden. Dies ermöglicht durchführbare Tests von einzelnen Hardware-Modulen auf einer Prototypen-Platine, die anhand von Software durch Registerbeschreibung von Hardware-Modulen gesteuert werden können.

5.4 Realisierung der Prototypen-Platine

Das Architekturkonzept von SEE, vorgestellt im Kapitel 4.1.4 auf Seite 67, wurde durch die Herstellung einer Prototypen-Platine (PCB = *Printed Circuit Board*) realisiert. Der Entwurf des Schaltplans der Prototypen-Platine wurde mit dem Softwarepaket Eagle von CadSoft durchgeführt (siehe auch Anhang K auf Seite 178). Die Entflechtung und Bestükkung sowie die Herstellung der Prototypen-Platine wurde von den Firmen CAD-UL GmbH und Rinde GmbH vollzogen. Über das Architekturkonzept von SEE hinaus müssen beim Schaltplanentwurf weitere Überlegungen bezüglich unterstützender Testsignalanschlüsse, anvisierter Operationsfrequenz und zu erwartender Gesamtverlustleistung in Betracht gezogen werden.

Testsignalanschlüsse

Testsignalanschlüsse sind für das Debuggen auf der Prototypen-Platine eine große Unterstützung. Anhand dieser Anschlüsse können die Signalpegel von unterschiedlichen internen Steuer- oder Taktsignalen der FPGAs durch einen Logic Analyzer visualisiert werden. Zusätzlich können diese Anschlüsse genutzt werden, um weitere externe Speicher, Schnittstellen oder Bauelemente, wie LEDs, anzusteuern, die sich auf einer Zusatzplatine befinden. Für das PPC2- und das NSC-FPGA wurden aufgrund der begrenzten Verfügbarkeit an nutzbaren IO-Pins und aus Platzgründen auf der Platine jeweils 10 Testsignalanschlüsse vorgesehen. Das NTC-FPGA stellt allerdings insgesamt 2×48 = 96 Testsignalanschlüsse zur Verfügung.

Operationsfrequenz

Die anvisierte Operationsfrequenz auf der Prototypen-Platine ist ausschlaggebend für die Auswahl der externen SRAM- und SDRAM-Speicher. Alle ausgewählten externen Speicher auf der Platine haben Zugriffszeiten, die kleiner als 10 ns sind, und können somit bei einer maximalen Operationsfrequenz von 100 MHz angesteuert werden. Bei den beiden verwendeten FPGA-Typen kann, wie in Kapitel 5.1.5 auf Seite 110 bereits erwähnt, zwischen 3 unterschiedlichen Geschwindigkeitsklassen ausgewählt werden.

Das Virtex-II FPGA, welches sowohl das NTC- als auch das NSC-FPGA realisieren soll, stellt in einer 0,15 µm-Technologie 8 Metall-Lagen und High-Speed Transistoren mit einer Kanallänge von 0,12 µm zur Verfügung. Es werden die Speedgrades -6, -5 und -4 unterschieden. Beim leistungsfähigsten Speedgrade von -6 betragen die Verzögerungszeiten von internen Registern zu IO-Pins 3,33 ns, und beim leistungsschwächsten Speedgrade von -4 liegen diese Verzögerungszeiten bei 4,22 ns ($T_{IOPICKD}$ bzw. T_{IOPLID} in [Xili05c]). Für den Speedgrade -6 beträgt die maximale Operationsfrequenz für einen 64-Bit-Addierer 114 MHz. Die Verzögerungszeit einer einstufigen synchronen Logik, die den Signalpfad von einem Register über eine LUT zu einem anderen Register darstellt, wird mit 1,63 ns (613 MHz) festgestellt [Xili05c].

Das Virtex-II Pro FPGA, welches als PPC2-FPGA vorgesehen ist, verwendet in einer 0,13 µm-Technologie 9 Kupfer-Lagen und High-Speed Transistoren mit einer Kanallänge von 90 nm. Es werden bei diesem FPGA die Speedgrades -7, -6 und -5 unterschieden. Beim leistungsfähigsten Speedgrade von -7 kann der PPC mit maximal 400 MHz betrieben werden, und die Verzögerungszeiten vom internen Register zum IO-Pin liegen bei 2,65 ns. Im Vergleich dazu kann beim leistungsschwächsten Speedgrade von -5 der PPC bei maximal 300 MHz betrieben werden, und die Verzögerungszeiten zu den IO-Pins betragen 3,42 ns. In [Xili05b] wird die maximale Operationsfrequenz eines 64-Bit-Addierers mit 202 MHz und die Verzögerungszeit einer einstufigen synchronen Logik mit 956 ps (1046 MHz) für den Speedgrade -6 angegeben.

Aus Kostengesichtspunkten musste bei der Auswahl der FPGAs ein Kompromiss zwischen Operationsgeschwindigkeit und Integrationsdichte gefunden werden. Auf Kosten einer höheren Integrationsdichte wurde die anvisierte Operationsfrequenz auf 50 MHz festgesetzt, um FPGAs mit dem leistungsschwächsten Speedgrade einsetzen zu können. Als Virtex-II Pro FPGA wurde der Typ XCVP20-5FF1152 mit dem Speedgrade -5 und als Virtex-II FPGA der Typ XCV4000-4FF1152 mit dem Speedgrade -4, der einer Integrationsdichte von 4 M Gattern entsprechen soll, ausgewählt.

Gesamtverlustleistung

Die Festsetzung der maximalen Operationsfrequenz lässt die Abschätzung der zu erwartenden Gesamtverlustleistung auf der Prototypen-Platine zu. Diese Abschätzung ist notwendig, um sicherzustellen, dass alle Bauteile auf der Platine mit adäquaten Spannungen und vor allem ausreichenden Strömen versorgt werden. Die dafür notwendigen Bauteile sind Spannungsregler und DC/DC-Converter, die bestimmte Versorgungsspannungen bei einem maximalen Versorgungsstrom generieren, und beim Schaltplanentwurf mit berücksichtigt werden müssen. Die genaue Auswahl dieser Bauteile kann nur über eine detailliertere Verlustleistungsabschätzung erfolgen.

5.4.1 Verlustleistungsabschätzung

Die maximalen Werte für statische und dynamische Verlustleistung, die beim Leerlauf- bzw. Lastbetrieb entstehen, sind bei allen diskreten Bauteilen, wie SRAM- und SDRAM-Speicher, den entsprechenden Datenblättern zu entnehmen (siehe [Hell05a]). Die Verlustleistungswerte der 3 FPGAs, die bei SEE zum Einsatz kommen, müssen anhand der anvisierten Operationsfrequenz, der genutzten FPGA-Ressourcen und des voraussichtlichen Datentransfers über die IO-Pins abgeschätzt werden. Für diese Verlustleistungsabschätzung kann das Werkzeug Web Power Tool von Xilinx verwendet werden [Xili05b] [Xili05c].

Verlustleistung der FPGAs

Das PPC2-FPGA verfügt neben den beiden PPC-Prozessoren, über 88 18x18-Multiplizierer, 88 18 KBit BRAMs, 564 IO-Pins und 2320 CLBs, die insgesamt 9280 Slices vereinen. Die Taktfrequenz der PPCs f_{PPC} beträgt standardmäßig das Dreifache der Operationsfrequenz. Die Verlustleistungsabschätzung des PPC2-FPGAs ist in Tab. 5-7 zusammengefasst.

Quellen	Ressourcen	Toggle-Rate	V_{DDint}	V_{DDo}	V_{DDaux}
Ruhezustand	-	-	450 mW	-	417 mW
CLB Logik	4640 (50 %) von 9280 Slices	50 %	1146 mW	-	-
BRAM	70 (80 %) von 88	50 %	1048 mW	-	-
Multiplizierer	22 (25 %) von 88	Medium	182 mW	-	-
Prozessoren	2 von 2 ($f_{PPC} = 3 \times f_{SYS}$)	-	321 mW	-	-
IO-Pins	89 (RS232, USB, Flash) 205 (SRAMs, SDRAM) 196 (NTC-FPGA, NSC-FPGA)	20 % 50 % 80 %	31 mW	1128 mW	-
Verlustleistungsabschätzung vom **PPC2-FPGA**:		**4723 mW**	= 3178 mW	+ 1128 mW	+ 417 mW

Tab. 5-7: Verlustleistung des PPC2-FPGAs (f_{SYS} = 50 MHz, V_{DDint} = 1,5 V, V_{DDo} = 3,3 V, V_{DDaux} = 2,5 V)

Die Toggle-Rate beschreibt die Schaltrate der internen Flip-Flops. Die Lastkapazität, die bei der Ansteuerung von externen Bauteilen umgeladen werden muss, wird für jeden IO-Pin mit 10 pF angenommen. Die interne Logik des FPGAs benötigt die Versorgungsspannung V_{DDint} von 1,5 V und die IO-Pins werden mit der separaten Spannung V_{DDo} von 3,3 V beliefert. Hinzu kommt noch eine Hilfsspannung V_{DDaux} von 2,5 V, die das FPGA im Ruhezustand versorgt. Die Summe der Verlustleistungen, die an den Hardware-Modulen dieser 3 Versorgungsspannungen auftreten, ergibt die abgeschätzte Gesamtverlustleistung. Diese beläuft sich beim PPC2-FPGA auf etwa 4,7 W.

Das NTC- und das NSC-FPGA stellen jeweils 120 18x18-Multiplizierer, 120 18 KBit BRAMs, 824 IO-Pins und 5760 CLBs (23040 Slices) zur Verfügung. Die Verlustleistungsabschätzungen der beiden FPGAs sind in Tab. 5-8 und in Tab. 5-9 dargelegt.

Quellen	Ressourcen	Toggle-Rate	V_{DDint}	V_{DDo}	V_{DDaux}
Ruhezustand	-	-	338 mW	-	66 mW
CLB Logik	18432 (80 %) von 23040 Slices	50 %	3609 mW	-	-
BRAM	48 (40 %) von 120	50 %	1049 mW	-	-

Tab. 5-8: Verlustleistung des NTC-FPGAs (f_{SYS} = 50 MHz, V_{DDint} = 1,5 V, V_{DDo} = 3,3 V, V_{DDaux} = 3,3 V)

Quellen	Ressourcen	Toggle-Rate	V_{DDint}	V_{DDo}	V_{DDaux}
Multiplizierer	48 (40 %) von 120	Medium	452 mW	-	-
IO-Pins	55 (SDRAM) 106 (SRAMs) 196 (PPC2-FPGA, NSC-FPGA)	20 % 80 % 80 %	46 mW	958 mW	-
Verlustleistungsabschätzung vom **NTC-FPGA**:		**6518 mW**	= 5494 mW	+ 958 mW	+ 66 mW

Tab. 5-8: Verlustleistung des NTC-FPGAs (f_{SYS} = 50 MHz, V_{DDint} = 1,5 V, V_{DDo} = 3,3 V, V_{DDaux} = 3,3 V)

Bei beiden FPGAs ist die Spannung V_{DDaux} im Gegensatz zum PPC2-FPGA mit der Versorgungsspannung der IO-Pins identisch und beträgt 3,3 V. Die abgeschätzte Gesamtverlustleistung vom NTC-FPGA liegt bei 6,5 W und beim NSC-FPGA wird ein Verlustleistung von über 8,3 W erreicht, die vor allem aus dem ständigen Datentransfer zum Parameterspeicher resultiert.

Quellen	Ressourcen	Toggle-Rate	V_{DDint}	V_{DDo}	V_{DDaux}
Ruhezustand	-	-	338 mW	-	66 mW
CLB Logik	23040 (100 %) von 23040 Slices	50 %	3718 mW	-	-
BRAM	96 (80 %) von 120	50 %	1573 mW	-	-
Multiplizierer	96 (80 %) von 120	Medium	902 mW	-	-
IO-Pins	339 (SDRAMs) 196 (PPC2-FPGA, NTC-FPGA)	80 % 80 %	73 mW	1657 mW	-
Verlustleistungsabschätzung vom **NSC-FPGA**:		**8327 mW**	= 6604 mW	+ 1657 mW	+ 66 mW

Tab. 5-9: Verlustleistung des NSC-FPGAs (f_{SYS} = 50 MHz, V_{DDint} = 1,5 V, V_{DDo} = 3,3 V, V_{DDaux} = 3,3 V)

Versorgungsströme

Aus den Verlustleistungsabschätzungen aller Bauteile auf der Prototypen-Platine lassen sich nun die maximal notwendigen Versorgungsströme, die bei unterschiedlichen Versorgungsspannungen bereitgestellt werden müssen, feststellen (siehe Tab. 5-10).

Anzahl	Beschreibung	$I_{DD\ 1,5V}$	$I_{DD\ 1,8V}$	$I_{DD\ 2,5V}$	$I_{DD\ 3,3V}$
1	PPC2-FPGA	2118 mA	-	250 mA	341 mA
1	NTC-FPGA	3662 mA	-	-	310 mA
1	NSC-FPGA	4402 mA	-	-	522 mA
9	SDRAMs	-	-	-	845 mA
3	SDRAM DIMMs	-	-	-	29160 mA
6	SRAMs	-	-	-	1280 mA
1	Flash	-	-	-	30 mA
1	RS232 Transceiver	-	-	-	64 mA
1	USB Transceiver	-	30 mA	-	39 mA
2	Quarzoszillatoren	-	-	-	24 mA
Notwendige Versorgungsströme:		**10,18 A**	**0,03 A**	**0,25 A**	**32,615 A**

Tab. 5-10: Notwendige Versorgungsströme der Prototypen-Platine

Der hohe Versorgungsstromwert von 29,16 A für die 3 SDRAM-DIMM-Speicher ergibt sich aus dem Datenblatt für 1 GB große SDRAM-DIMM-Speicher [Hell05a]. Diese Speicher-Module bestehen aus 16 SDRAM-Standardbausteinen und benötigen jeweils für die Steuerung des “Refresh”-Vorgangs einen maximalen Strom von 9,72 A (*Auto Refresh Current*). Auf der Prototypen-Platine werden für die beiden Versorgungsspannungen 1,8 V und 2,5 V Spannungsregler vorgesehen, die einen maximalen Ausgangsstrom von 1,5 A liefern. Für die beiden Versorgungsspannungen 1,5 V und 3,3 V kommen DC/DC-Converter zum Einsatz, die bei einer Eingangsspannung von 5 V einen maximalen Ausgangsstrom von 15 A bereitstellen können. Die 3 SDRAM-DIMM-Speicher werden von einem

externen PC-Netzteil versorgt, welches eine Spannung von 3,3 V mit einem maximalen Strom von 30 A liefert.

5.4.2 Eigenschaften

Die Eigenschaften der realisierten Prototypen-Platine sind in Tab. 5-11 zusammengefasst. Die Mehrlagen-Platine (*Multilayer Board*) verfügt über eine Größe von 25×15 cm und verwendet 10 Lagen, wovon jeweils 2 Lagen für Versorgungsspannungen und Massekontakte dienen und die restlichen 6 Lagen für Signalleitungen zur Verfügung stehen (siehe auch Anhang L auf Seite 184).

Eigenschaften	Kommentar
Größe / Dicke	250 × 150 mm / ca. 2 mm
Lagenanzahl	10 (6 Signal-, 2 Versorgungs- und 2 Masselagen)
Kleinster Leiterbahnabstand	120 µm
Kleinste Leiterbahnbreite	150 µm
Kleinster Bohrdurchmesser	250 µm
Anzahl der Bohrungen	> 6000

Tab. 5-11: Eigenschaften der Prototypen-Platine [Hell05a]

Anzahl der Bauelemente

Die Anzahl der integrierten Schaltkreise und passiven Bauelemente auf der Prototypen-Platine sind in Tab. 5-12 aufgelistet. Neben den 3 FPGAs werden 6 SDRAMs (inklusive der 3 DIMM-Speicher) und 12 SRAMs eingesetzt. Bei den passiven Bauteilen werden mehr als 350 Entkoppelkapazitäten verwendet, die notwendig sind, um bei Änderungen der Versorgungsströme die Versorgungsspannung stabil zu halten.

Anzahl	Beschreibung	Anzahl	Beschreibung
12	SRAM-Bausteine	1	Flash-Baustein
6	SDRAM-Bausteine (inkl. 3 DIMM Module)	1	RS232 Transceiver
3	FPGA-Bausteine	1	USB Transceiver
2	Quarzoszillatoren	357	Kapazitäten (0,01 µF…470 µF)
2	DC/DC-Converter	89	Widerstände (0 Ω…∞ Ω)
2	Spannungsregler	2	Induktivitäten

Tab. 5-12: ICs und passive Bauelemente auf der Prototypen-Platine (modifiziert von [Hell05a])

Die realisierte Prototypen-Platine ist in Abb. 5-16 dargestellt und wird von einem externen PC-Netzteil mit der Spannung 5 V (links oben auf der Platine) und der Spannung 3,3 V, die nur für die 3 SDRAM-DIMM-Speicher bestimmt ist (rechts oben auf der Platine), beliefert. Auf der Platine selber müssen insgesamt 5 verschiedene Versorgungsspannungen bereitgestellt werden. Die Spannung von 1,5 V ist für die interne Versorgung aller 3 FPGAs notwendig, die Spannung von 1,8 V wird nur vom USB Transceiver gebraucht, die Spannung von 2,5 V ist lediglich für das PPC2-FPGA nötig, die Spannung von 3,3 V versorgt alle aktiven Bauelemente und mit der Spannung von 5 V muss schließlich noch der USB-Stecker ausgestattet werden. Die beiden Spannungsregler für 1,8 V und 2,5 V sind links oben nebeneinander platziert, und die beiden DC/DC-Converter zur Generierung der Spannungen 1,5 V und 3,3 V sind auf der linken Seite der Platine übereinander angeordnet.

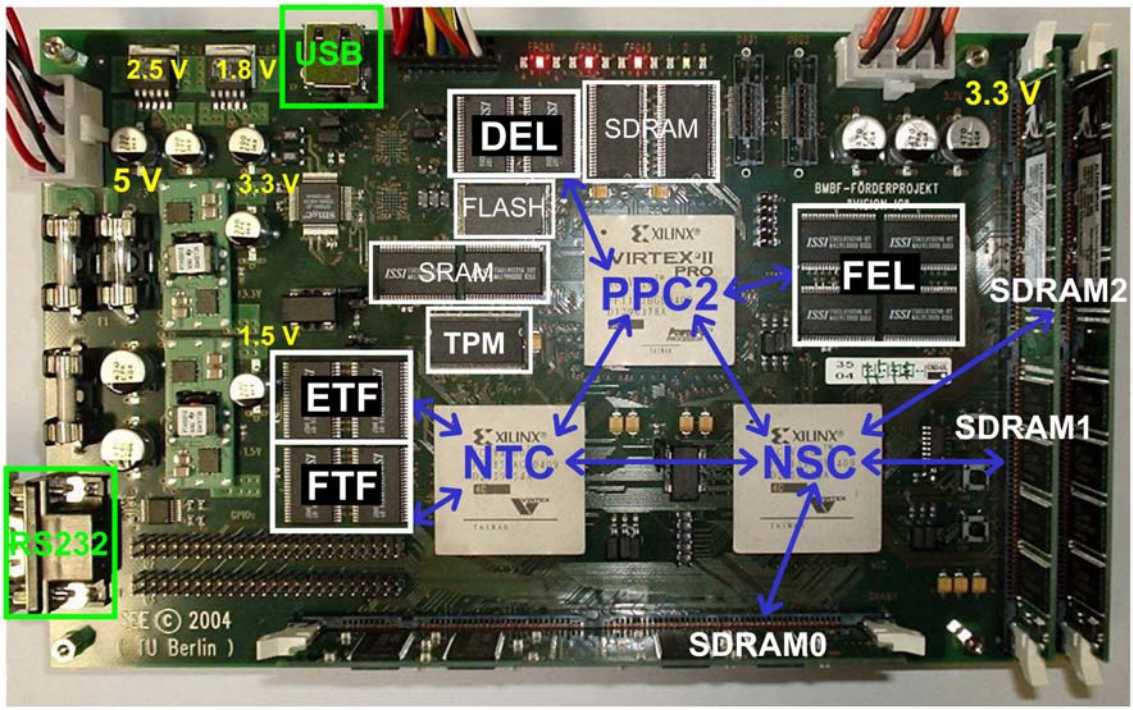


Abb. 5-16: Prototypen-Platine der digitalen Emulationsplattform [Hell04c]

Anordnung der Bauelemente

Die 3 FPGAs sind, analog zum Blockdiagramm in Abb. 4-6 auf Seite 68, in einem Dreieck angeordnet, um möglichst kurze Leitungswege zu den angeschlossenen Speicherbausteinen und untereinander zu haben. Das PPC2-FPGA ist mit der DEL, bestehend aus 2 SRAM-Bausteinen, und der FEL (4 SRAM-Bausteine) verbunden. Neben einem Flash-Baustein sind noch ein SRAM- und ein SDRAM-Speicher, die sich aus jeweils 2 Speicherbausteinen zusammensetzen, an das PPC2-FPGA angeschlossen. Das NTC-FPGA ist mit dem ETF und dem FTF, die jeweils auf 2 SRAM-Bausteinen basieren, verknüpft und ist zusätzlich mit dem TPM in Form eines SDRAM-Bausteins verbunden. Schließlich stellt das NSC-FPGA die Verbindungen zu den 3 SDRAM-DIMM-Speichern zur Verfügung. Die jeweiligen 10 Testanschlüsse für das PPC2-FPGA bzw. für das NSC-FPGA befinden sich rechts oben bzw. links unten neben dem FPGA. Die Testanschlüsse des NTC-FPGAs sind unterhalb des FTFs platziert und in 2 Reihen à 50 Pins angeordnet. Der erste und letzte Pin einer Reihe ist jeweils mit 3,3 V bzw. Masse verbunden, so dass insgesamt 2×48 Testanschlüsse zur Verfügung stehen.

Nachdem die Realisierung der Prototypen-Platine vorgestellt wurde, wird im folgenden dargestellt, wie die Inbetriebnahme von SEE für eine PCNN Emulation erfolgen muss.

5.5 Durchführung der Emulation von pulscodierten neuronalen Netzen

Vor der Durchführung einer PCNN Emulation müssen 2 Bearbeitungsvorgänge erfolgen: die Netzwerkbeschreibung und die Netzwerkkonfiguration. Bei der Netzwerkbeschreibung wird der Architekturtyp, der Eingangsreiz in Form eines Eingangsbildes sowie die Anzahl der Netzwerkschichten (Anzahl der Neuronen) und die Verbindungsstruktur (Anzahl der Synapsen) definiert. Die Netzwerkkonfiguration beinhaltet die Konfiguration der FPGAs und die Übertragung der erstellten Netzwerkbeschreibung über eine serielle Schnittstelle auf die Emulationsplattform. Nachdem die PCNN Emulation durchgeführt wurde, werden die aufgetretenen Pulsereignisse von der Emulationsplattform zum Host-Rechner zurückübertragen. Anschließend erfolgt eine Visualisierung der Pulsereignisse in Form von statischen und dynamischen Pulsmustern.

5.5.1 Netzwerkbeschreibung

Die Netzwerkbeschreibung erfolgt im Rahmen einer graphischen Benutzeroberfläche (GUI), die mit der Skriptsprache Perl/Tk entwickelt wurde. Das Hauptmenü des GUIs ist in Abb. 5-17a gezeigt. Das GUI stellt die 7 Submenüs *File*, *Init*, *View*, *Data*, *Run*, *Generate* und *Show* zur Verfügung (siehe Abb. 5-17b), deren Reihenfolge dem sequentiellen Ablauf zur Durchführung einer PCNN Emulation entspricht. Eine ausführliche Beschreibung des GUIs sind den Arbeiten in [Mahr07] zu entnehmen.

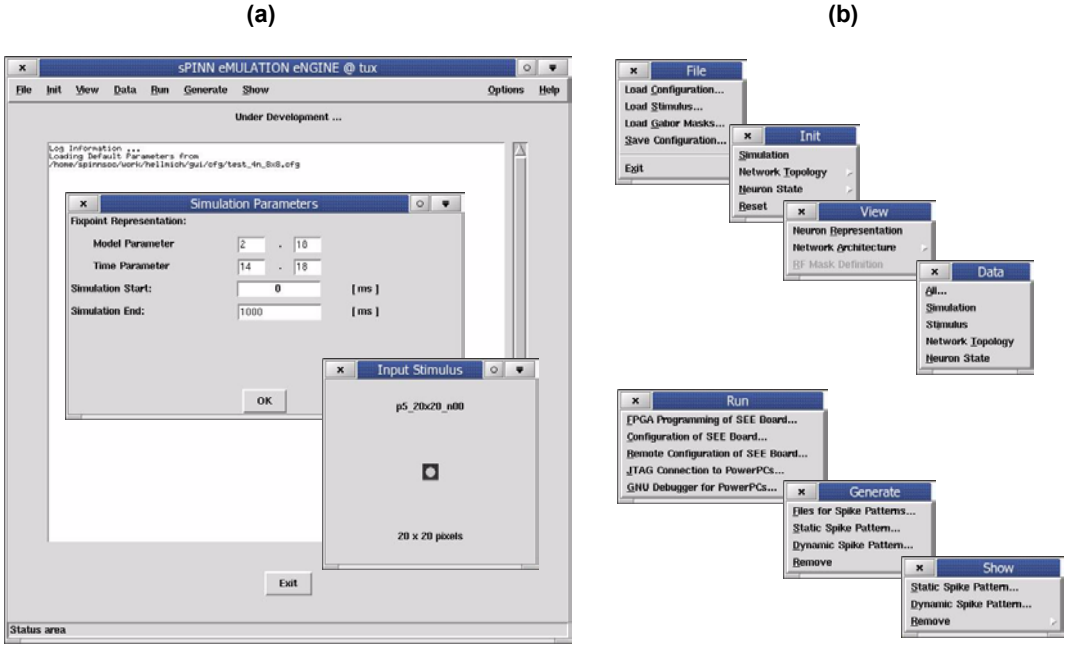


Abb. 5-17: Graphische Benutzeroberfläche (GUI) für die Durchführung der PCNN Emulation: (a) Hauptmenü, (b) Submenüs [Hell05b]

Mit dem *File*-Submenü werden Eingangsbilder geladen, die in Eingangsreize umgewandelt werden. Als Eingangsbilder werden nur Bilder mit 256 Graustufen berücksichtigt, deren Pixelwerte bei der Umwandlung in Eingangsreize auf Werte zwischen 0 und 1 normiert werden. Für Gabor-Architekturen lassen sich die notwendigen Koordinatenbeschreibungen von RF-Masken laden. Zusätzlich besteht die Möglichkeit, vollständige Netzwerkbeschreibungen abzuspeichern (*Save Configuration*) bzw. bereits erstellte Netzwerkbeschreibungen für die Initialisierung einer neuen Netzwerkbeschreibung zu verwenden (*Load Configuration*).

Initialisierung

Die Initialisierung einer Netzwerkbeschreibung (*Init*-Submenü) besteht aus der Definition von Parametern für die Emulation, für die Netzwerktopologie- und für die Neuronenzustandsberechnung. Mit den Emulationsparametern werden binäre Vor- und Nachkommastellen für die Modell- und Zeitparameter und auch die Netzwerkzeit für das Ende der Emulation festgelegt. Die Netzwerktopologieparameter sind je nach ausgewähltem Netzwerkarchitekturtyp (Kaskaden- oder Gabor-Architektur) unterschiedlich. Bei der Kaskaden-Architektur werden, nachdem die Anzahl der Netzwerkschichten festgelegt wurde, für jede Netzwerkschicht die Anzahl der lateralen und externen Netzwerkverbindungen sowie der Schichttyp (Eingangs-, Zwischen- oder Ausgangsschicht) definiert. Bei der Gabor-Architektur wird zunächst die Anzahl der Gabor-Detektoren bestimmt, woraus sich dann die Gesamtanzahl der Netzwerkschichten ergibt. Jedem Gabor-Detektor müssen dann zuvor geladene RF-Masken zugewiesen und die Dichteverteilung der RF-Masken auf der Eingangsschicht festgelegt werden. Dann ergeben sich die notwendigen Parameter wie Anzahl der RFs, der Gradientenpaare, der Neuronen und der Synapsen pro Gabor-Detektor. Die Neuronenzustandsparameter definieren die Anfangszustände der Neuronen pro Netzwerkschicht, indem die Pulsdauer, der Schwellwert und der Initialwert des Membranpotentials bestimmt werden. Der Initialwert des Membranpotentials

kann neben einem festgelegten Wert (sinnvollerweise nicht größer als der Schwellwert) auch ein Zufallswert sein, so dass alle Neuronen der Netzwerkschicht mit zufällig vergebenen Membranpotentialwerten versehen werden. Für jede bei der Definition der Netzwerktopologieparameter angelegte Netzwerkverbindung werden die Neuronenzustandsparameter um Parameter für die Synapsen ergänzt. Mit diesen Parametern werden die Initialwerte für die Synapsen festgelegt und gegebenenfalls eine Adaptionsregel für die Synapsen ausgewählt. Bei adaptiven Synapsen müssen dann Werte für Verstärkungsfaktor, Abklingkonstante und eventuellen Schwellstrom eingegeben werden. Die Parameter für die Netzwerktopologie- und Neuronenzustandsberechnung werden in Netzwerkschicht-bezogenen ein- oder mehrdimensionalen Arrays abgelegt. Arrays sind dadurch gekennzeichnet, dass die Elemente über einen Index (eindimensional) oder mehrere Indizes (mehrdimensional) angesprochen werden. Der erste Index bei diesen Parameter-Arrays entspricht dabei immer der Netzwerkschichtnummer der Netzwerkbeschreibung.

Darstellung

Die initialisierte Netzwerkbeschreibung kann im *View*-Submenü als gesamte Netzwerkarchitektur dargestellt werden. Dabei wird die Gesamtanzahl an Neuronen und Synapsen des PCNNs auch als Prozentwert der maximal emulierbaren Neuronen und Synapsen angegeben, um einschätzen zu können, welche Ressourcen von SEE für die definierte Netzwerkbeschreibung verwendet werden. Zusätzlich kann zu jeder Netzwerkschicht die Anzahl an afferenten Synapsen mit Quellschicht und die Anzahl an efferenten Synapsen mit Zielschicht dargestellt werden, um die Verbindungsstrukturen des Gesamtnetzwerks zu überprüfen.

Generierung

Nachdem die PCNN Größe einschließlich der Verbindungsstrukturen überprüft worden sind, wird die Generierung der Konfigurationsdateien durchgeführt (*Generate*-Submenü). Diese Generierung führt die Umwandlung der Array-Darstellung der Parameter für die Emulation, die Netzwerktopologie- und Neuronenzustandsberechnung in eine Repräsentation von Hexadezimal-Werten durch, die auf mehrere Konfigurationsdateien in Form von Textdateien verteilt werden. Für die PCNN Emulation werden 5 verschiedene Konfigurationsdateien generiert: 1.) Emulationsparameter (`data_sim.hex`), 2.) Netzwerktopologieparameter (`data_ntc.hex`), 3.) Neuronenzustandsparameter (`data_nsc.hex`), 4.) Eingangsschichtneuronen nach der Neuronencodierung in Abb. 4-20 auf Seite 86, die vor dem Emulationsstart in die DEL eingetragen werden müssen (`data_del.hex`), und 5.) Eingangsreize für jedes Eingangsschichtneuron (`data_stm.hex`).

Nach der Generierung dieser Konfigurationsdateien ist die Netzwerkbeschreibung abgeschlossen, und die Netzwerkkonfiguration kann mit dem *Run*-Submenü gestartet werden.

5.5.2 Netzwerkkonfiguration

Die Netzwerkkonfiguration umfasst die Konfiguration der FPGAs (Übertragung der PROM-Datei) und die Konfiguration der Emulationsplattform (Übertragung der Konfigurationsdateien).

FPGA Konfiguration

Voraussetzung für die erfolgreiche Konfiguration der FPGAs ist, dass das externe PC-Netzteil eingeschaltet ist, um die Emulationsplattform mit den Versorgungsspannungen 3,3 V und 5 V zu beliefern, und dass die parallele Schnittstelle des Host-Rechners über ein FPGA-Konfigurationskabel mit der FPGA-Konfigurationsschnittstelle (*Slave Serial*) der Emulationsplattform verbunden ist. Mit der ersten Auswahloption des *Run*-Submenüs (*FPGA Programming of SEE Board*) kann nun die Konfiguration der FPGAs erfolgen. Wie im Kapitel 5.3 auf Seite 114 bereits erwähnt, wird das Werkzeug iMPACT aufgerufen, um die am Ende des Entwurfsfluss entstandene PROM-Datei seriell über eine Datenleitung des FPGA-Konfigurationskabels zu übertragen. Nach etwa 20 bis 25 Sekunden ist dieser Konfigurationsvorgang abgeschlossen, und die Hardware in den FPGAs ist betriebsbereit. Nach dem Druck auf die Hardware-Reset-Taste der Emulationsplattform wird sicherge-

stellt, dass alle Register der Hardware-Module einen definierten Ausgangswert und die Taktsignale für alle 3 FPGAs die gleiche Phase haben.

SEE Konfiguration

Nun kann mit der zweiten Auswahloption des *Run*-Submenüs (*Configuration of SEE Board*) die Übertragung der Konfigurationsdateien erfolgen, wenn ein 9-poliges Nullmodem-Kabel zwischen der seriellen Schnittstelle des Host-Rechners und der RS232-Schnittstelle von SEE angeschlossen ist. Hierbei wird das Terminalprogramm MiniCOM aufgerufen, das empfangene ASCII-Zeichen der Emulationsplattform in einem Terminalfenster darstellen kann und Programmoptionen zur Verfügung stellt, um mit Hilfe des Befehls `ascii-xfr` Textdateien zu übertragen oder zu empfangen. Das Konfigurationsmenü, das von den Software-Modulen von SEE realisiert wird (siehe auch Kapitel 5.2 auf Seite 112), ist in Abb. 5-18a dargestellt und in [Rafa06] detailliert beschrieben.

(a) **(b)**

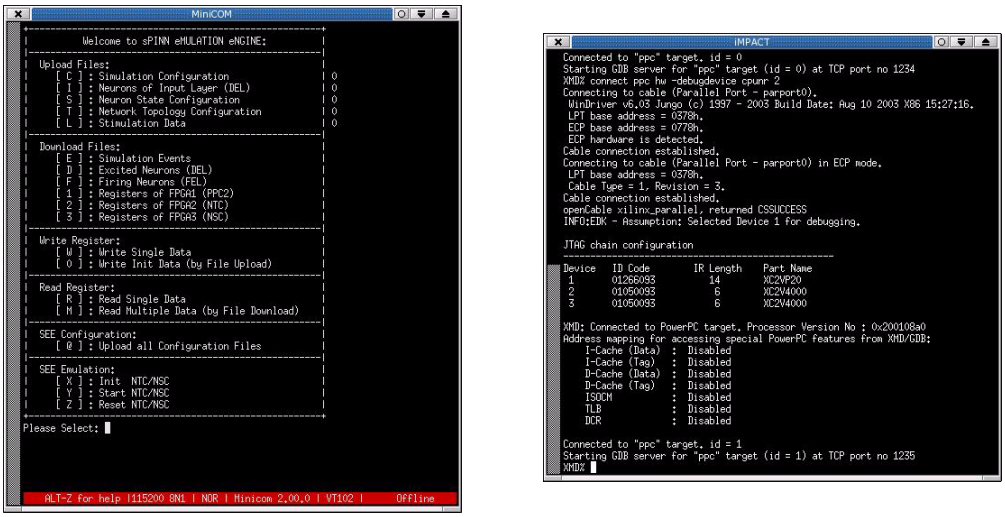


Abb. 5-18: (a) SEE Konfigurationsmenü, (b) JTAG-Verbindung zu Debugzwecken beider PowerPCs

Für die Überprüfung, ob die beiden PPCs im PPC2-FPGA funktionsfähig sind, lässt sich zu jedem einzelnen Prozessor eine JTAG-Verbindung herstellen (siehe Abb. 5-18b). Hierfür muss die parallele Schnittstelle des Host-Rechners über ein FPGA-Konfigurationskabel mit der JTAG-Schnittstelle der Emulationsplattform verbunden sein.

Mit der Auswahloption `@` im Konfigurationsmenü werden sämtliche 5 Konfigurationsdateien hintereinander auf die Emulationsplattform übertragen. Die Konfigurationsdatei mit den Eingangsschichtneuronen wird dabei von der Software direkt in die DEL geschrieben. Die Hexadezimal-Werte der restlichen 4 Konfigurationsdateien werden jeweils in eine der insgesamt 4 Speicherbänke des SDRAM-Speichers transferiert. Mit der Auswahloption `X` werden das NTC- und das NSC-FPGA initialisiert, indem anhand der übertragenen Netzwerktopologie- und Neuronenzustandsparameter die internen Hardware-Module initialisiert werden. Zusätzlich werden bei der Initialisierung des NTC-FPGAs das FTF und ETF mit Nullen initialisiert, und beim NSC-FPGA erfolgt die Übertragung der NIBs, die sich aus den Neuronenzustandsparametern ergeben, für jedes Neuron in die SDRAM-DIMM-Speicher. Mit der Auswahloption `Y` lässt sich nach erfolgreicher Initialisierung von NTC- und NSC-FPGA die PCNN Emulation starten. Es besteht die Möglichkeit einen Einzelschrittbetrieb zu aktivieren, um den Emulationsablauf zu Testzwecken in Teilschritten durchführen zu können. Durch Tastendruck lassen sich einzelne Emulationsphasen (Netzwerktopologie- bzw. Neuronenzustandsphase) oder einzelne Zustände innerhalb der Emulationsphasen abarbeiten.

Mit der Auswahloption zum Starten der PCNN Emulation wird dann die Emulationsplattform eigentlich erst in Betrieb genommen.

5.5.3 Inbetriebnahme der Emulationsplattform

In Abb. 5-19 ist die Emulationsplattform im Betrieb dargestellt. Oben links ist das externe PC-Netzteil zu sehen, welches die Emulationsplattform mit 2 unterschiedlichen Spannungen versorgt. Einige Testanschlüsse der 3 FPGAs sind über Verbindungskabel mit dem Logic Analyzer verbunden, der sich rechts neben dem PC-Netzteil befindet. Der eingesetzte Logic Analyzer von Agilent bietet die Möglichkeit, sich bis zu 16 digitale Signalverläufe anzeigen zu lassen. Eine Zusatzplatine mit insgesamt 32 LEDs ist durch ein Flachbandkabel mit den Testanschlüssen des NTC-FPGAs verbunden. Diese Zusatzplatine befindet sich in Abb. 5-19 unterhalb der Emulationsplattform und dient zur weiteren Visualisierung von Statusinformationen während der Emulation. Statusinformationen, die durch die LEDs angezeigt werden, sind beispielsweise abgeschlossene Übertragung einer Konfigurationsdatei, abgeschlossene Initialisierung eines FPGAs oder Anzeige der aktuellen Zustände innerhalb der Emulationsphasen. Links neben der Zusatzplatine befindet sich das Nullmodem-Kabel, welches den Host-Rechner, der durch einen tragbaren PC repräsentiert wird (in Abb. 5-19 rechts), mit der Emulationsplattform verbindet. Die beiden FPGA-Konfigurationskabel (*Parallel Cable IV* von Xilinx) für die Übertragung der PROM-Datei und für das Debuggen der Software-Module im JTAG-Modus befinden sich oberhalb der Emulationsplattform.

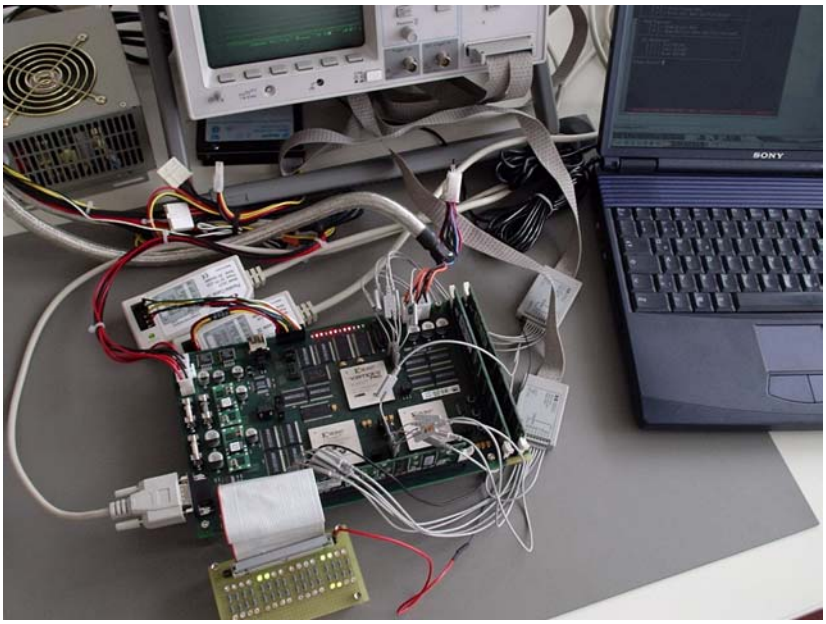


Abb. 5-19: Inbetriebnahme der Emulationsplattform

Nachdem die PCNN Emulation beendet ist, können die aufgetretenen Pulsereignisse, die in Form von Neuronennummer mit zugehöriger Emulationsereigniszeit im 32 MB großen SDRAM-Speicher des PPC2-FPGAs abgelegt wurden, zum Host-Rechner übertragen werden (Auswahloption `E` im Konfigurationsmenü). Mit dem *Generate*- und *Show*-Submenü (siehe Abb. 5-17b auf Seite 122) kann mit den übertragenen Pulsereignissen auf dem Host-Rechner eine Visualisierung erfolgen.

5.5.4 Visualisierung von Pulsereignissen

Die Visualisierung von Pulsereignissen erfolgt auf dem Host-Rechner mit dem Software-Paket Matlab von Mathworks, welches eine Skript-basierte Umwandlung der von der Emulationsplattform empfangenen Hexadezimalzahlen und eine anschließende Weiterverarbeitung für eine Visualisierung erlaubt. Die Pulsereignisse werden durch die Generierung von statischen und dynamischen Pulsmustern visualisiert. Bei der Generierung von statischen Pulsmustern, können bestimmte Neuronen ausgewählt werden, deren Pulszeitpunkte über einen bestimmten Netzwerkzeitraum in einer Bilddatei im GIF-Format

zusammengefasst werden (siehe Abb. 5-20a). Dadurch lässt sich feststellen, ob sich das synchrone Pulsverhalten bei bestimmten Neuronengruppen einstellt oder nicht. Bei der Generierung von dynamischen Pulsmustern werden die Pulsereignisse von sämtlichen Neuronen einer Netzwerkschicht herangezogen. Dabei werden die Neuronen in einem Koordinatensystem mit x- und y-Koordinaten dargestellt. Nachdem ein Netzwerkzeitraum festgelegt wurde, wird in Zeitabständen von 0,1 ms eine Sequenz von Bildern erzeugt, die nur Neuronen markiert, die sich im Pulszustand befinden. Somit werden Bildsequenzen im MPEG-Format erzeugt, die das Pulsverhalten aller Neuronen einer Netzwerkschicht widerspiegeln.

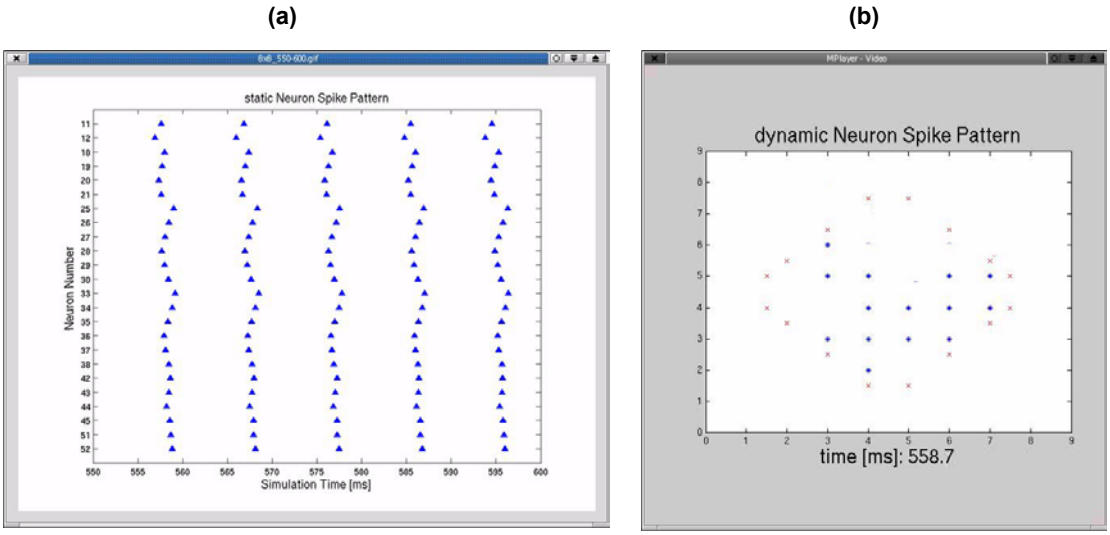


Abb. 5-20: Generierung von Pulsmustern: (a) statische Pulsmuster, (b) dynamische Pulsmuster

Resümee

Im Rahmen dieses fünften Kapitels wurde die Implementierung der Emulationsplattform SEE skizziert. Die Implementierung von Hardware- und Software-Modulen sowie der durchgeführte Hardware/Software-Entwurfsfluss wurden vorgestellt. Die Emulationsablaufsteuerung ist im PPC2-FPGA integriert und beinhaltet neben 2 Prozessoren jeweils 2 unabhängige Ansteuerungen zu den Ereignislisten DEL und FEL und zu den beiden FPGAs, in denen die Berechnungseinheiten für Netzwerktopologie und Neuronenzustand integriert sind (siehe Abb. 5-1 auf Seite 92). Die Neuronenzustandsberechnung ist im NSC-FPGA implementiert und besteht aus einer Kontrolleinheit, 3 Prozessoreinheiten und einer Topologie-Vektor-Empfangseinheit (siehe Abb. 5-2 auf Seite 96). Die Netzwerktopologieberechnung wird durch das NTC-FPGA realisiert und umfasst neben einer Kontrolleinheit, einen Topologie-Cache, eine Tag-Berechnungseinheit zur Umwandlung von Topologie-Vektoren und Neuronennummern, sowie Kontrolleinheiten für die Berechnung von Neuronenposition und -markierung innerhalb der beiden Markierungsspeicher ETF und FTF (siehe Abb. 5-6 auf Seite 102). Die Verifikation von Hardware-Modulen (RTL, Post-Synthese und Post-Layout Simulation) erfolgte, wie bei komplexen Digitalsystemen mittlerweile üblich, stets im Zusammenspiel mit einem Software-Code, der von CPU Modellen ausgeführt wird. Die realisierte Emulationsplattform in Form einer Prototypen-Platine wurde präsentiert und die notwendigen Überlegungen bezüglich der zu erwartenden Verlustleistungen für den Schaltplan-Entwurf dargelegt. Die Gesamtverlustleistung der 3 verwendeten FPGAs beträgt aufgerundet 20 W (siehe Tab. 5-7, Tab. 5-8 und Tab. 5-9 auf Seite 118). Schließlich wurde die Durchführung einer Emulation auf der Emulationsplattform SEE dargestellt. Hierfür wurde ein GUI entwickelt, welches die durchzuführenden Schritte der Netzwerkbeschreibung, Netzwerkkonfiguration, Inbetriebnahme der Prototypen-Platine und die Visualisierung von Pulsereignissen in einer leicht bedienbaren und übersichtlichen Benutzerumgebung vereint.

Im folgenden Kapitel wird eine Leistungsbewertung der realisierten Emulationsplattform vorgenommen, zu der durchgeführte PCNN Simulationen auf Einprozessor-Systemen und die Leistungsfähigkeit von existierenden Emulationsplattformen herangezogen werden.

KAPITEL

6

Leistungsbewertung von SEE

In diesem Kapitel erfolgt eine Leistungsbewertung von SEE anhand von durchgeführten Netzwerksimulationen. Als Vergleich dienen hierfür durchgeführte Netzwerksimulationen auf einem Einprozessor-System und Performanzabschätzungen anderer bereits in Kapitel 3.4 vorgestellter digitaler Simulationsplattformen. Bei den Leistungsbewertungen anderer Emulationsplattformen, die auf einem Halb-Kunden-Entwurf oder auf FPGAs basieren, wurde versucht, das Neuronenmodell mit adaptiven Synapsengewichten einzubeziehen. Anhand dieser Leistungsabschätzungen wird eine Bewertung für die Implementierung von SEE vorgenommen, in der ein Effizienzvergleich mit den anderen Emulationsplattformen durchgeführt wird. Schließlich erfolgt eine Evaluation des Neuronenmodells mit adaptiven Synapsengewichten, die zusammenfassend den Rechenaufwand dieses Neuronenmodells verdeutlichen soll.

6.1 Simulationen auf einem Einprozessor-System

Um die Emulationszeit von SEE abschätzen zu können, wurden mit einem Ad-hoc Simulator (siehe auch Kapitel 3.4.1 auf Seite 46), der mit der Programmiersprache C++ beschrieben ist und von den Autoren in [Heit02] bereitgestellt wurde, Software Simulationen auf einem Einprozessor-System durchgeführt. Der Ad-hoc Simulator verwendet die vorgeschlagenen SW-Prozeduren in [Nume05], um unterschiedliche numerische Integrationsverfahren oder Nullstellensuchverfahren zu verwenden. Der Aufruf des Simulators (`pnn see.in`) erfolgt stets in Verbindung mit einer Initialisierungsdatei, die Informationen wie Verbindungsstruktur, Anfangswerte für Membranpotentiale und Synapsengewichte, sowie Auswahl des Eingangsbildes enthält. Als Einprozessor-System diente ein PC mit dem Betriebssystem Linux, der mit einer 2,4 GHz getakteten Pentium-IV-CPU und einem 1 GB großen Arbeitsspeicher ausgestattet ist.

Software Profiling

An dem bereitgestellten Ad-hoc Simulator mussten noch einige Modifikationen vorgenommen werden, um dem parallelisierten Emulationsablauf bei SEE (siehe Kapitel 4.2.2 auf Seite 70) zu entsprechen. Diese Modifikationen ermöglichten zum einen die Funktionalität des Software Simulators mit der Implementierung von SEE zu vergleichen und zum anderen eine geeignete Profildarstellung der Software (*Software Profiling*) zu erhalten. Eine Profildarstellung der Software gibt Auskunft darüber, wieviel Rechenzeit einzelne SW-Prozeduren für die Ausführung benötigen (*Flat Profile*) oder wieviele Male einzelne SW-Prozeduren aufgerufen wurden (*Call Graph*). Um die Profildarstellung während der PCNN Simulation zu aktivieren, muss der Ad-hoc Simulator mit `g++ -pg` compiliert werden. Nach der Ausführung der PCNN Simulation entsteht eine Datei, die alle Profildaten enthält (`gmon.out`). Mit dem Befehl `gprof` ist es nun möglich mit Angabe des Ad-hoc Simulators im ausführbaren Format (*Executable*) und der Datei mit Profildaten, die Profildarstellung der Software zu erstellen (`gprof pnn gmon.out`).

Software Simulation

Bei den durchgeführten Netzwerksimulation auf dem Einprozessor-System wurden Eingangsschichten der Größen 10×10, 15×15, 20×20, 25×25 und 30×30 mit einer lateralen Vierer-Nachbarschaftsverbindung (4n), einer lateralen Achter-Nachbarschaftsverbindung (8n) und einer lateralen Vollvernetzung (vv) simuliert. Weiterhin erfolgten Netzwerksimulation mit Eingangsschichtgrößen von 32×32, 48×48, 64×64, 80×80 und

96×96 mit einer lateralen Vierer- und Achter-Nachbarschaftsverbindung. Jede Simulation wurde bis zu einer Netzwerkzeit von 500 ms durchgeführt. Die Membranpotentiale a_K aller Neuronen wurden mit zufälligen Werten zwischen 0 und 1 initialisiert. Der Schwellwert jedes Neurons wurde mit $\theta = 1$ definiert und die Pulsdauer auf $t_d = 1$ ms gesetzt. Alle Synapsengewichte (W_{KL} und W_0) wurden zu Beginn jeder Simulation mit dem Wert 0,12 ms^{-1} initialisiert und für die adaptiven Synapsengewichte W_{KL} zwischen den Neuronen der Eingangsschicht wurde die Adaptionsregel nach Gl. 2-15 auf Seite 20 angewendet. Dabei wurde jeweils die Abklingkonstante auf $\gamma = 0{,}1$ ms^{-1} und der Verstärkungsfaktor auf $\mu = 0{,}3$ ms^{-2} festgelegt.

NSC-Phase

In Abb. 6-1 sind die Prozentanteile an der Gesamtsimulationszeit T_{SW} für die Zustände *Next-Spike* und *Update* in der NSC-Phase (siehe auch Abb. 4-7 auf Seite 71) dargestellt.

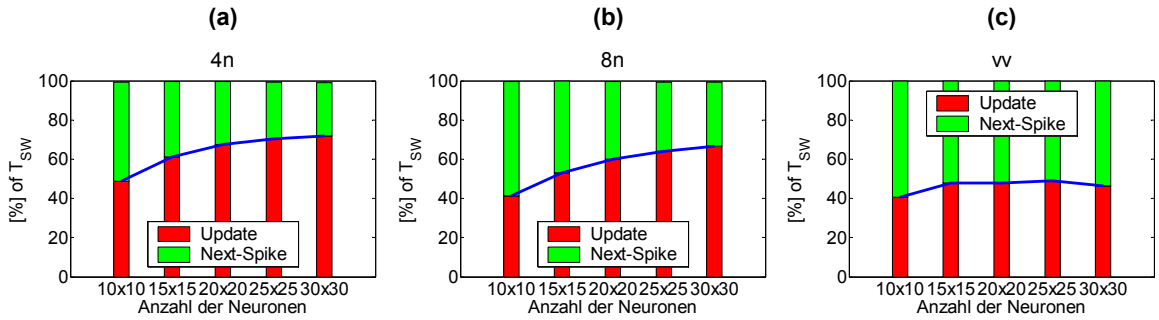


Abb. 6-1: Prozentanteil an der Gesamtsimulationszeit T_{SW} für die Zustände *Next-Spike* und *Update* in der Neuronenzustandsphase

Es zeigt sich, dass die anderen beiden Zustände der NSC-Phase *Read* und *Next-Event* in Abb. 4-7 bezüglich der notwendigen Simulationszeit vernachlässigt werden können. In Abb. 6-1a zeigt sich für eine Vierer-Nachbarschaftsverbindung, dass das Verhältnis vom *Update* Zustand zum *Next-Spike* Zustand mit zunehmender Netzwerkgröße größer wird. Bei einer Netzwerkschichtgröße von 30×30 ist die Simulationszeit für den *Update* Zustand etwa einen Faktor 3 größer als für den *Next-Spike* Zustand. Da im Verhältnis zur Gesamtanzahl der Neuronen immer weniger Neuronen durch eine Puls-Start-Ereignis erregt werden, ist mit zunehmender Netzwerkgröße die relative Anzahl an erregten Neuronen geringer, bei denen der Zeitpunkt eines möglichen Pulszustands ermittelt werden muss. Diese Verhältnisverschiebung zeigt sich ebenfalls bei einer Achter-Nachbarschaft in Abb. 6-1b. Bei einer Vollvernetzung allerdings bleibt dieses Verhältnis der notwendigen Simulationszeiten für *Next-Spike* zu *Update* konstant, da durch einen Puls sämtliche Neuronen in der Netzwerkschicht erregt werden. Dies bedeutet, dass sowohl bei der Suche nach dem nächsten Puls-Start-Ereignis (*Next-Spike*) als auch bei der Aktualisierung der Neuronenzustände (*Update*) für jedes Neuron eine numerische Integration durchgeführt werden muss.

Numerische Integration

In Abb. 6-2 sind die Prozentanteile an der Gesamtsimulationszeit T_{SW} für die numerische Integration dargestellt, die aus den beiden Operationen MMID-Integration und PZEXTR besteht.

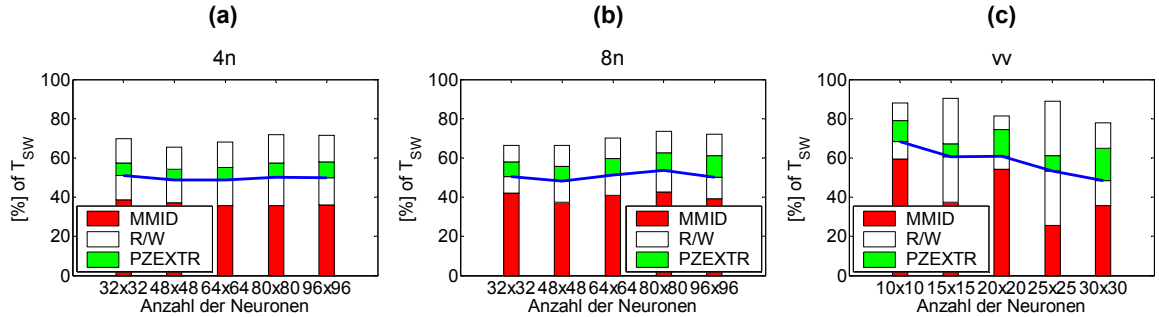


Abb. 6-2: Prozentanteil an der Gesamtsimulationszeit T_{SW} für die numerische Integration bestehend aus MMID-Integration und PZEXTR

Die weißen Balken stellen jeweils den Lese- und Schreibtransfer für die Membranpotentiale und adaptiven Synapsengewichte dar, die bei beiden Operationen zum Einsatz kommen und deshalb gleichmäßig auf die Prozentanteile der Simulationszeiten für die MMID-Integration und der PZEXTR verteilt sind. Man erkennt, dass etwa 70 % der Simulationszeit für das numerische Integrationsverfahren bei einer Vierer- und Achter-Nachbarschaft aufgebracht werden musste (siehe Abb. 6-2a und Abb. 6-2b). Bei einer lateralen Vollvernetzung liegt der Anteil an der Gesamtsimulationszeit für die numerische Integration bei über 80 % (siehe Abb. 6-2c).

4 Variablen

Um Aussagen über den Rechenaufwand des numerischen Integrationsverfahrens treffen zu können, wurden während der Software Simulationen 4 Variablen protokolliert: 1.) die Anzahl der aufgetretenen Simulationsereignisse (N_{EVENT}), 2.) die Anzahl der durchgeführten Integrationen nach Bulirsch-Stoer pro Neuron (N_{BS}), 3.) die Anzahl der durchgeführten Integrationsintervalländerungen pro Integration (N_H), und 4.) die Anzahl der notwendigen Integrationsschrittunterteilungen pro Integration (N_I).

Da bei jedem Simulationsereignis eine numerische Integration für ein Neuron immer im *Update* Zustand für die Netzwerkaktualisierung und nur im *Next-Spike* Zustand, wenn das Neuron durch einen oder mehrere präsynaptische Pulse erregt ist, durchgeführt wird, gilt für die Anzahl der durchgeführten Integrationen nach Bulirsch-Stoer pro Neuron die Bedingung $N_{EVENT} \leq N_{BS} \leq 2 \cdot N_{EVENT}$. Dies bedeutet, dass pro Simulationsereignis für jedes Neuron maximal 2 numerische Integrationen durchgeführt werden.

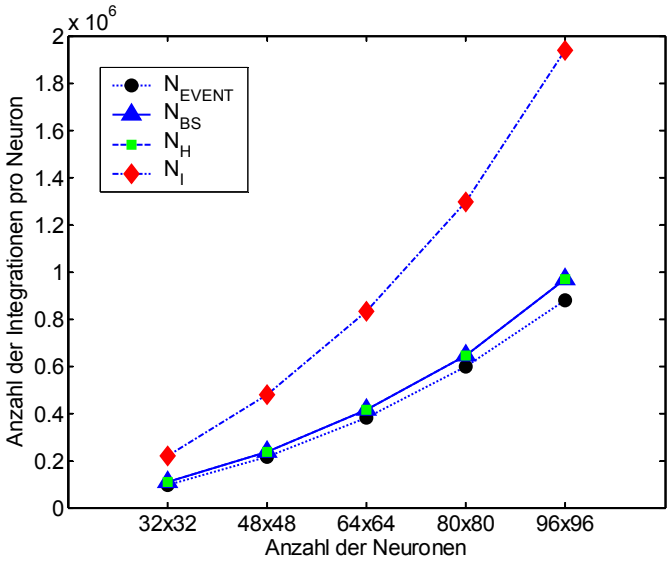


Abb. 6-3: Durchgeführte Integrationen pro Neuron (Vierer-Nachbarschaft)

In Abb. 6-3 und in Abb. 6-4 sind die 4 protokollierten Variablen für eine Vierer-Nachbarschaft und für eine Vollvernetzung angegeben. In Abb. 6-3 erkennt man, dass sich N_{BS} etwas oberhalb von N_{EVENT} befindet. Dies bedeutet, dass die Neuronen aufgrund der spärlichen Vernetzung selten von präsynaptischen Pulsen erregt wurden, und somit auch im *Next-Spike* Zustand selten eine numerische Integration durchgeführt werden muss. In Abb. 6-4 liegt N_{BS} allerdings näher an der oberen Grenze von $2 \cdot N_{EVENT}$, was bedeutet, dass die Neuronen häufig durch präsynaptische Pulse erregt werden. Für eine Vierer-Nachbarschaftsverbindung (siehe Abb. 6-3) als auch für eine Achter-Nachbarschaftsverbindung entspricht $N_H = N_{BS}$ und $N_I \approx 2 \cdot N_{BS}$. Dies bedeutet, dass die Vorgabe für das Integrationsintervall H für jede durchgeführte numerische Integration eines Neurons ausreichend war ($N_H = N_{BS}$) und dass es im Schnitt 2 Integrationsschrittunterteilungen pro durchgeführter numerischer Integration gab ($N_I \approx 2 \cdot N_{BS}$). Bei einer Vollvernetzung (siehe Abb. 6-4) zeigt sich, dass $N_H \approx N_{BS}$ und $N_I \approx 3 \cdot N_{BS}$ gilt.

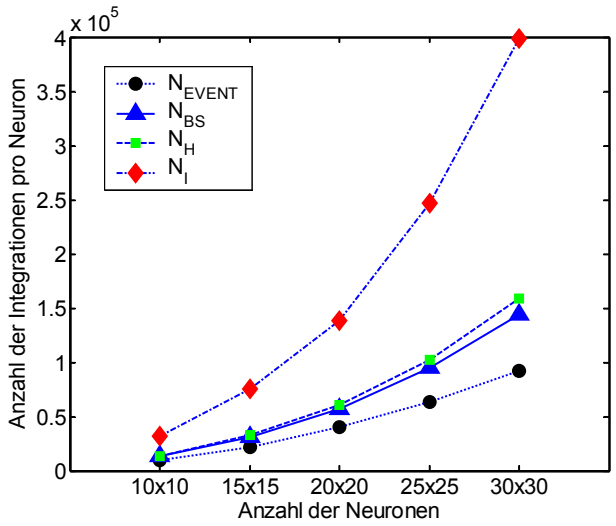


Abb. 6-4: Durchgeführte Integrationen pro Neuron (Vollvernetzung)

Simulationszeiten

In Abb. 6-5 ist die Anzahl der Neuronen (x-Achse), die Anzahl der Synapsen N_{WEIGHT} (linke y-Achse), sowie die Simulationszeiten auf dem Einprozessor-System T_{SW} (rechte y-Achse) dargestellt. Die Simulationszeiten verhalten sich bei diesen Netzwerkgrößen annähernd proportional zu der Anzahl der adaptiven Synapsengewichte. Die Simulationszeiten für Eingangsschichten mit einer Achter-Nachbarschaftsverbindung (8n) sind um den Faktor 1,1 bis 1,5 größer als mit einer Vierer-Nachbarschaftsverbindung (4n). Bei einer 10×10 großen Eingangsschicht dauert die PCNN Simulation bei einer Vollvernetzung (vv) um den Faktor 10 länger als bei einer Achter-Nachbarschaftsverbindung. Mit ansteigender Netzwerkgröße (15×15 bis 30×30) erhöht sich dieser Faktor jeweils um den Faktor 2. Bei einer 30×30 großen Eingangsschicht ist die Simulationszeit bei einer Vollvernetzung somit um den Faktor 160 größer als bei einer Achter-Nachbarschaft und beträgt 352460 s, was einer Simulationszeit von knapp 98 Stunden entspricht.

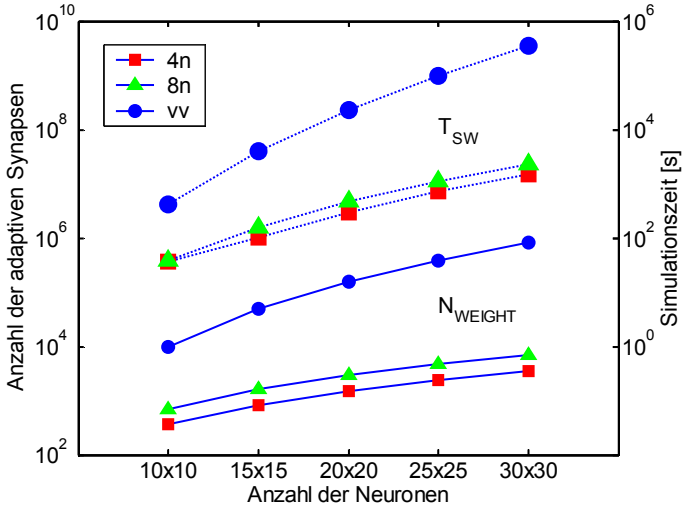


Abb. 6-5: Anzahl der Synapsen und Neuronen, Software Simulationszeiten: Vierer-Nachbarschaft (4n), Achter-Nachbarschaft (8n) und Vollvernetzung (vv)

In Abb. 6-6 ist die Anzahl der Neuronen N_{NEURON}, die Anzahl der Synapsen (linke y-Achse), sowie die Simulationszeiten für die numerische Integration auf dem Einprozessor-System $T_{SW\text{-}BS}$ (rechte y-Achse) dargestellt, die etwa 70 % der Simulationszeit insgesamt entspricht. Die simulierten Netzwerkgrößen liegen zwischen 32×32 und 96×96 Neuronen bei einer Vierer- und Achter-Nachbarschaftsverbindung. Man erkennt, dass ab einer Netzwerkgröße von 64×64 ein exponentieller Anstieg der Simulationszeit für die numerische Integration zu verzeichnen ist und sich somit nicht mehr annähernd proportional zur Anzahl der adaptiven Synapsengewichte verhält. Die Simulationszeit für die numerische

Integration bei einer 96×96 großen Eingangsschicht mit einer Achter-Nachbarschaftsverbindung beträgt 0,7211·198079 s = 142834 s, was einer Simulationszeit von mehr als 39,5 Stunden entspricht. Der Faktor 0,7211 entspricht dem Simulationsanteil, der für die numerische Integration (MMID-Integration und PZEXTR) benötigt wurde (siehe auch Reihe 10 in Tab. 6-1).

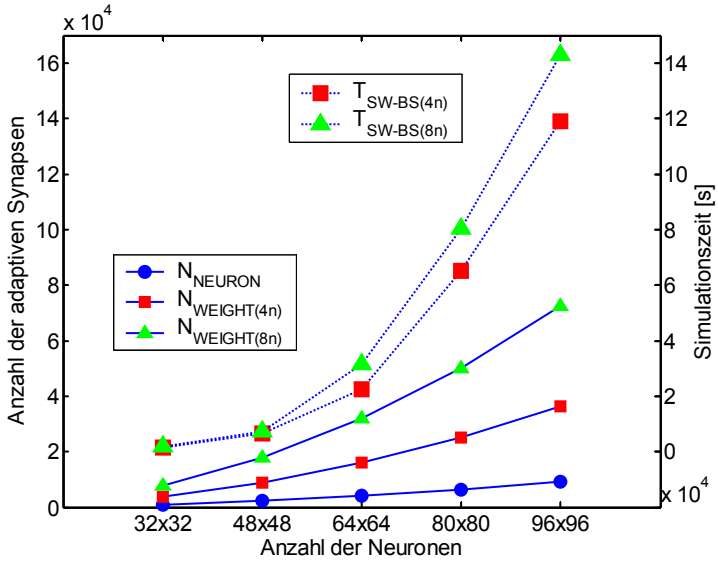


Abb. 6-6: Anzahl der Synapsen und Neuronen, Software Simulationszeiten: Vierer-Nachbarschaft (4n) und Achter-Nachbarschaft (8n)

Die Simulationszeiten der durchgeführten PCNN Simulationen sind in Tab. 6-1 zusammengefasst. Dabei entspricht n_S der durchschnittlichen Anzahl an adaptiven Synapsen pro Neuron. T_{MMID} bzw. T_{PZEXTR} sind die Simulationszeiten, die für die MMID-Integration bzw. für die Polynomextrapolation während der PCNN Simulation benötigt wurden.

n_S	N_{NEURON}	N_{BS}	T_{SW}	T_{MMID}	T_{PZEXTR}
4	32×32	98717	2012 s	51,00 % · T_{SW}	18,88 % · T_{SW}
	48×48	222365	9960 s	48,62 % · T_{SW}	16,92 % · T_{SW}
	64×64	420299	33182 s	48,72 % · T_{SW}	19,45 % · T_{SW}
	80×80	721463	90750 s	50,21 % · T_{SW}	21,72 % · T_{SW}
	96×96	926458	166307 s	49,83 % · T_{SW}	21,79 % · T_{SW}
8	32×32	107276	2998 s	50,51 % · T_{SW}	15,88 % · T_{SW}
	48×48	235863	10968 s	48,00 % · T_{SW}	18,22 % · T_{SW}
	64×64	413861	45044 s	51,24 % · T_{SW}	18,81 % · T_{SW}
	80×80	645694	109478 s	53,53 % · T_{SW}	19,89 % · T_{SW}
	96×96	967572	198079 s	50,14 % · T_{SW}	21,97 % · T_{SW}
N_{NEURON}^2	10×10	13064	421 s	68,25 % · T_{SW}	19,75 % · T_{SW}
	15×15	31063	4053 s	60,50 % · T_{SW}	29,90 % · T_{SW}
	20×20	57320	23024 s	60,90 % · T_{SW}	20,40 % · T_{SW}
	25×25	95140	97920 s	53,45 % · T_{SW}	35,55 % · T_{SW}
	30×30	143756	352460 s	48,40 % · T_{SW}	29,40 % · T_{SW}

Tab. 6-1: Software Simulationszeiten ($T_{SW-BS} = T_{MMID} + T_{PZEXTR}$)

6.1.1 Emulationszeitabschätzungen von SEE

Die Emulationszeit für die numerische Integration auf der digitalen Emulationsplattform SEE lässt sich nach der folgenden Formel abschätzen,

$$T_{SEE-BS} = N_{BS} \cdot \frac{N_{NEURON}}{N_{IO}} \cdot T_{CLK} \cdot T_{NEURON}(n_S) \quad \textbf{(6-1)}$$

wobei N_{BS} der durchschnittlichen Anzahl der durchzuführenden numerischen Integrationen pro Neuron und N_{NEURON} der Anzahl der zu emulierenden pulscodierten Neuronen auf der Emulationsplattform entspricht. N_{IO} ist die Anzahl der parallel nutzbaren Speicherkanäle der PEs zum Parameterspeicher, und T_{CLK} stellt die Taktperiode dar, mit der die Emulationsplattform betrieben werden kann. T_{NEURON} beschreibt die Taktanzahl, die für die Berechnung einer numerischen Integration eines postsynaptischen Neurons benötigt wird. Diese Taktanzahl ist von der Anzahl der adaptiven Synapsengewichte zu präsynaptischen Neuronen n_S abhängig und kann wie folgt berechnet werden,

$$T_{NEURON} = H_{AVG} \cdot \sum_{j=0}^{I_{AVG}-1} [T_{MMID}(n_S, j) + T_{PZEXTR}(n_S, j)] + t_{SDRAM} \quad \textbf{(6-2)}$$

wobei t_{SDRAM} die Latenzzeit in Takten für den einmaligen Lese- und Schreibzugriff eines NIBs im SDRAM ist. Dies ist allerdings nur bei einer geringen Anzahl von adaptiven Synapsengewichten möglich, da in diesem Fall die Funktionszwischenwerte für die numerische Integration (siehe auch Abb. 4-10 auf Seite 75) auf dem FPGA zwischengespeichert werden können. Bei einer dichten Vernetzungsstruktur wie bei einer Vollvernetzung werden die Funktionszwischenwerte immer wieder auf dem SDRAM-Speicher zwischengespeichert, so dass die Zugriffe auf den SDRAM der Anzahl der zu berechnenden Funktionszwischenwerte entspricht (siehe auch Abschn. "Vollvernetzung" auf Seite 101). H_{AVG} und I_{AVG} in Gl. 6-2 stellen die durchschnittliche Integrationsintervalländerung bzw. die Integrationsschrittunterteilungen dar, die pro Integration eines Neurons notwendig waren und die sich nach Gl. 6-3 aus den durchgeführten Simulationen auf dem Einprozessor-System ergeben (siehe auch Abb. 6-3 und Abb. 6-4 auf Seite 130).

$$H_{AVG} = \frac{N_H}{N_{BS}} \qquad I_{AVG} = \frac{N_I}{N_{BS}} \quad \textbf{(6-3)}$$

Es ergeben sich für Netzwerkschichten mit einer Vierer- oder Achter-Nachbarschaftsverbindung $H_{AVG} \approx 1$ und $I_{AVG} \approx 2$, und für Netzwerkschichten mit lateraler Vollvernetzung gilt $H_{AVG} \approx 1$ und $I_{AVG} \approx 3$ (siehe auch Abschn. "4 Variablen" auf Seite 129).

T_{MMID} und T_{PZEXTR} in Gl. 6-2 stellen die Anzahl der Takte dar, die für die MMID-Integration und für die PZEXTR notwendig sind. Die notwendige Taktanzahl für die MMID-Integration ergibt sich aus der Hardware-Abbildung in Abb. 4-12 auf Seite 77 und ist in Gl. 6-4 angegeben,

$$T_{MMID} = t_{MMID} + \mathrm{ceil}(n_S/2) \cdot [2 \cdot (j+1) + 1] \quad \textbf{(6-4)}$$

wobei t_{MMID} der Latenzzeit entspricht, bevor das erste gültige Datum aus der Pipeline des implementierten HW-Moduls erscheint. Der Term $n_S/2$ in Gl. 6-4 kommt dadurch zustande, dass die Datenbusbreite des Parameterspeichers in Form eines SDRAM-DIMM-Speichers 8 Bytes beträgt und somit stets 2 Synapsengewichte gleichzeitig gelesen werden können. Der Faktor $[2\cdot(j + 1) + 1]$ in Gl. 6-4 beschreibt die Anzahl der Funktionszwischenwerte, die bei einer Integrationsschrittunterteilung berücksichtigt werden müssen. Bei 2 Integrationsschritten ($j = 0$) müssen 3 Funktionszwischenwerte berechnet werden, bei 4 Integrationsschritten ($j = 1$) müssen 5 Funktionszwischenwerte berechnet werden usw. Bei einer dichten Vernetzungsstruktur muss aufgrund der großen Anzahl an adaptiven Synapsen für jede Berechnung eines Funktionszwischenwertes auf den

SDRAM-Speicher zugegriffen werden, so dass Gl. 6-4 entsprechend unter Berücksichtigung der jeweiligen Latenzzeiten zum SDRAM modifiziert werden muss (siehe Gl. 6-5).

$$T_{MMID-VV} = [t_{SDRAM} + t_{MMID} + \text{ceil}(n_S/2)] \cdot [2 \cdot (j+1) + 1] \quad \textbf{(6-5)}$$

Die notwendige Taktanzahl für die PZEXTR ergibt sich aus der Hardware-Abbildung in Abb. 4-14 auf Seite 80 und ist in Gl. 6-6 beschrieben,

$$T_{PZEXTR} = t_{PZEXTR} + \text{ceil}(n_S/2) + j \cdot [\text{ceil}(n_S/(2 \cdot t_{PZEXTR})) \cdot t_{PZEXTR}] \quad \textbf{(6-6)}$$

wobei t_{PZEXTR} wieder der Latenzzeit in Takten entspricht, die sich aufgrund der Pipeline des HW-Moduls ergibt. Der Faktor $j \cdot [\text{ceil}(n_S/(2 \cdot t_{PZEXTR})) \cdot t_{PZEXTR}$ in Gl. 6-6 beschreibt die Anzahl der zusätzlichen Takte, die notwendig ist, wenn man die Polynomextrapolation anhand von j Funktionswerten durchführt. Um diesen Faktor verständlicher darzustellen, ist in Abb. 6-7 ein Beispiel mit $n_S = 9$ und $t_{PZEXTR} = 4$ für $j = 0$ und $j = 1$ angegeben.

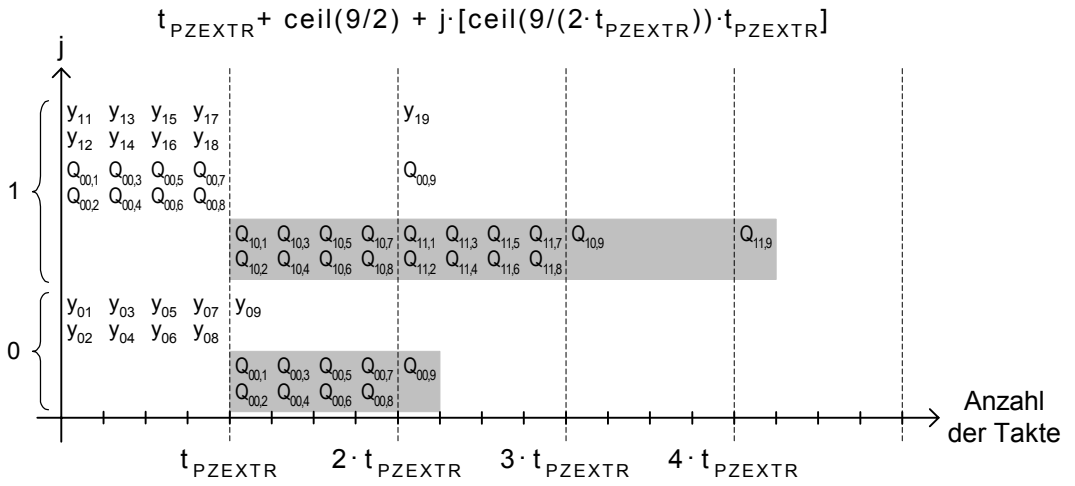


Abb. 6-7: Anzahl der notwendigen Takte für die Polynomextrapolation ($n_S = 9$; $t_{PZEXTR} = 4$)

Bei $j = 0$ ergibt sich nach Gl. 6-6 für $T_{PZEXTR} = 4 + \text{ceil}(9/2) = 4 + 5 = 9$ Takte. Die ersten 9 Funktionswerte y_{0x} generieren die Zwischenterme $Q_{00,x}$, die weggeschrieben werden (siehe grau markierte Terme in Abb. 6-7 für $j = 0$). Bei den zweiten 9 Funktionswerten y_{1x} werden zeitgleich die beim ersten Durchlauf der Polynomextrapolation generierten Zwischenterme $Q_{00,x}$ gelesen und die Zwischenterme $Q_{10,x}$ und $Q_{11,x}$ zurückgeschrieben (siehe auch Abschn. "Zahlenbeispiel" auf Seite 79). Für $j = 1$ ergibt sich nun eine Anzahl von $T_{PZEXTR} = 4 + \text{ceil}(9/2) + [\text{ceil}(9/8) \cdot 4] = 4 + 5 + [2 \cdot 4] = 17$ Takte.

Emulationszeiten

Anhand von Gl. 6-4 und Gl. 6-6 kann nun durch Einsetzen der entsprechenden Latenzzeiten ($t_{SDRAM} = 10$, $t_{MMID} = 12$ (siehe Gl. 4-8 auf Seite 77), $t_{PZEXTR} = 4$ (siehe Gl. 4-22 auf Seite 81)) die Berechnungsdauer in Takten für die numerische Integration eines Neurons mittels Gl. 6-2 und Gl. 6-3 angegeben werden ($H_{AVG} = 1$ und $I_{AVG} = 2$).

$$\begin{aligned} T_{NEURON}(n_S = 4) &= [2 \cdot 12 + 2 \cdot (3+5)] + [2 \cdot 4 + 2 \cdot 2 + 4] + 10 = 66 \\ T_{NEURON}(n_S = 8) &= [2 \cdot 12 + 4 \cdot (3+5)] + [2 \cdot 4 + 2 \cdot 4 + 4] + 10 = 86 \end{aligned} \quad \textbf{(6-7)}$$

Bei der Vollvernetzung ($H_{AVG} = 1$ und $I_{AVG} = 3$) wird für die Berechnungsdauer in Takten für die numerische Integration eines Neurons nur die Taktanzahl der MMID-Integration aus Gl. 6-5 betrachtet.

$$T_{NEURON}(n_S = 30^2) \approx H_{AVG} \cdot \sum_{j=0}^{I_{AVG}-1} T_{MMID-VV}(n_S, j) = (10 + 12 + 450) \cdot (3 + 5 + 7) = 7080 \quad \textbf{(6-8)}$$

Für eine Operationsfrequenz von 50 MHz ($T_{CLK} = 20$ ns) und der parallelen Nutzung von 3 Speicherkanälen bei einer spärlichen Vernetzungsstruktur (4n, 8n $\Rightarrow N_{IO} = 3$) und von nur einem Speicherkanal bei einer dichten Vernetzungsstruktur (vv $\Rightarrow N_{IO} = 1$) lassen sich mit der Anzahl an durchgeführten numerischen Integrationen pro Neuron (N_{BS} in Tab. 6-1 auf Seite 131) die Emulationszeiten für SEE abschätzen.

In Gl. 6-9 sind die Emulationszeiten für die numerische Integration von einer 96×96 großen Eingangsschicht mit einer Vierer- und Achter-Nachbarschaftsverbindung angegeben.

$$T_{SEE-BS} = 926458 \cdot \frac{96^2}{3} \cdot 20 \cdot 10^{-9}\text{s} \cdot 66 \approx 3757\text{s} \quad | \; N_{NEURON} = 96 \times 96, n_S = 4$$
$$T_{SEE-BS} = 967572 \cdot \frac{96^2}{3} \cdot 20 \cdot 10^{-9}\text{s} \cdot 86 \approx 5113\text{s} \quad | \; N_{NEURON} = 96 \times 96, n_S = 8 \qquad \textbf{(6-9)}$$

Die abgeschätzte Emulationszeit für die MMID-Integration einer vollvernetzten Netzwerkschicht mit 30×30 Neuronen ist in Gl. 6-10 dargestellt.

$$T_{SEE-MMID} = 143756 \cdot \frac{30^2}{1} \cdot 20 \cdot 10^{-9}\text{s} \cdot 7080 \approx 18321\text{s} \quad | \; N_{NEURON} = 30 \times 30, n_S = 30^2 \qquad \textbf{(6-10)}$$

Die resultierenden Beschleunigungsfaktoren *F* berechnen sich aus dem Verhältnis von $T_{SW\text{-}BS}$ zu $T_{SEE\text{-}BS}$ bei einer Vierer- und Achter-Nachbarschaft und aus dem Verhältnis von $T_{SW\text{-}MMID}$ zu $T_{SEE\text{-}MMID}$ bei einer Vollvernetzung. Für unterschiedliche Netzwerkgrößen sind diese Beschleunigungsfaktoren in Tab. 6-2 und in Tab. 6-3 zusammengefasst.

n_S	N_{NEURON}	H_{AVG}	I_{AVG}	T_{NEURON}	$T_{SEE\text{-}BS}$	$T_{SW\text{-}BS}$	F
4	32×32	1	2	66	45 s	1405 s	31,2
	48×48				226 s	6527 s	28,9
	64×64				758 s	22620 s	29,8
	80×80				2032 s	65277 s	32,1
	96×96				3757 s	119109 s	31,7
8	32×32	1	2	86	63 s	1990 s	31,6
	48×48				312 s	7263 s	23,3
	64×64				972 s	31548 s	32,5
	80×80				2370 s	80378 s	33,9
	96×96				5113 s	142834 s	27,9

Tab. 6-2: Abgeschätzte Emulationszeiten von SEE (bei Vierer- und Achter-Nachbarschaftsverbindungen)

n_S	N_{NEURON}	H_{AVG}	I_{AVG}	T_{NEURON}	$T_{SEE\text{-}MMID}$	$T_{SW\text{-}MMID}$	F
N_{NEURON}^2	10×10	1	3	1080	29 s	287 s	9,8
	15×15			2025	284 s	2452 s	8,6
	20×20			3330	1528 s	14021 s	9,1
	25×25			5025	5976 s	52338 s	8,7
	30×30			7080	18321 s	170590 s	9,3

Tab. 6-3: Abgeschätzte Emulationszeiten von SEE (bei lateraler Vollvernetzung)

Bei einer Taktfrequenz von 50 MHz kann die Emulationsplattform SEE für die numerische Integration, die etwa 70 % der Simulationszeit einnimmt (siehe Tab. 6-1 auf Seite 131), einen Beschleunigungsfaktor von 30 für spärliche Vernetzungsstrukturen, wie Vierer- und Achter-Nachbarschaftsverbindungen, erzielen. Für die MMID-Integration, die bei dichten Vernetzungsstrukturen (laterale Vollvernetzung) etwa 60 % der Simulationszeit benötigt, kann auf der Emulationsplattform ein Beschleunigungsfaktor von 8 erreicht werden.

6.1.2 Funktionalitätsüberprüfung von SEE

Die Funktionalitätsüberprüfung der Emulationsplattform erfolgte im Rahmen von Eingangsschichten von bis zu 64 Neuronen mit einer Vierer-Nachbarschaftsverbindung. Dabei wurde auf der Emulationsplattform nur die Funktion einer einzigen PE bei einer Operationsfrequenz von 25 MHz überprüft.

Die PCNN Simulation auf dem Einprozessor-System benötigt für 64 Neuronen bis zur Netzwerkzeit von 1000 ms exakt 20 s. Die numerische Integration nimmt bei diesen Netzwerkgrößen etwa 60 % der Simulationszeit in Anspruch. Die PCNN Emulation auf der Emulationsplattform benötigt für die gleiche Eingangsschicht unter 5 s, nachdem das PCNN über die serielle Schnittstelle konfiguriert wurde. Pro Neuron werden knapp 10000 numerische Integrationen durchgeführt. Nach Gl. 6-1 auf Seite 132 würde sich für eine Vierer-Nachbarschaftsverbindung (T_{NEURON} = 66) und der Verwendung von nur einer PE (N_{IO} = 1), eine Emulationszeit von 10000·64·40·10-9 s·66 = 1,7 s für die numerische Integration ergeben. Der Beschleunigungsfaktor *F* für die numerische Integration entspricht in diesem Fall 0,6·20 s/1,7 s = 6,3. Wenn man die Simulationszeit und Emulationszeit insgesamt betrachtet, ergibt sich ein Beschleunigungsfaktor von 20 s/5 s = 4. Da nur eine PE für die Emulation bei einer doppelt so großen Taktperiode verwendet wurde, muss man den abgeschätzten Beschleunigungsfaktor von 30 für eine Vierer-Nachbarschaftsverbindungen durch den Faktor 6 dividieren. Diese würde dann einem abgeschätzten Beschleunigungsfaktor von 5 entsprechen.

6.2 Leistungsgrenzen von SEE

Die Leistungsgrenzen von SEE sind zum einen durch die Speicherbandbreite, welche sich proportional zur Performanz der Neuronenzustandsberechnung verhält, und zum anderen durch die Speicherkapazität, welche die emulierbare PCNN Größe widerspiegelt, gegeben.

Performanzgrenze

Wie in Kapitel 4.2.3 auf Seite 71 bereits ermittelt, beträgt die Speicherbandbreite zum Parameterspeicher, indem die NIBs abgelegt sind, 1,2 GB/s. Diese Speicherbandbreite steht für die Aktualisierung der Neuronenzustände zur Verfügung, um dynamische Netzwerkparameter, wie Membranpotentiale und adaptive Synapsengewichte, zu aktualisieren. Das verwendete numerische Integrationsverfahren von Bulirsch-Stoer erfordert, dass 2 Funktionswerte gelesen werden müssen, um den nächsten Funktionswert zu ermitteln. Die Bereitstellung von 3 parallel nutzbaren Speicherkanälen ermöglicht die Aktualisierung von Neuronenzuständen bei einer optimalen Auslastung der Speicherkanäle, auch wenn die Funktionszwischenwerte aufgrund einer dichten Vernetzungsstruktur nicht auf dem FPGA gespeichert werden können. Es können jeweils 2 Speicherkanäle genutzt werden, um parallel Funktionswerte zu lesen. Über den dritten Speicherkanal werden die neu errechneten Funktionswerte zeitgleich in den Parameterspeicher zurückgeschrieben. Insgesamt verfügt SEE über 7 weitere parallel nutzbare Datenkanäle mit einer jeweiligen Busbreite von 32 Bit. Diese Datenkanäle sind notwendig, um parallel auf Markierungsspeicher, Ereignislisten und Netzwerktopologieinformationen zuzugreifen und um aufgetretene Emulationsereignisse abzuspeichern. Die Gesamtspeicherbandbreite der insgesamt 10 Datenkanäle beträgt 2,6 GB/s.

Emulationsgrenze

Die maximale Anzahl der Neuronen und Synapsen bei SEE ist durch die Größe der DEL und des Paramterspeichers begrenzt und beträgt für die Neuronen $0{,}5 \cdot 10^6$ und für die Synapsen $800 \cdot 10^6$. Als Netzwerkarchitekturtypen wird zwischen einer Kaskaden-Architektur und einer Gabor-Architektur unterschieden.

Bei einer Kaskaden-Architektur kann bei einem einlagigen Netzwerk ein Eingangsbild im SVGA-Format (SVGA = *Super Video Graphics Array*), das aus 800×600 Pixeln besteht, berücksichtigt werden. Dies ist in Abb. 6-8a dargestellt, in der sich die maximalen x- und y-Koordinatenwerte unterhalb der gestrichelten Linie für die Netzwerkschicht befinden müssen. Bei einer Netzwerkschicht mit lateraler Vollvernetzung, bei der die Synapsenan-

zahl annähernd dem Quadrat der Neuronenanzahl entspricht, ist allerdings nur eine Größe der Eingangsschicht im QQVGA-Format (*Quarter Quarter VGA*) mit 160×120 Pixeln zulässig (siehe Abb. 6-8b).

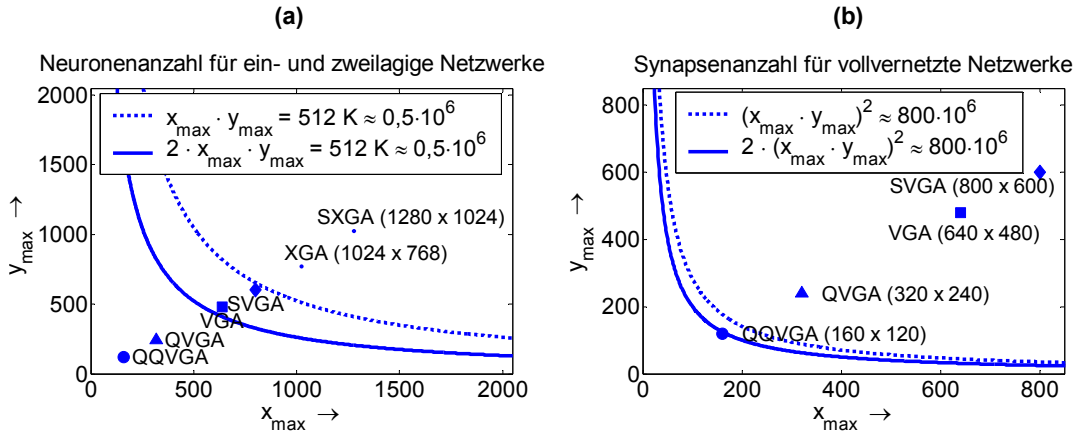


Abb. 6-8: Begrenzung der Neuronen- und Synapsenzahlen (bei einer Kaskaden-Architektur)

Bei einer Gabor-Architektur kann die maximale Anzahl der emulierbaren Neuronen und Synapsen nach den beiden Gleichungen Gl. 4-25 und Gl. 4-26 auf Seite 88, die die Neuronen- und Synapsenanzahl für einen Gabor-Detektor formulieren, angegeben werden. In Abb. 6-9 sind die Neuronen- bzw. Synapsenanzahl einer Gabor-Architektur als Produkt der Anzahl von Gabor-Detektoren n_D und der Anzahl von RF-Masken n_{RF} angegeben, welches von der durchschnittlichen Anzahl der Gradientenpaare n_{GDP} pro RF-Maske abhängig ist. Die maximale Neuronenanzahl wird durch die Größe der Eingangsschicht beeinflusst, und man erkennt in Abb. 6-9a, dass das Produkt $n_D \cdot n_{RF}$ bei n_{GDP} = 30 etwa 4000 beträgt, wenn die Eingangsschicht im QQVGA-Format vorliegt. Das bedeutet, dass 4 Gabor-Detektoren mit jeweils 1000 RF-Masken oder aber auch ein einziger Gabor-Detektor mit 4000 RF-Masken berücksichtigt werden können. Die Abb. 6-9b verdeutlicht, dass man weitaus mehr Gabor-Detektoren und -Masken berücksichtigen kann, wenn man nur die Speicherkapazität für die Synapsen betrachtet (das Produkt $n_D \cdot n_{RF}$ ist etwa um den Faktor 500 größer). Die Gabor-Architektur ist folglich eine Netzwerkarchitektur, die im wesentlichen durch die Anzahl der Neuronen eingeschränkt wird.

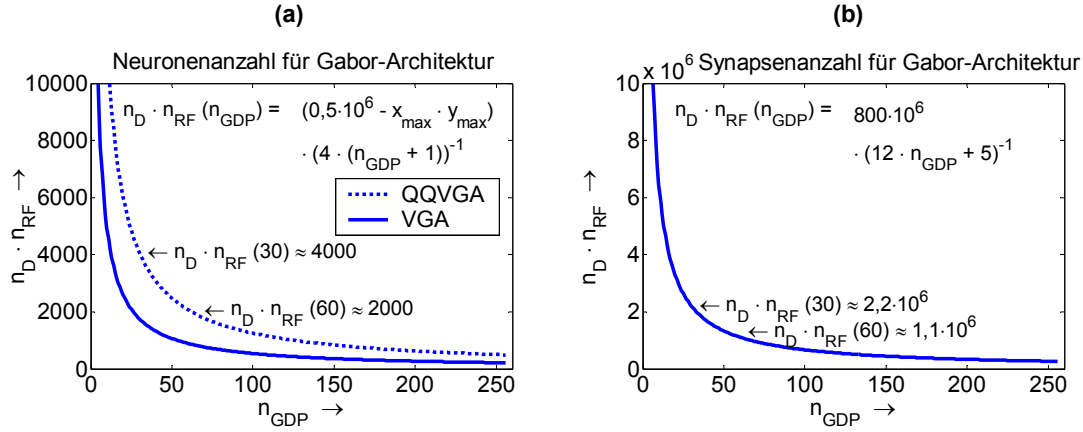


Abb. 6-9: Begrenzung der Neuronen- und Synapsenzahlen (bei einer Gabor-Architektur)

Bevor die Implementierung von SEE bewertet werden kann, ist eine Leistungsbewertung von anderen digitalen Emulationsplattformen notwendig. Dabei wird versucht, das verwendete Neuronenmodell mit adaptiven Gewichten auf die jeweilige Emulationsplattform zu übertragen. Für die Ermittlung der PCNN Größe wird die Codierung von 4 Bytes für Membranpotentiale (Neuronen) und Gewichtswerte (Synapsen) herangezogen, die für eine ausreichende Genauigkeit der Festkomma-Arithmetik für das Neuronenmodell erforderlich ist.

6.3 Leistungsbewertung anderer Emulationsplattformen

Bei der Leistungsbewertung anderer digitaler Emulationsplattformen werden Halb-Kunden-Entwurf basierte und FPGA basierte Emulationsplattformen herangezogen. Bei der Leistungsbeurteilung der Plattformen werden von den 4 Gesichtspunkten, die beim Vergleich von analoger und digitaler PCNN Emulation (siehe Tab. 3-1 auf Seite 46) und bei der Effizienzbetrachtung digitaler Simulationsplattformen (siehe Kapitel 3.4.6 auf Seite 57) verwendet wurden, die beiden Gesichtspunkte Performanz (Speicherbandbreite) und PCNN Größe (On-Board Speicherkapazität) herausgestellt. Die Performanz einer Emulationsplattform wird mit der verfügbaren Speicherbandbreite, die das Produkt von Taktfrequenz und Bitbreite des Speicherbusses darstellt, gleichgesetzt. Die im Kapitel 3.3.3 auf Seite 42 vorgestellten Mechanismen zur Reduzierung der Speicherbandbreite lassen sich bei der Verwendung des Neuronenmodells mit adaptiven Synapsengewichten nicht nutzen. Das Konzept des Weight Sharings, bei dem konstante Synapsengewichte von Neuronen mehrfach genutzt werden, ist bei der Berücksichtigung von adaptiven Synapsengewichten nicht möglich. Die Vernachlässigung von Potentialwerten führt zu einer Verfälschung der Netzwerkdynamik und würde auch bei einer PCNN Software Simulation auf einem Einrechner-System zu einer Beschleunigung führen. Schließlich ist das Konzept der markierten Potentialspeicherung ungeeignet, wenn der Parameterblock eines Neurons neben den Neuronen-spezifischen Parametern, wie Membranpotential und Schwellwert, auch die adaptiven Synapsengewichte von präsynaptischen Neuronen enthält. Bei einer komprimierten Darstellung von Parameterblöcken würde die Zuordnung der Synapsengewichte zu den präsynaptischen Neuronen verloren gehen. Bei dem Neuronenmodell mit adaptiven Synapsengewichten ist nur die Markierung von Neuronen im Ruhezustand sinnvoll (siehe Tab. 4-1 auf Seite 70), da in diesem Fall der komplette Parameterblock des Neurons nicht aus dem Parameterspeicher gelesen werden muss. Bezogen auf die Gesichtspunkte Kosten und Flexibilität, sind die Kosten bei Halb-Kunden-Entwurf basierten Emulationsplattformen aufgrund der geringen Stückzahl höher als bei FPGA-Entwürfen. Die Flexibilität ist allerdings bei FPGA basierten Emulationsplattformen aufgrund der Konfigurierbarkeit der Hardware-Module höher im Vergleich zu Halb-Kunden-Entwürfen, deren Hardware nach der Fertigung nicht mehr verändert werden kann.

6.3.1 Halb-Kunden-Entwurf basierte Plattformen

Die Performanz und PCNN Größe (maximale Anzahl an Neuronen und Synapsen) von den in Kapitel 3.4.4 auf Seite 50 vorgestellten Halb-Kunden-Entwurf basierten Emulationsplattformen NESPINN, MASPINN und SP²INN wird im folgenden zusammengefasst.

NESPINN

Die NESPINN-Plattform (siehe auch Kapitel 3.4.4.1 auf Seite 50) operiert bei einer Taktfrequenz von 50 MHz [Jahn96]. Der Connection Chip, der die Neuronenzustandsberechnung durchführt, ist mit 4 SRAM-Speichern à 64 KB verbunden, die jeweils einen 24 Bit-breiten Datenbus haben. Für die Neuronenzustandsberechnung stehen somit 4 parallel nutzbare Speicherkanäle zur Verfügung, die eine Speicherbandbreite von 4·3 Bytes·50 MHz = 600 MB/s bereitstellen. Ein zusätzlicher DRAM-Speicher (128 MB) dient als Topologiespeicher für irreguläre Netzwerkverbindungen. Somit verfügt NESPINN insgesamt über 5 parallel nutzbare Datenkanäle. Unter der Annahme, dass der SDRAM-Speicher über einen 32 Bit Datenbus verfügt, ergibt sich eine Speicherbandbreite von insgesamt (12+4) Bytes·50 MHz = 800 MB/s.

In [Jahn96] wird angegeben, dass die NESPINN-Plattform 10^5 Neuronen verarbeiten kann. Die Pulsliste, die mit dem Connection Chip verbunden ist, müsste eine entsprechende Speicherkapazität bereitstellen, damit diese Anzahl an Neuronen berücksichtigt werden kann. Für adaptive Synapsengewichte, die mit 4 Bytes codiert werden, steht der Plattform eine Gesamtspeicherkapazität von 4·64 KB + 128 MB = 128,256 MB zur Verfügung. Somit beträgt die maximale Anzahl an Synapsen $(128{,}256 \cdot 10^6 - 4 \cdot 10^5)/4 \approx 32 \cdot 10^6$.

Die Gatteranzahl des Connection Chips wird in [Jahn96] mit 10 K angegeben, und die Chip-Fläche des Processing Chips ist um den Faktor 1,1 größer als der Neuro-Pipe-Chip der Nachfolge-Plattform MASPINN (siehe [Schö02b] auf Seite 167).

MASPINN

Die anvisierte Taktfrequenz bei der MASPINN-Plattform (siehe auch Kapitel 3.4.4.2 auf Seite 51) beträgt 100 MHz [Schö02a]. Der für die Neuronenzustandsberechnung realisierte Neuro-Pipe-Chip ist mit 2 SRAM-Speichern verbunden, die für die Abspeicherung von Dendritenpotentialen und Dendritenmarkierungen (*DP-Tag*) vorgesehen sind. Beide SRAM-Speicher verfügen über einen 32 Bit-breiten Datenbus. Für die Neuronenzustandsberechnung können somit 2 Speicherkanäle parallel angesteuert werden, die eine Speicherbandbreite von 2·4 Bytes·100 MHz = 800 MB/s bereitstellen. Zusätzlich wird bei MASPINN über einen Weight-Cache, der für Dendritenpotentiale 8 MB und für Dendritenmarkierungen 512 KB vorsieht, auf den Gewichtsspeicher zugegriffen. Dadurch stehen auf der MASPINN-Plattform insgesamt 3 parallel nutzbare Datenkanäle zur Verfügung. Unter der Annahme, dass der Datenbus zum Gewichtsspeicher ebenfalls aus 32 Bit besteht, beträgt die Speicherbandbreite insgesamt (8+4) Bytes·100 MHz = 1,2 GB/s.

Da sowohl für die SRAM-Speicher des Neuro-Pipe-Chips als auch für den Gewichtsspeicher keine Angaben über die Speicherkapazität in [Schö02b] existieren, wird angenommen, dass die SRAM-Speicher jeweils 2 MB bereitstellen, und der Gewichtsspeicher aus einem 128 MB großen SDRAM-Speicher besteht. Die Speicherkapazität für Neuronen und Synapsen, die jeweils mit 4 Bytes codiert sind, würde in diesem Fall 2·2 MB + 128 MB = 132 MB betragen. Die maximale Anzahl an Neuronen wäre demnach 10^6 und die Anzahl an Synapsen würde sich auf $(132 \cdot 10^6 - 4 \cdot 10^6)/4 \approx 32 \cdot 10^6$ belaufen.

Die Gatteranzahl des Neuro-Pipe-Chip beträgt nach [Schö02a] 100 K. Zum Connection Chip der Connection Unit auf der MASPINN-Plattform gibt es jedoch keine Gatteranzahlangaben.

SP²INN

Die SP²INN-Plattform (siehe auch Kapitel 3.4.4.3 auf Seite 52) wird mit einer Taktfrequenz von 100 MHz betrieben [Mehr03]. Der konzipierte Neuro-Prozessor steuert einen SDRAM-DIMM-Speicher von 256 MB an, der als Parameterspeicher dient und einen 36 Bit-breiten Datenbus hat. Die relevanten Parameter für die Neuronenzustandsberechnung, wie Membranpotentiale und Synapsengewichte, sind somit nur über einen einzigen Speicherkanal erreichbar. Daraus ergibt sich eine Speicherbandbreite von 4,5 Bytes·100 MHz = 450 MB/s. Weiterhin ist der Neuro-Prozessor mit jeweils 3 SRAM-Speichern verbunden: einer Ereignisliste (256 KB) mit einem Datenbus von 24 Bit, dem TNT-Speicher (32 KB) und dem SNT-Speicher (64 KB) mit einem jeweiligen Datenbus von 32 Bit. Die Speicherbandbreite der insgesamt 4 parallel nutzbaren Datenkanäle beträgt (4,5+3+2·4) Bytes·100 MHz = 1,55 GB/s.

Die maximale Anzahl an Neuronen ist durch die Speicherkapazität des TNT-Speichers begrenzt und könnte bei der Speicherung von Neuronen-Tags lediglich $2^{15} \cdot 2^5 \approx 10^6$ betragen. Die insgesamt zur Verfügung stehende Speicherkapazität vom SDRAM-DIMM-Speicher und den 3 SRAMs beläuft sich auf 256 MB + (256+32+64) KB = 256,352 MB. Werden Neuronen und Synapsen mit 4 Bytes codiert, ergibt sich eine Gesamtanzahl von $(256{,}352 \cdot 10^6 - 4 \cdot 10^6)/4 \approx 63{,}1 \cdot 10^6$ Synapsen.

Die Gatteranzahl des SP²INN-Neuro-Prozessors ist in [Mehr03] mit 250 K angegeben.

6.3.2 FPGA basierte Plattformen

Die Darstellung von Speicherbandbreite und Speicherkapazität der 3 Plattformen SPIKE128K, RAPTOR und BEE (siehe auch Kapitel 3.4.5 auf Seite 53) zeigt die beeindruckende Entwicklung von FPGA basierten Emulationsplattformen bezüglich ihrer Performanz und der emulierbaren PCNN Größe.

SPIKE128K

Die SPIKE128K-Plattform (siehe auch Kapitel 3.4.5.1 auf Seite 54) ist für eine Taktfrequenz von 10 MHz ausgelegt [Fran99]. Das Neuronenmodul, welches die Neuronenzustandsphase ausführt, besteht aus 6 Submodulen. Jedes dieser Submodule verfügt über 2 SRAM-Speicher (128 K×16 Bit) für die Variablen- und Parameterspeicherung. Zusätz-

lich ist das Neuronenmodul über einen 32 Bit-breiten Datenbus mit dem 64 MB großen Topologiespeicher verbunden und besitzt auf diese Weise 6·2 + 1 = 13 parallel nutzbare Speicherkanäle. Die dadurch verfügbare Speicherbandbreite beläuft sich auf (6·2·2 + 4) Bytes·10 MHz = 280 MB/s. Im Kommunikationsmodul, welches den Neuronentransfer zwischen Neuronen- und Topologiemodul steuert, kommen schließlich noch 2 Neuronenlisten zum Einsatz (32 K×48 Bit). Somit stehen beim SPIKE128K insgesamt 14 parallel nutzbare Datenkanäle zur Verfügung. Die maximal erzielbare Speicherbandbreite beträgt (28+6) Bytes·10 MHz = 340 MB/s.

Für 16 Bit-breite Neuronenparameter beträgt die maximale Neuronenanzahl $128 \cdot 10^3$ Neuronen. Für Neuronenparameter, die mit 4 Bytes codiert werden, halbiert sich diese Zahl auf $64 \cdot 10^3$. Die insgesamt zur Verfügung stehende Speicherkapazität beträgt 64 MB + 6·2·256 KB + 192 KB = 67,264 MB. Erfolgt die Codierung von Synapsen ebenfalls mit 4 Bytes können maximal $(67{,}264 \cdot 10^6 - 4 \cdot 64 \cdot 10^3)/4 \approx 16{,}75 \cdot 10^6$ Synapsen von der SPIKE128K-Plattform berücksichtigt werden.

Die SPIKE128K-Plattform ist nach [Fran99] mit 9 FPGAs von Xilinx (8×XC3190 und 1×XC4008) bestückt. Dies entspricht nach [Xili98] und nach [Xili99] einer Anzahl von 8·320·2 + 324·2 ≈ 5,8 K Logic Cells (bei diesen älteren FPGAs gilt, dass ein CLB etwa 2 Logic Cells entspricht; siehe auch Abb. 4-2 auf Seite 63).

RAPTOR2000

Die RAPTOR2000-Plattform (siehe auch Kapitel 3.4.5.2 auf Seite 55) wird bei einer Taktfrequenz von 65 MHz betrieben [Porr02]. Jedes dieser insgesamt 6 applikationsspezifischen Module (ASM) ist mit einem 128 MB großen SDRAM-Speicher über einen 32 Bit-breiten Datenbus verbunden und kann für die Neuronenzustandsberechnung eingesetzt werden. Es können also 6 parallel nutzbare Speicherkanäle eingesetzt werden, die zusammen eine Speicherbandbreite von 6·4 Bytes·65 MHz = 1,56 GB/s zur Verfügung stellen. Diese Speicherbandbreite kann nicht ausschließlich für die Neuronenzustandsberechnung genutzt werden, da diese Datenübertragungsrate gleichzeitig auch die Gesamtdatenübertragungsrate der RAPTOR2000-Plattform darstellt. Alle 6 ASMs sind zum Datenaustausch zusätzlich an einen Dual-Ported SRAM-Speicher angeschlossen, der für 32 Bit-breite Ein- und Ausgangsdaten einen eigenen Datenbus zur Verfügung stellt.

Als Gesamtspeicherkapazität stehen insgesamt 6·128 MB = 768 MB zur Verfügung. Wenn die Anzahl der Neuronen, wie bei SEE, $0{,}5 \cdot 10^6$ Neuronen betragen soll, würde sich eine maximale Anzahl an $(768 \cdot 10^6 - 4 \cdot 0{,}5 \cdot 10^6)/4 \approx 191 \cdot 10^6$ Synapsen ergeben.

Die RAPTOR2000-Plattform verwendet nach [Porr02] bei 6 ASMs insgesamt 6 FPGAs von Xilinx (XCV1000). Dies entspricht nach [Xili01] einer Anzahl von 6·27648 ≈ 166 K Logic Cells.

BEE

Die BEE-Plattform (siehe auch Kapitel 3.4.5.3 auf Seite 56) operiert bei einer Taktfrequenz von etwa 13 MHz [Chan03b], wenn die für Berechnungsaufgaben verwendbaren 16 FPGAs nur ihre jeweiligen SRAM-Speicher à 1 MB nutzen, welche sich auf der Platine befinden. Für eine PCNN Emulation müsste man die insgesamt 2112 verfügbaren IO-Pins über Zusatzplatinen an SDRAM-DIMM-Speicher anschließen. Wenn man die Hälfte dieser IO-Pins für Datenbits nutzen kann (die andere Hälfte stünde dann für Adress- und Steuerleitungen zur Verfügung), ergeben sich 0,5·2112 Bit = 1056 Bit = 264 Bytes. Wenn jeweils 8 Bytes einem SDRAM-DIMM-Speicher zugeordnet werden, ergeben sich 264/8 = 33 Speicherkanäle, die parallel genutzt werden könnten. Die für die Neuronenzustandsberechnung zur Verfügung stehende Speicherbandbreite wäre in diesem Fall 264 Bytes·13 MHz = 3,432 GB/s. Wenn man die 16 SRAM-Speicher mitberücksichtigt, ergeben sich 33 + 16 = 49 Datenkanäle, die die BEE-Plattform zeitgleich nutzen könnte. Die insgesamt zur Verfügung stehende Speicherbandbreite würde sich demnach auf (264 +16·4) Bytes·13 MHz = 4,264 GB/s erhöhen.

Bei der Verwendung von 1 GB großen SDRAM-DIMM-Speichern wäre die Anzahl der Neuronen und Synapsen bei der BEE-Plattform (aufgrund der 33 Speicherkanäle) um den Faktor 11 größer als bei SEE. Dies würde zu insgesamt $11 \cdot 0{,}5 \cdot 10^6 = 5{,}5 \cdot 10^6$ Neuronen und $11 \cdot 800 \cdot 10^6 = 8{,}8 \cdot 10^9$ Synapsen führen, die emuliert werden könnten.

Die RAPTOR2000-Plattform verwendet 16 FPGAs von Xilinx (XCV2000E), was nach [Xili02] einer Anzahl von 16·43200 ≈ 692 K Logic Cells entspricht.

SEE

Die Performanz bezüglich der maximalen Speicherbandbreite und der maximalen emulierbaren PCNN Größe bei SEE ist dem Abschn. "Performanzgrenze" und dem Abschn. "Emulationsgrenze" auf Seite 135 zu entnehmen. Die Gesamtanzahl der Logic Cells der 3 FPGAs, die bei SEE verwendet werden, ist der Tab. 5-5 auf Seite 111 bzw. [Xili05b] und [Xili05c] zu entnehmen, und beträgt 9280·2 + 2·23040·2 ≈ 111 K.

Die im Rahmen der Leistungsbewertung ermittelten Werte für Speicherbandbreite, PCNN Größe und Anzahl der ASIC Gatter bzw. FPGA Logic Cells der digitalen Emulationsplattformen sind in Tab. 6-4 zusammengefasst.

Emulations-plattform	Speicherbandbreite für		Gesamtanzahl der		Gesamtanzahl der	
	NSC	PCNN Emulation	Neuronen	Synapsen	Gatter	Logic Cells
NESPINN	0,6 GB/s	0,8 GB/s	$0{,}1 \cdot 10^6$	$32 \cdot 10^6$	120 K	–
MASPINN	0,8 GB/s	1,2 GB/s	10^6	$32 \cdot 10^6$	> 100 K	–
SP²INN	0,45 GB/s	1,55 GB/s	10^6	$63{,}1 \cdot 10^6$	250 K	–
SPIKE128K	0,28 GB/s	0,34 GB/s	$0{,}064 \cdot 10^6$	$16{,}75 \cdot 10^6$	–	5,8 K
RAPTOR	< 1,56 GB/s	1,56 GB/s	$0{,}5 \cdot 10^6$	$191 \cdot 10^6$	–	166 K
BEE	3,43 GB/s	4,26 GB/s	$5{,}5 \cdot 10^6$	$8800 \cdot 10^6$	–	692 K
SEE	1,2 GB/s	2,6 GB/s	$0{,}5 \cdot 10^6$	$800 \cdot 10^6$	–	111 K

Tab. 6-4: Speicherbandbreite, PCNN Größe und Anzahl der ASIC Gatter bzw. FPGA Logic Cells von digitalen Emulationsplattformen

6.4 Bewertung der Implementierung

Anhand der Leistungsbewertung von Halb-Kunden-Entwurf und FPGA basierten Emulationsplattformen wird nun der Versuch unternommen, die Implementierung bzw. die Systemarchitektur von SEE zu bewerten. Hierfür wird ein Effizienzvergleich von allen Emulationsplattformen, die in Tab. 6-4 aufgelistet sind, durchgeführt und zusätzlich die Flexibilität herausgestellt, die von der Emulationsplattform SEE bereitgestellt wird.

6.4.1 Effizienzvergleich

Die Ermittlung der Effizienz erfolgt auf der Basis der Effizienzdefinition einer digitalen integrierten Schaltungsarchitektur nach Gl. 3-6 auf Seite 59.

η für PCNN Emulation

Bei der digitalen PCNN Emulation ist allerdings der Aufwand der Operation bei Verwendung eines dichteren Netzwerks (adaptive Synapsen pro Neuron) größer und demzufolge von der emulierbaren PCNN Größe *G* abhängig. Die Durchsatzrate *D* bei einer digitalen Schaltungsarchitektur entspricht der Datenübertragungsrate oder Speicherbandbreite, die bei einer PCNN Emulation für die Neuronenzustandsberechnung genutzt werden kann. Die Chip-Fläche einer digitalen Schaltungsarchitektur stimmt im Grunde genommen mit der Anzahl an Gattern überein, die für die HW-Implementierung (in Form von ASICs oder FPGAs) bei der Beschleunigung der PCNN Emulation verwendet wurden. Der Beschleunigungsfaktor (siehe auch N_{10} in Gl. 3-9 und Gl. 3-10 auf Seite 59) einer Emulationsplattform bleibt bei der Ermittlung der Effizienz unberücksichtigt, da jede Emulationsplattform von ihrer beschleunigenden HW-Implementierung (insbesondere bei Halb-Kunden-Entwürfen) auf das zu beschleunigende Neuronenmodell abgestimmt ist. Basierend auf der Gl. 3-6 lässt sich die Effizienz η_{PCNN} von digitalen Plattformen für die PCNN Emulation wie folgt formulieren:

$$\eta_{PCNN} = \frac{G \cdot D}{A} \qquad \textbf{(6-11)}$$

Für den Effizienzvergleich werden die Daten aus Tab. 6-4 herangezogen, wobei eine Umrechnung von FPGA Logic Cells nach ASIC Gattern erfolgen muss. Wie bereits in Abschn. "Flächenbedarf" auf Seite 111 erwähnt, entspricht nach [Xili97] eine Logic Cell im FPGA von Xilinx etwa 12 ASIC Gattern. In der Tab. 6-4 sind bei den auf Halb-Kunden-Entwurf basierten Plattformen (NESPINN, MASPINN und SP²INN) die Gatterzahlen angegeben, die auch tatsächlich verwendet werden. Bei den FPGA basierten Plattformen (SPIKE128K, RAPTOR, BEE und SEE) allerdings sind alle zur Verfügung stehenden Logic Cells angegeben, die jedoch nicht alle verwendet werden. Die Flexibilität bei der HW-Implementierung mit FPGAs besteht auch darin, ausreichend Logik-Ressourcen zur Verfügung zu stellen, die eventuell gar nicht genutzt werden. Um diesen Umstand bei den FPGA basierten Plattformen zu berücksichtigen, wird davon ausgegangen, dass maximal 80 % der insgesamt zur Verfügung stehenden Logik-Ressourcen der FPGAs tatsächlich genutzt sind. Dies zeigt auch die Logik-Ressourcen-Nutzung bei der Emulationsplattform SEE. Nur das NSC-FPGA verwendet 80 % der Logik-Ressourcen des FPGAs, die anderen beiden FPGAs nutzen etwa die Hälfte (siehe “Number of SLICEs” in Tab. 5-5 auf Seite 111). Die ungenormten Effizienzwerte *E* sind nach Gl. 6-11 für die Emulationsplattformen aus Tab. 6-4 in Gl. 6-12 dargestellt.

$$E_{NESPINN} = \frac{320 \cdot 0,6}{120} = 1,6 \qquad E_{SPIKE128K} = \frac{261 \cdot 0,28}{(0,8 \cdot 12) \cdot 5,8} = 1,31$$

$$E_{MASPINN} = \frac{32 \cdot 0,8}{100} = 0,27 \qquad E_{RAPTOR} = \frac{382 \cdot 1,56}{(0,8 \cdot 12) \cdot 166} = 0,37 \qquad \textbf{(6-12)}$$

$$E_{SP^2INN} = \frac{63,1 \cdot 0,45}{250} = 0,11 \qquad E_{BEE} = \frac{1600 \cdot 3,43}{(0,8 \cdot 12) \cdot 692} = 0,83$$

$$E_{SEE} = \frac{1600 \cdot 1,2}{(0,8 \cdot 12) \cdot 111} = 1,8$$

Der Faktor *(0,8·12)* bei der Gatteranzahl der FPGA basierten Plattformen berücksichtigt, dass eine Logic Cell 12 ASIC Gattern entspricht und dass 80 % der zur Verfügung stehenden Logic Cells im FPGA für die HW-Implementierung verwendet werden.

In Abb. 6-10 sind die (auf E_{SEE} in Gl. 6-12) genormten Effizienzen η aller Emulationsplattformen nochmals grafisch zusammengefasst.

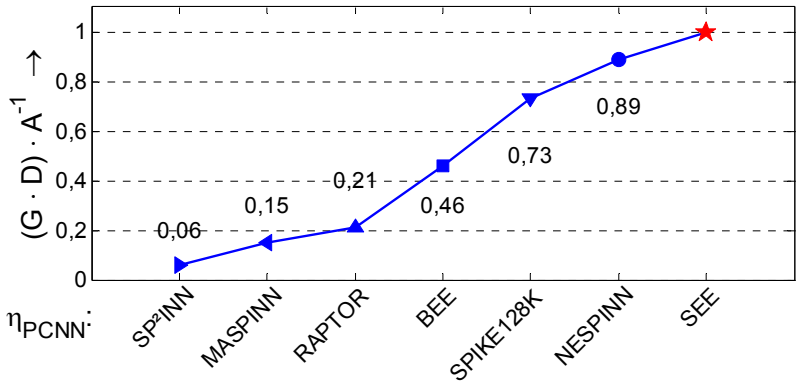


Abb. 6-10: Effizienzvergleich von digitalen Emulationsplattformen

Insgesamt kann festgestellt werden, dass die Effizienz von Emulationsplattformen, wie NESPINN und SEE, am höchsten ist, wenn durch parallele Datenkanäle eine Erhöhung der Speicherbandbreite erzielt wird ohne dabei im gleichen Maße die Anzahl der Gatter für die HW-Implementierung zu erhöhen. Die digitale Emulationsplattform SEE stellt dabei die effizienteste Lösung für eine PCNN Emulation dar, wenn man die verfügbare Speicherbandbreite für die Neuronenzustandsberechnung, das Verhältnis von Synapsen zu Neuronen und die verwendete Gatteranzahl berücksichtigt.

6.4.2 Flexibilitätsbereitstellung

Die Emulationsplattform SEE stellt im Vergleich zu anderen FPGA basierten Emulationsplattformen ein hohes Maß an Flexibilität bereit, die sich nicht nur auf die Rekonfigurierbarkeit von HW-Modulen beschränkt, sondern auch die Programmierbarkeit von SW-Modulen für die Ablaufsteuerung der Emulation und die Beschreibung einer PCNN Architektur umfasst.

GUI

Anhand der in Kapitel 5.5.1 auf Seite 122 bereits vorgestellten graphischen Benutzerumgebung für die Netzwerkbeschreibung und -konfiguration können Software-Prozeduren mit der Skriptsprache Perl/Tk innerhalb des GUIs modifiziert oder hinzugefügt werden. Dadurch ist es möglich, Anpassungen von bestehenden Netzwerkarchitekturtypen, wie zusätzliche reguläre Verbindungsstrukturen bei einer Kaskaden-Architektur oder Verarbeitungsschichten bei einer Gabor-Architektur, vorzunehmen oder aber auch neue Netzwerkarchitekturtypen zu definieren. Bei der Neuronen- und Synapsenmodellierung können dadurch auch mit geringem Programmieraufwand zusätzliche Neuronenparameter und neue Adaptionsregeln für Synapsen berücksichtigt werden.

2 programmierbare PPCs

Die Modifikation der SW-Module, die mit der Programmiersprache C beschrieben sind und von den beiden PPCs innerhalb des PPC2-FPGAs ausgeführt werden, erlaubt die Änderung des Emulationsablaufs. Dies kann erforderlich werden, wenn eventuell ein anderes Neuronenmodell emuliert werden soll, das eine andere Klassifizierung von Neuronen erfordert. Somit kann es unter Umständen zur Einführung weiterer Ereignislisten kommen, die zusätzlich verwaltet werden müssen. Weiterhin kann man mit SW-Modulen sehr viel flexibler die Protokollierung von weiteren Netzwerkparametern, wie einzelne Synapsengewichtsverläufe innerhalb des PCNNs, steuern. Ohne einen Aufwand an der Steuerung der Hardware-Module zu betreiben, müssen von der Software lediglich am Ende der Bearbeitung eines Emulationsereignisses bestimmte Speicherzellen ausgelesen werden, an denen sich die zu protokollierenden Netzwerkparameter befinden. Bei dem Architekturkonzept von SEE wurde ebenfalls darauf geachtet, dass zu den externen Speichern jeweils Standardbusbreiten von 32 Bit oder 64 Bit verwendet werden, damit diese auch von den PPCs, die eine 32 Bit-Architektur haben, effizient angesteuert werden können. Architekturkonzepte, die Busbreiten wie 24 Bit bei NESPINN oder 36 Bit bei SP²INN berücksichtigen, sind für die Implementierung eines effizienten Hardware/Software-Systems eher ungeeignet. Bei 24 Bit könnte ein weiteres Byte von der Software ausgewertet werden, das nicht zur Verfügung steht, und bei 36 Bit müssen für die Auswertung stets 2 Lese- oder Schreiboperationen erfolgen.

Konfigurierbare NTC/NSC

Die Konfigurierbarkeit der Hardware-Architektur für die Netzwerktopologie- und die Neuronenzustandsberechnung bietet die Möglichkeit, je nach durchzuführender PCNN Emulation, die optimalste Implementierung zur Verfügung zu stellen. Bei der Neuronenzustandsberechnung beispielsweise kann das NSC-FPGA mit 2 unterschiedlichen PE-Architekturen, die entweder nur einen oder alle 3 Speicherkanäle nutzen, konfiguriert werden, um eine effiziente PCNN Emulation bei spärlichen und dichten Vernetzungsstrukturen durchführen zu können. Bei der Netzwerktopologieberechnung können durch konfigurierbare HW-Architekturen zusätzliche HW-Module integriert werden, um weitere reguläre Verbindungsstrukturen, die sich für die Detektionen anderer Bildmerkmale als sinnvoll erweisen, berücksichtigen zu können. Die eigentliche Zielsetzung dieser Rekonfigurierbarkeit der Hardware ist, dass sich sowohl das NTC- als auch das NSC-FPGA aus einer Bibliothek von HW-Modulen bedient, die speziell für die digitale PCNN Emulation entwikkelt worden sind. Diese könnte man dann nach dem Baukastenprinzip zusammenstellen, um aufgrund der erstellten Netzwerkbeschreibung die effizientesten HW-Architekturen für die Netzwerktopologie- und die Neuronenzustandsberechnung zu generieren. Dieses Prinzip der Wiederverwendbarkeit von HW-Modulen (*Design Reuse*) wird immer noch nicht effizient in die Praxis umgesetzt, aber gewinnt bei immer komplexer werdenden HW-Architekturen immer mehr an Bedeutung für den integrierten digitalen Schaltungsentwurf [Keat02]. Auf der Prototypen-Platine stehen außerdem für den flexiblen Anschluss von

Zusatzplatinen 98 Testanschlüsse vom NTC-FPGA zur Verfügung, mit denen zusätzliche Speicher oder sonstige Schaltungsteile angesteuert werden können.

Im folgenden wird anhand von protokollierten Netzwerkparametern versucht, einen Einblick in die Netzwerkdynamik des verwendeten Neuronenmodells mit adaptiven Synapsengewichten zu geben.

6.5 Evaluation des Neuronenmodells mit adaptiven Synapsengewichten

Das Neuronenmodell mit adaptiven Synapsengewichten zeichnet sich durch die Eigenschaft aus, ein synchrones Pulsverhalten bei benachbarten Neuronen zu erzeugen, wenn diese im Rahmen einer Vierer-Nachbarschaftsverbindung untereinander vernetzt sind. Dieses Pulsverhalten eignet sich in der Bildvorverarbeitung für eine Objekt-Hintergrund-Trennung [Eckh99] bzw. für eine Fleck-Detektion [Heit02]. Für die Bewertung der Netzwerkdynamik, die auf dem dynamischen Verhalten der adaptiven Synapsengewichte basiert, wurden zahlreiche PCNN Simulationen durchgeführt. Dabei wurden, wie bei den durchgeführten Simulationen in Kapitel 6.1 auf Seite 127, die gleichen Anfangsbedingungen verwendet, bei denen die Membranpotentiale a_K der Neuronen mit zufälligen Werten zwischen 0 und 1 und die Synapsengewichte W_{KL} und W_0 mit dem Wert 0,12 ms^{-1} initialisiert wurden.

Korreliertes Pulsverhalten

Ein synchrones bzw. korreliertes Pulsverhalten stellt sich bei einer Eingangsschicht, in der die Neuronen lateral mit ihren nächsten 4 Nachbarneuronen verbunden sind, nach einer Netzwerkzeit von etwa 150 ms ein. Dies ist in Abb. 6-11b zu erkennen, in der die 4 Pulse von präsynaptischen Neuronen (X_{L0}, X_{L1}, X_{L2} und X_{L3}) und die Pulse des postsynaptischen Neurons (X_K) dargestellt sind. Das postsynaptische Neuron N_{209} generiert immer dann einen Puls, wenn das Membranpotential a_K (siehe Abb. 6-11c) des Neurons den Schwellwert von $\theta = 1$ erreicht hat. Alle 5 Neuronen generieren innerhalb von 100 ms 10 Pulse, was einer Pulsfrequenz von 100 Hz entspricht. Dies bedeutet, dass alle 5 Neuronen von starken Eingangsreizen stimuliert werden, da die maximale Pulsfrequenz bei einem maximalen Eingangsreiz (I_K = 1 und W_0 = 0,12 ms^{-1}) 120 Hz betragen kann (siehe auch Gl. 4-1 auf Seite 70). Die Neuronen befinden sich somit innerhalb eines hellen Flecks, der durch ein korreliertes Pulsverhalten detektiert werden soll.

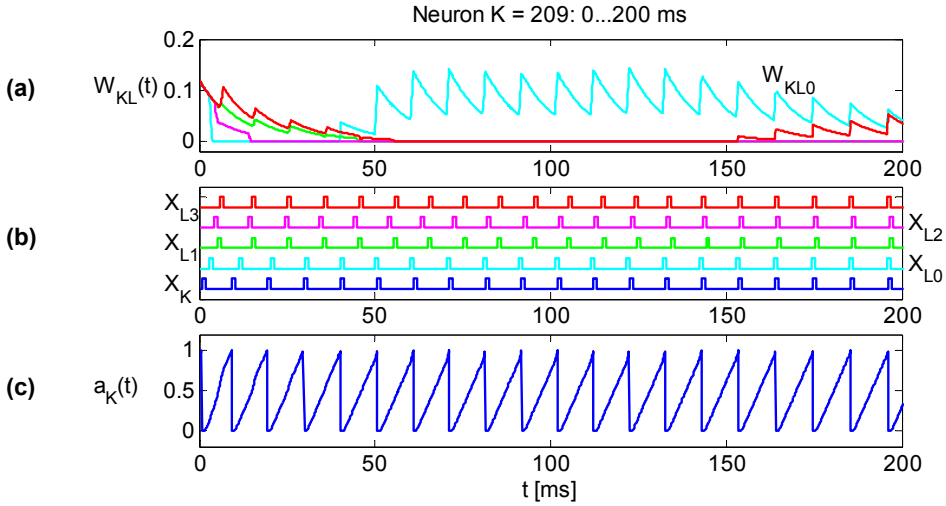


Abb. 6-11: Zeitverläufe von Neuron 209: (a) adaptive Synapsengewichte, (b) präsynaptische Pulse X_{L0}...X_{L3} und postsynaptischer Puls X_K, (c) Membranpotential des postsynaptischen Neurons

Die Verläufe der adaptiven Synapsengewichte sind in Abb. 6-11a dargelegt. Man erkennt, dass in den ersten 50 ms alle Synapsengewichte zu den präsynaptischen Neuronen abklingen und danach das Synapsengewicht W_{KL0} verstärkt wird, um das Pulsverhalten zwischen dem verbundenen präsynaptischen Neuron und dem postsynaptischen

Neuron N_{209} zu synchronisieren. In Abb. 6-12 ist der zeitliche Verlauf der Synapsengewichte über eine Netzwerkzeit von 1000 ms aufgetragen. Es treten dabei kurzeitige Verstärkungen der Synapsengewichte W_{KL0}, W_{KL3} und W_{KL1} auf, die jeweils eine Korrelation der Pulse zum postsynaptischen Neuron bewirken. Wenn alle Neuronen voll synchron pulsen, stellt sich ein stabiler Zustand ein, in dem alle Synapsengewichte gegen den Minimalwert $W_{KLmin} = 0$ konvergieren. Es findet somit kein nennenswerter verstärkender oder hemmender Effekt der Synapsen mehr statt, da sich zu den Zeitpunkten von präsynaptischen Pulsen das postsynaptische Neuron ebenfalls im Pulszustand befindet. Die Bedingung, dass sich für eine Verstärkung bzw. Hemmung der Synapse das präsynaptische Neuron im Sendezustand ($X_L = 1$) und das postsynaptische Neuron im Empfangszustand ($X_K = 0$) befinden müssen, ist also nicht mehr erfüllt (siehe auch Gl. 2-15 auf Seite 20).

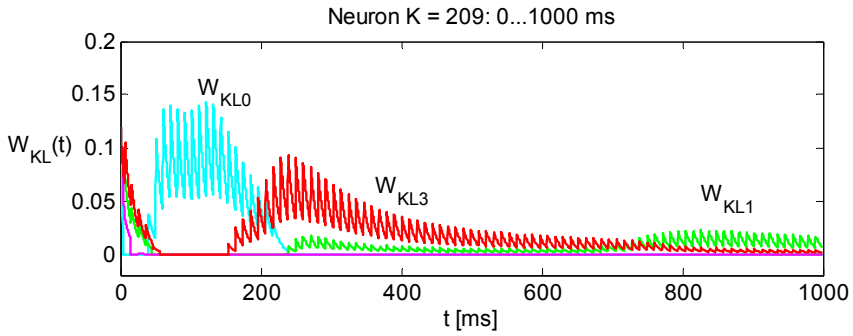


Abb. 6-12: Zeitverläufe der adaptiven Synapsengewichte von Neuron 209

Inhibitorischer Effekt

Wenn verbundene Neuronen mit unterschiedlich starken Eingangsreizen stimuliert werden, befinden sich diese am Übergang von einem Fleck zum Hintergrund. Die Neuronen pulsen mit unterschiedlichen Frequenzen, da sie mit unterschiedlichen Eingangsreizen stimuliert werden (siehe Abb. 6-13b). In diesen Fällen ist der hemmende Effekt von Synapsengewichten zu beobachten, bei denen das präsynaptische Neuron mit einer etwa doppelt so hohen Pulsfrequenz pulst wie das postsynaptische Neuron. In Abb. 6-13a bewirkt der präsynaptische Puls X_{L3} bei 220 ms (↑), dass sich das zugehörige Synapsengewicht W_{KL3} zunächst erhöht, da sich das Membranpotential des postsynaptischen Neurons N_{237} nah am Schwellwert befindet. Bei der Netzwerkzeit von 230 ms (↓) tritt nun ein inhibitorischer Effekt für das Synapsengewicht W_{KL3} ein, da sich das Membranpotential des Neurons N_{237} unterhalb des halben Schwellwertes befindet, und der Verstärkungsterm mit dem Verstärkungsfaktor μ aus Gl. 2-15 negativ wird.

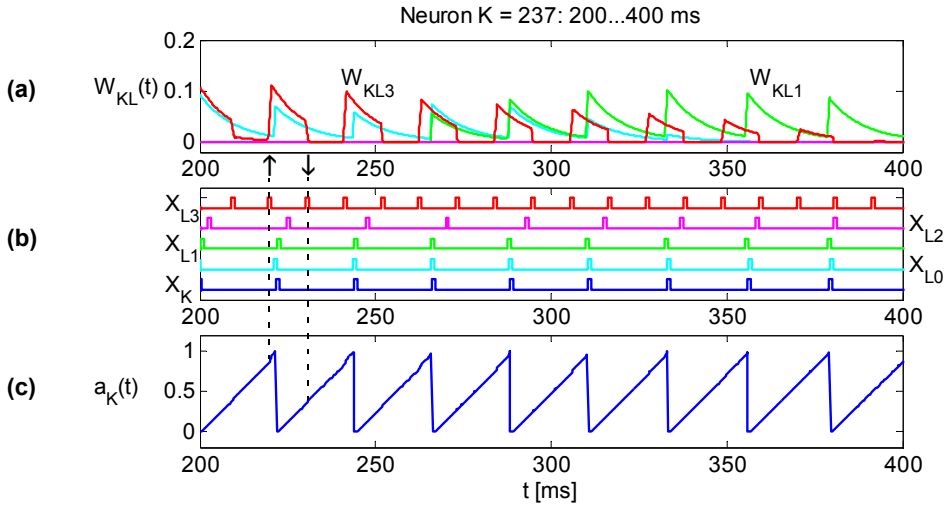


Abb. 6-13: Zeitverläufe von Neuron 237: (a) adaptive Synapsengewichte, (b) präsynaptische Pulse $X_{L0}...X_{L3}$ und postsynaptischer Puls X_K, (c) Membranpotential des postsynaptischen Neurons

Wenn man sich wieder den entsprechenden zeitlichen Verlauf der Synapsengewichte über eine Netzwerkzeit von 1000 ms in Abb. 6-14 ansieht, erkennt man, dass das Synap-

sengewicht W_{KL3} keinen stabilen Zustand erreicht. Diese Netzwerkdynamik spiegelt die Tatsache wieder, dass die Neuronen im Fleck und die Neuronen im Hintergrund unterschiedliche Pulsverhalten haben, die nicht miteinander korreliert werden können.

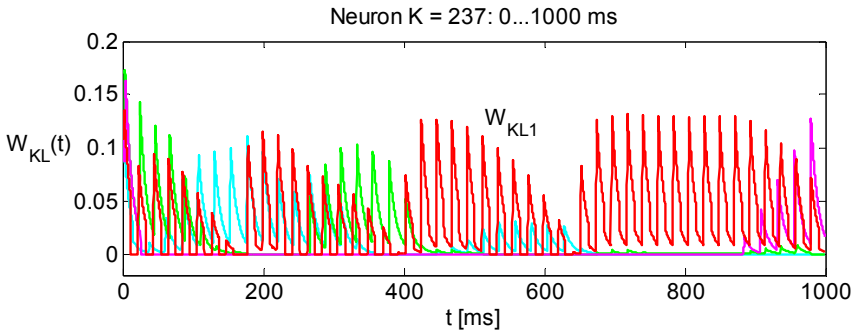


Abb. 6-14: Zeitverläufe der adaptiven Synapsengewichte von Neuron 237

Unkorreliertes Pulsverhalten

Ein unkorreliertes Pulsverhalten weisen auch Neuronen auf, die sich ausschließlich im Hintergrund eines Bildes befinden und mit ähnlich schwachen Eingangsreizen stimuliert werden. In Abb. 6-15b ist der zeitliche Verlauf von 4 präsynaptischen Pulsen und vom Puls des Neurons N_{56} dargestellt. Aufgrund des schwächeren Eingangsreizes generieren die Neuronen jeweils 4 bis 5 Pulse innerhalb von 100 ms, was zu einer Pulsfrequenz von 40 Hz bis 50 Hz führt. In Abb. 6-15a sind ab 500 ms exzitatorische Effekte des Synapsengewichts W_{KL2} zu erkennen, die versuchen, die Pulse des postsynaptischen Neurons (X_K) mit den Pulsen des zugehörigen präsynaptischen Neurons (X_{L2}) zu synchronisieren.

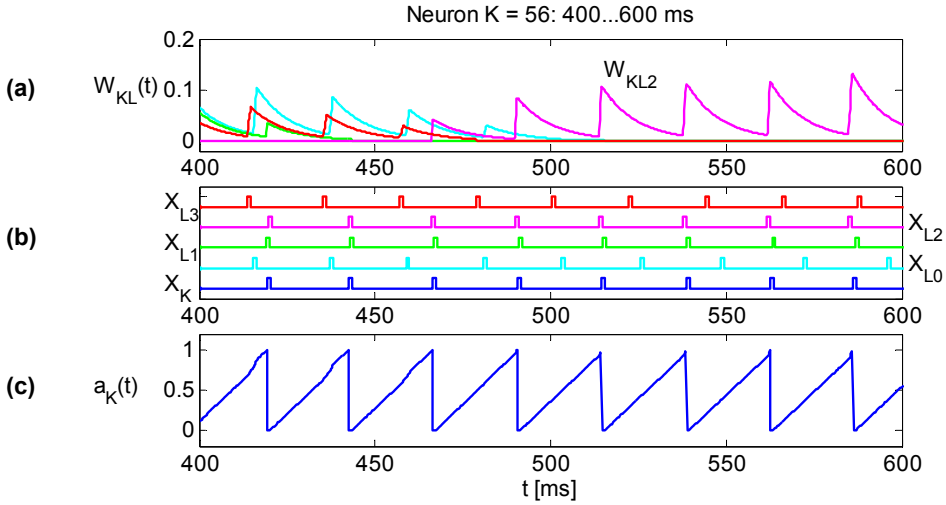


Abb. 6-15: Zeitverläufe von Neuron 56: (a) adaptive Synapsengewichte, (b) präsynaptische Pulse X_{L0}...X_{L3} und postsynaptischer Puls X_K, (c) Membranpotential des postsynaptischen Neurons

Ein synchrones Pulsverhalten der Neuronen, die mit einem schwachen Eingangsreiz stimuliert werden, stellt sich aber nicht ein. Dies wird auch in Abb. 6-16 deutlich, in der wieder der zeitliche Verlauf der mit dem Neuron N_{56} verbundenen adaptiven Synapsengewichte über eine Netzwerkzeit von 1000 ms dargestellt ist. Alle Synapsengewichte der präsynaptischen Neuronen sind gleichermaßen, bezüglich ihres adaptiven Verhaltens, aktiv. Die Synapsen werden wechselweise über einen Zeitraum von bis zu 200 ms verstärkt und klingen dann ab. Auf dem Minimalwert $W_{KLmin} = 0$ verweilen die Synapsen schließlich für eine Zeitdauer von 100 ms bis 300 ms in einer Ruhephase, bevor wieder eine Verstärkungsphase eintritt. Die Verstärkungsphase für ein Synapsengewicht tritt immer dann ein, wenn bei einem präsynaptischen Puls das Membranpotential des postsynaptischen Neurons oberhalb des halben Schwellwertes liegt. Aufgrund der minimal unterschiedlichen Pulsfrequenzen und des unkorrelierten Pulsverhaltens, treten die Pulse des präsynaptischen Neurons zunächst für eine gewisse Zeit immer dann ein, wenn das Membranpotential sich oberhalb des Schwellwertes befindet. Dann erfolgt eine Sequenz

von präsynaptischen Pulsen, wenn das Membranpotential sich unterhalb des Schwellwertes befindet. In dieser Situation klingt das Synapsengewicht ab und bleibt in einer länger andauernden Ruhephase. Im Gegensatz zu Neuronen, die mit starken Eingangsreizen stimuliert werden (siehe Abb. 6-12), stellt sich kein stabiler Zustand ein, in dem die Synapsengewichte gegen den Minimalwert konvergieren. Das resultierende unkorrelierte Pulsverhalten basiert somit auf den periodischen Verstärkungen der Synapsen, die erfolglos versuchen, ein korreliertes Pulsverhalten zum postsynaptischen Neuron zu erreichen.

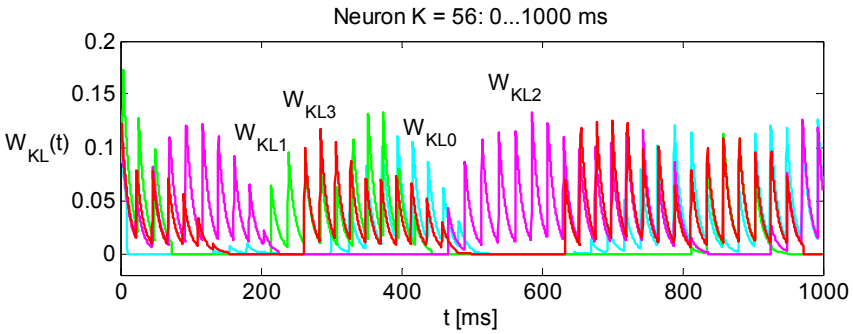


Abb. 6-16: Zeitverläufe der adaptiven Synapsengewichte von Neuron 56

Short-Term Memory

Die exzitatorischen und inhibitorischen Effekte der Synapsen, die für den Lern- und Speicherprozess innerhalb eines PCNNs verantwortlich sind, halten bei dem verwendeten Neuronenmodell jeweils nur für einen kurzen Zeitraum von einigen 100 ms an und haben dadurch eine kurzfristige Wirksamkeit. Die Dynamik der adaptiven Synapsengewichte modelliert somit ein kurzfristiges Speicherverhalten (STM = *Short-Term Memory*). Diese Dynamik stellt hohe Anforderungen an die Performanz einer Simulationsplattform, insbesondere bei der Berücksichtigung einer dichten Vernetzungsstruktur, da hier eine Vielzahl von Synapsen pro Neuron adaptiert werden muss. Diesen Anforderungen kann man nur mit einer erhöhten Speicherbandbreite begegnen. Maßnahmen zur Reduzierung der Speicherbandbreite können nicht genutzt werden, da adaptive Synapsengewichte für jede Aktualisierung des Netzwerks wieder im Parameterspeicher abgelegt werden müssen. Alternativ könnte man bei der Adaption der Synapsengewichte einen Mechanismus berücksichtigen, der es ermöglicht, dass sich exzitatorische und inhibitorische Effekte der Synapsen auch über einen längeren Zeitraum erstrecken (LTM = *Long-Term Memory*). Dies könnte man, wie in [Matt00], mit einem Schwellwert auch für Synapsen realisieren. Wenn sich das Synapsengewicht über einen bestimmten Zeitraum oberhalb bzw. unterhalb dieser Schwelle befindet, wird der verstärkende bzw. hemmende Effekt der Synapse für einen längeren Zeitraum konstant beibehalten, ohne dass in diesem Zeitraum eine Adaption dieses Synapsengewichts erfolgen muss. Dies wäre besonders bei Neuronen sinnvoll, die sich im Fleck eines Eingangsbildes befinden. Die Verläufe der Synapsengewichte bei diesen Neuronen haben in Abb. 6-12 auf Seite 144 gezeigt, dass nach Netzwerkzeiten von 500 ms die Synapsengewichte sich nahezu konstant verhalten. Diese “eingeschwungenen” Synapsengewichte bräuchten bei der Einführung einer längerfristigen Wirksamkeit von Synapsen für die weitere PCNN Simulation nicht mehr berücksichtigt werden. Nur durch solch einen Modellierungsansatz wäre eine Reduzierung der notwendigen Speicherbandbreite möglich.

Untersuchungen zum Verhalten der Netzwerkdynamik bei Variationen der Synapsenparameter μ und γ, sowie unterschiedliche Anfangsbedingungen für Membranpotentiale und Synapsengewichte, sind den Arbeiten in [Sann03] zu entnehmen. Diese Untersuchungen haben bestätigt, dass die von den Autoren in [Heit02] vorgeschlagenen Werte für die Abklingkonstante $\gamma = 0{,}1\ \text{ms}^{-1}$, den Verstärkungsfaktor $\mu = 0{,}3\ \text{ms}^{-2}$ und das Synapsengewicht $W_0 = 0{,}12\ \text{ms}^{-1}$ für die Fleck-Detektion am geeignetsten sind.

Resümee

Im Rahmen dieses sechsten Kapitels wurde eine Leistungsbewertung der digitalen Emulationsplattform SEE durchgeführt. Netzwerksimulationen auf einem Einprozessor-System haben gezeigt, dass für eine 96×96 große Netzwerkschicht mit einer Vierer-Nachbarschaftsverbindung eine Simulationszeit von mehr als 55 Stunden (198079 s) und für

eine 30×30 große Netzwerkschicht mit lateraler Vollvernetzung eine Simulationszeit von knapp 98 Stunden (352460 s) benötigt wird, um eine Netzwerkzeit von 500 ms zu simulieren (siehe Tab. 6-1 auf Seite 131). Bei einer Taktfrequenz von 50 MHz kann SEE für die numerische Integration, die etwa 70 % der Simulationszeit einnimmt, einen Beschleunigungsfaktor von 30 für spärliche Vernetzungsstrukturen, wie Vierer- und Achter-Nachbarschaftsverbindungen, und ein Beschleunigungsfaktor von 8 für dichte Vernetzungsstrukturen (laterale Vollvernetzung) erzielen. Bei der Leistungsbewertung von 3 Emulationsplattformen, die auf Halb-Kunden-Entwürfen beruhen, und 3 weiteren Emulationsplattformen, die auf FPGAs basieren, zeigt sich, dass die Effizienz von Emulationsplattformen am höchsten ist, wenn durch parallele Datenkanäle eine Erhöhung der Speicherbandbreite erzielt wird ohne im gleichen Maße die Anzahl der Gatter für die HW-Implementierung zu erhöhen (siehe auch Abb. 6-10 auf Seite 141).

Die im Rahmen dieser Arbeit konzipierte und realisierte Emulationsplattform SEE stellt sich dabei als effizienteste Plattform für die Emulation eines Netzwerks von pulscodierten Neuronen mit adaptiven Synapsengewichten heraus, wenn man die verfügbare Speicherbandbreite für die Neuronenzustandsberechnung, das Verhältnis von Synapsen zu Neuronen und die verwendete Gatteranzahl berücksichtigt.

Zusammenfassung

Im Rahmen der vorliegenden Arbeit wurde das Architekturkonzept und die Realisierung einer Hardware/Software-Emulationsplattform, die abgekürzt mit SEE (*Spiking Neural Network Emulation Engine*) bezeichnet wird, für die Simulationsbeschleunigung von pulscodierten neuronalen Netzen mit adaptiven Synapsengewichten vorgestellt. Die Durchführung einer PCNN Emulation (siehe Kapitel 5.5 auf Seite 121) erfordert im Vorfeld eine Netzwerkbeschreibung sowie eine Netzwerkkonfiguration. Im Anschluss an eine durchgeführte PCNN Emulation erfolgt die Visualisierung von Pulsereignissen, um experimentell das Pulsverhalten von Neuronen des beschriebenen Netzwerks zu untersuchen (siehe Abb. 7-1).

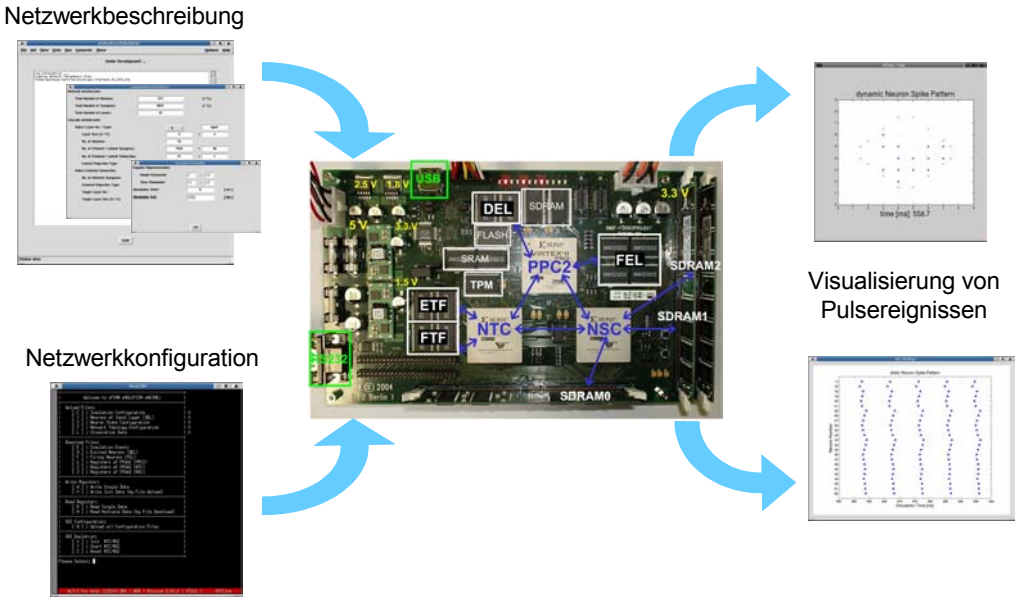


Abb. 7-1: PCNN Emulation mit der digitalen Emulationsplattform SEE

7.1 Ergebnisse

Experimentierplattform

Die digitale Emulationsplattform SEE stellt eine Experimentierplattform für den ganzheitlichen Bildverarbeitungsprozess dar. Der ganzheitlichen Bildverarbeitungsprozess mit pulscodierten Neuronen umfasst 2 unterschiedliche visuelle Verarbeitungspfade, die als ventraler und als dorsaler Pfad bezeichnet werden (siehe auch Abb. 2-15 auf Seite 25). Der ventrale Pfad verarbeitet die "Was"-Informationen eines visuellen Objektes und ist für die Identifikation von visuellen Elementarmerkmalen (*Feature-Extracting*) zuständig. Bei diesem Verarbeitungsprozess, der auch als Bildvorverarbeitungsprozess bezeichnet wird, benötigen die Neuronen lediglich eine spärliche Verbindungsstruktur (siehe auch Abschn. "2 Verbindungsstrukturen" auf Seite 29). Das Verhalten der Synapsen ist hierbei eher statisch (reduzierte Synapsenplastizität). Der dorsale Pfad verarbeitet die "Wo"-Informationen eines visuellen Objektes und führt die Verknüpfung von visuellen Elementarmerkmalen (*Assembly-Forming*) durch. Dieser Verarbeitungsprozess erfordert

Neuronen mit einer dichten Verbindungsstruktur, und das Verhalten der Synapsen ist sehr dynamisch (erhöhte Synapsenplastizität). Aufgrund einer möglichen dichteren Verbindungsstruktur wurde bei SEE eine hohe Speicherbandbreite von 1,2 GB/s durch 3 parallel nutzbare Datenkanäle zum Parameterspeicher, der über eine hohe Speicherkapazität von 3 GB verfügt, realisiert. In diesem Parameterspeicher sind sowohl adaptive Synapsengewichte als auch Membranpotentiale gemeinsam abgelegt und nach postsynaptischen (Puls-empfangenden) Neuronen in Parameterblöcken geordnet. Existierende Plattformen für die PCNN Emulation, wie BIONIC, SYNAPSE, CNAPS (siehe Kapitel 3.4.3 auf Seite 48), NESPINN, MASPINN, SP²INN (siehe Kapitel 3.4.4 auf Seite 50), SPIKE128K oder RAPTOR (siehe Kapitel 3.4.5 auf Seite 53), realisieren ausschließlich einen Bildvorverarbeitungsprozess, da als visueller Verarbeitungspfad nur der ventrale Pfad berücksichtigt wird. Weiterhin wird bei den meisten Plattformen der Parameterspeicher für Membran- und Dendritenpotentiale mit mehreren Datenkanälen und der Gewichtsspeicher für Synapsengewichte immer nur mit einem einzigen Datenkanal realisiert. Zusätzlich findet bei den meisten Plattformen (mit Ausnahme von SP²INN) stets eine Trennung von Parameter- und Gewichtsspeicher statt. Die Realisierung einer Experimentierplattform für den ganzheitlichen Bildverarbeitungsprozess erforderte die Neuentwicklung einer Prototypenplatine für SEE, da existierende Rapid-Prototyping- und ASIC-Prototyping-Plattformen aufgrund der ungeeigneten Balance von erzielbarer Operationsfrequenz, nutzbarer Speicherbandbreite, verfügbarer Speicherkapazität und FPGA-Ressourcen (siehe auch Kapitel 4.1 auf Seite 61) nicht verwendet werden konnten.

Architekturkonzept

Beim Architekturkonzept von SEE stand neben der Realisierung einer Experimentierplattform für den ganzheitlichen Bildverarbeitungsprozess die Bereitstellung einer hohen Flexibilität bei der Implementierung im Vordergrund. Diese Flexibilitätsbereitstellung wurde durch 3 Maßnahmen im Rahmen des Architekturkonzeptes realisiert:

1. FPGA basierte heterogene SIMD/MIMD-Architektur (siehe auch Kapitel 1.1 auf Seite 1): Eine SIMD-Architektur ist für die Implementierung des ventralen Verarbeitungspfades geeignet, da bei der Identifikation von visuellen Elementarmerkmalen eine hohe Datenparallelität vorliegt. Notwendige Low-Level Operationen (siehe auch Abschn. "Low-Level/High-Level Operationen" auf Seite 31 und Kapitel 3.3.2 auf Seite 41), wie die Aktualisierung der Membranpotentiale und Synapsengewichte, können in FPGAs aufgrund der Rekonfigurierbarkeit flexibel parallelisiert werden. Für eine effiziente Parallelverarbeitung wurden Parameter- und Gewichtsspeicher in einem gemeinsamen Speicher mit 3 Datenkanälen in Form von Neuroneninformationsblöcken (NIBs) zusammengefasst. Eine MIMD-Architektur eignet sich für die Implementierung des dorsalen Verarbeitungspfades. Bei der Verknüpfung von visuellen Elementarmerkmalen liegt eine funktionale Parallelität vor, die die serielle Durchführung von High-Level Operationen, wie die Assoziation von bekannten visuellen Elementarmerkmalen, erfordert. Diese Operationen können sehr flexibel von Softwareprogrammierbaren Prozessoren ausgeführt werden, die parallel unabhängig voneinander arbeiten und sich innerhalb der FPGAs befinden.
2. Effiziente Kommunikation zwischen Netzwerktopologie- und Neuronenzustandsberechnung anhand von Topologie-Vektoren (siehe auch Abb. 4-6 auf Seite 68): Anhand von Topologie-Vektoren übermittelt die Netzwerktopologieberechnung die Pulszustände von präsynaptischen (Puls-sendenden) Neuronen an die Neuronenzustandsberechnung. Der Pulszustand eines präsynaptischen Neurons, der mit einem Bit codiert wird, ist für die Aktualisierung eines Synapsengewichts eines postsynaptischen Neurons notwendig. Erst nach Aktualisierung der Synapsengewichte kann das aktuelle Membranpotential eines postsynaptischen Neurons ermittelt werden. Durch dieses Kommunikationsprotokoll findet eine echte Trennung zwischen diesen beiden Berechnungsarten statt, da die Netzwerktopologieberechnung nicht in die Neuronenmodellierung durch Adaption von Synapsengewichten involviert ist. Die Implementierung für die Netzwerktopologieberechnung ist somit flexibler einsetzbar, da sie unabhängig von der Neuronenmodellierung ist. Existierende Plattformen, die ebenfalls

eine getrennte Berechnung von Netzwerktopologie und Neuronenzuständen vornehmen, kommunizieren über Synapsengewichte (siehe NESPINN in Kapitel 3.4.4.1 auf Seite 50) oder Synapsengewichtsänderungen (siehe MASPINN in Kapitel 3.4.4.2 auf Seite 51). Für die Adaption von Synapsengewichten ist diese Kommunikation nachteilig, da ein erhöhter Datentransfer von der Netzwerktopologieberechnung zur Neuronenzustandsberechnung notwendig ist (ein Synapsengewichtswert kann nicht mit einem einzigen Bit codiert werden). Darüber hinaus ist ein zusätzlicher Datentransfer in umgekehrter Richtung von der Neuronenzustandsberechnung zur Netzwerktopologieberechnung erforderlich, da für die Aktualisierung der Synapsengewichte die Membranpotentialwerte von postsynaptischen Neuronen benötigt werden (siehe Gl. 2-15 auf Seite 20).

3. Hardware-Beschleunigung eines numerischen Integrationsverfahrens (siehe auch Kapitel 4.3 auf Seite 72): Bei SEE wird zum ersten Mal der Versuch unternommen, ein numerisches Integrationsverfahren [Stoe00] für die Hardware-Beschleunigung bei der PCNN Emulation zu realisieren. Der Vorteil eines numerischen Integrationsverfahrens liegt in der Flexibilität, wenn Änderungen bezüglich der Neuronenmodellierung erfolgen müssen oder alternative Neuronenmodelle verwendet werden sollen. Außerdem ist die Berücksichtigung von komplexeren Neuronenmodellen möglich, die eine exakte bzw. analytische Berechnung der Neuronenzustände nicht erlauben, da unterschiedliche Differentialgleichungen für Membranpotentiale und Synapsengewichte verwendet werden (siehe auch Abschn. "Analytisch/Numerisch" auf Seite 35). Alle existierenden Plattformen realisieren eine analytische Berechnung der Neuronenzustände. Dies ist nur bei einer einfachen Neuronenmodellierung möglich, die beispielsweise nur eine Differentialgleichung für die Membranpotentialfunktion vorsieht (siehe Integrate-and-Fire Modell in Kapitel 2.2.1 auf Seite 16).

FPGA basierte PCNN Emulation

Das Architekturkonzept von SEE wurde mit 3 FPGAs realisiert, die als PPC2-FPGA, NTC-FPGA und NSC-FPGA bezeichnet sind. Die Aufteilung von Neuronenzustandsberechnung (im NSC-FPGA) und der Netzwerktopolgieberechnung (im NTC-FPGA) hat sich bei existierenden Emulationsplattformen als vorteilhaft erwiesen. Die Synchronisation dieser beiden Berechnungen erfolgt anhand von Ereignis- bzw. Neuronenlisten (im PPC2-FPGA).

1. Das PPC2-FPGA (XCVP20-5FF1152) ist für die Emulationsablaufsteuerung zuständig und beinhaltet 2 eingebettete PPCs (PowerPC-Prozessoren). Die Emulationsablaufsteuerung umfasst dabei die folgenden 3 Aufgaben: 1.) Konfiguration des Netzwerks vor dem Emulationsstart über externe serielle Schnittstellen, 2.) Protokollierung von Netzwerkparametern und Pulsereignissen, und 3.) Verwaltung von Ereignislisten. Es müssen während der Emulation 2 Ereignislisten verwaltet werden: die dynamische Ereignisliste (DEL = *Dynamic Event List*) und die Feuer-Ereignisliste (FEL = *Fire Event List*). Die DEL erfasst alle erregten Neuronen, die zum betrachteten Emulationszeitpunkt einen Puls oder einen externen Reiz empfangen. Die FEL berücksichtigt alle Neuronen, die sich im Pulszustand befinden, und den dazugehörigen Zeitpunkt, an dem das Neuron wieder den Pulszustand verlässt. Das PPC2-FPGA benötigt 4345 Slices und 54 interne BRAM-Speicher (54·2 KB = 108 KB) an Hardware-Ressourcen.
2. Die Netzwerktopologieberechnung wird vom NTC-FPGA (XCV4000-4FF1152) durchgeführt und besteht aus 2 Aufgaben: 1.) Ermittlung der präsynaptischen Pulsaktivität eines erregten postsynaptischen Neurons, und 2.) Berechnung der postsynaptischen Neuronen, die durch einen neuen Puls eines präsynaptischen Neurons in den Erregungszustand gelangen. Für diese Berechnungen werden 2 Markierungsspeicher verwendet: der Feuer-Tag-Speicher (FTF = *Fire Tag-Field*) und der Erregungs-Tag-Speicher (ETF = *Excitation Tag-Field*). Das FTF ist eine komprimierte Darstellung der FEL und wird für die Berechnung der präsynaptischen Pulsaktivität benötigt. Das ETF stellt eine komprimierte Version von FEL und DEL dar und wird bei der Berechnung der postsynaptischen Neuronen gebraucht. Mit Hilfe der Markierungen im ETF wird verhindert, dass Neuronen, die sich bereits im Puls- bzw. Erregungszustand befinden,

als erregte postsynaptische Neuronen mehrfach in die DEL eingetragen werden. Die Hardware-Ressourcen, die das NTC-FPGA nutzt, sind 11615 Slices, 19 Multiplizierer-Blöcke und 39 interne BRAM-Speicher (39·2 KB = 78 KB).

3. Das NSC-FPGA (XCV4000-4FF1152) führt für die Zustandsberechnung von postsynaptischen Neuronen ein numerisches Integrationsverfahren durch. Während der Fließband- und Parallelverarbeitung der numerischen Integration wird auf 3 notwendige Informationsquellen zeitgleich zugegriffen: 1.) Erregte Neuronen in der DEL, bei denen der Neuronenzustand über das PPC2-FPGA aktualisiert werden muss, 2.) Präsynaptische Pulsaktivität in Form von Topologie-Vektoren über das NTC-FPGA, und 3.) Membranpotentiale und Synapsengewichte von verbundenen präsynaptischen Neuronen, die über den Parameterspeicher (NIB = *Neuron Information Block*) zugänglich sind. Die vom NSC-FPGA verwendeten Hardware-Ressourcen sind 18788 Slices, 40 Multiplizierer-Blöcke und 88 interne BRAM-Speicher (88·2 KB = 176 KB).

Für die Beschleunigung der PCNN Emulation werden bei SEE insgesamt 5 Maßnahmen ergriffen:

1. Klassifizierung der Neuronen anhand der Ereignislisten DEL und FEL
   ⇒ Erhöhung des Wirkungsgrads bei der PCNN Emulation (es werden nur relevante Neuronen während der Emulation betrachtet)
2. Parallelverarbeitung des Pulsereignis-basierten Emulationsablaufs
   ⇒ Parallelisierung von High-Level Operationen (die Netzwerktopologie- und Neuronenzustandsberechnungen erfolgen parallel)
3. Parallelisierung des numerischen Integrationsverfahrens
   ⇒ Parallelisierung von Low-Level Operationen (durch den Einsatz mehrerer HW-Integratoren wird das numerische Integrationsverfahren parallelisiert)
4. 3 parallel nutzbare Datenkanäle à 64 Bit zum Parameterspeicher
   ⇒ Bereitstellung einer hohen Speicherbandbreite (somit können auch Netzwerke mit dichter Vernetzungsstruktur emuliert werden)
5. Autarkie der Emulationsplattform
   ⇒ Bereitstellung ausreichender Speicherkapazität (es ist keine Kommunikation zu einem Host-PC während der Emulation notwendig)

Performanz

Die maximal emulierbare PCNN Größe von SEE beläuft sich auf $0{,}5 \cdot 10^6$ Neuronen und auf $800 \cdot 10^6$ adaptiven Synapsen. Die Anzahl der Neuronen ist durch den 2 MB großen SRAM-Speicher des PPC2-FPGAs, der für die DEL vorgesehen ist, begrenzt. Die Anzahl der adaptiven Synapsen ist durch die Verwendung der 1 GB großen SDRAM-DIMM-Speicher, die vom NSC-FPGA angesteuert werden, limitiert. Diese PCNN Größe ist für die Modellierung von einzelnen Gehirnarealen ausreichend, die die Berücksichtigung von 10^5 Neuronen und 10^8 Synapsen erfordert [Reut02]. Wenn man die DEL im SDRAM-Speicher, der allerdings wesentlich längere Zugriffszeiten als der SRAM-Speicher benötigt, unterbringen würde, ist die maximale Anzahl an emulierbaren Neuronen durch die Speicherkapazität der Markierungsspeicher vom NTC-FPGA begrenzt, die 256 KB beträgt. Folglich könnten dann $2^{18} \cdot 8 = 2^{21} \approx 2 \cdot 10^6$ Neuronen berücksichtigt werden. Bei Verwendung von mittlerweile verfügbaren 2 GB großen SDRAM-DIMM-Speichern könnte man zusätzlich die emulierbare Anzahl an adaptiven Synapsen verdoppeln.

Die verteilte Speicherarchitektur von SEE stellt insgesamt 10 eigenständige Datenkanäle zu externen Speichern zur Verfügung. Es werden 4 Datenkanäle mit insgesamt 16 Bytes vom PPC2-FPGA, 3 Datenkanäle mit 12 Bytes vom NTC-FPGA und 3 Datenkanäle mit insgesamt 24 Bytes vom NSC-FPGA angesteuert. Die theoretisch maximal nutzbare Speicherbandbreite, die von der Emulationsplattform SEE bereitgestellt wird, beträgt somit 52 Bytes·50 MHz = 2,6 GB/s. Der Datentransfer von den PEs zum Parameterspeicher, welcher vom NSC-FPGA durchgeführt wird und sich als der größte sequentielle Teil der PCNN Simulation erweist, stellt das größte Flaschenhals-Problem bei der Simulationsbeschleunigung von PCNNs dar. Für die effiziente Parallelisierung dieses Datentransfers

wird fast die Hälfte der Speicherbandbreite von SEE (24 Bytes·50 MHz = 1,2 GB/s) zur Verfügung gestellt.

Die Systemarchitektur von SEE zeichnet sich durch eine geeignete Balance zwischen SIMD- und MIMD-Architektur aus und erzielt somit einen ausreichenden Beschleunigungsfaktor im Vergleich zu PCNN Simulationen, die auf einem Einprozessor-System in Form eines Linux-PCs durchgeführt wurden (2,4 GHz Pentium-IV-Prozessor von Intel mit 1 GB RAM Arbeitsspeicher). Bei einer Taktfrequenz von 50 MHz kann ein Beschleunigungsfaktor von 30 für spärliche Vernetzungsstrukturen, wie Vierer- und Achter-Nachbarschaftsverbindungen, und ein Beschleunigungsfaktor von 8 für dichte Vernetzungsstrukturen hinsichtlich einer Vollvernetzung erzielt werden. Dies entspricht, zumindest bei einer spärlichen Vernetzungsstruktur, der Zielsetzung, dass die Emulationszeit um eine Größenordnung kleiner sein soll als die notwendige Simulationszeit auf einem Einprozessor-System (siehe Abschn. "Funktionale Emulation" auf Seite 4).

Evaluation

Die Evaluation des Neuronenmodells mit adaptiven Synapsengewichten hat gezeigt, dass die Netzwerkdynamik, die die Informationsverarbeitung anhand von Pulsen bestimmt, auf der Adaption der Synapsengewichte W_{KL} basiert. Für Neuronen, die mit ihren 4 nächsten Nachbarneuronen verbunden sind und gemeinsam mit ähnlich starken Eingangsreizen stimuliert werden, stellt sich ein korreliertes Pulsverhalten ein, das sich zur Fleck-Detektion oder Objekt-Hintergrund-Trennung im Rahmen einer Bildvorverarbeitung eignet (siehe Abschn. "Korreliertes Pulsverhalten" auf Seite 143). Die adaptiven Synapsengewichte erreichen dabei nach einer Netzwerkzeit von 500 ms einen “eingeschwungenen” Zustand, wenn die Membranpotentiale a_K der Eingangsschichtneuronen am Anfang der Simulation mit zufälligen Werten zwischen 0 und dem Schwellwert θ initialisiert werden. Wenn benachbarte Neuronen allerdings mit ähnlich schwachen Eingangsreizen stimuliert werden, ist das Pulsverhalten dieser Neuronen zueinander unkorreliert. Die Adaption der Synapsengewichte führt in diesem Fall zu einem periodischen Verhalten, in denen die Synapsen sich abwechselnd in einer Verstärkungsphase und in einer Ruhephase befinden. Das unkorrelierte Pulsverhalten hat einen periodischen Zustand der adaptiven Synapsengewichte zur Folge (siehe Abschn. "Unkorreliertes Pulsverhalten" auf Seite 145). Das Neuronenmodell mit adaptiven Synapsengewichten modelliert lediglich die kurzeitige Wirksamkeit von Synapsen (*Short-Term Memory*), da die Synapsen, selbst nach aufeinanderfolgenden verstärkenden Effekten, sofort abklingen, wenn keine präsynaptischen Pulse mehr vorhanden sind.

Untersuchungen zu den Genauigkeitsanforderungen an die Festkomma-Arithmetik haben gezeigt, dass die gleichen Pulsfrequenzen wie bei der Verwendung einer Fließkomma-Arithmetik mit 16 binären Nachkommastellen erreicht werden. Um allerdings auch eine ausreichende Übereinstimmung bei der Standardabweichung der Pulszeitpunkte im Vergleich zur Fließkomma-Arithmetik zu erzielen, müssen neben den 2 binären Vorkommastellen mindestens 18 binäre Nachkommastellen für die Darstellung von Festkomma-Zahlen berücksichtigt werden. Membranpotentiale und Synapsengewichte wurden aus diesem Grund mit 20 Bit dargestellt. Dies zeigt, dass das verwendete Neuronenmodell im Vergleich zu anderen pulscodierten Neuronenmodellen [Eckh90] eine höhere Auflösung benötigt, und bestätigt die Ausführungen in [Stre01a], dass die in Neurocomputern häufig verwendete Wortbreite von 16 Bit in vielen Fällen unzureichend ist.

Bewertung und Einordnung

Bei der Leistungsbewertung von weiteren 6 Emulationsplattformen, die auf Halb-Kunden-Entwürfen und auf FPGAs basieren, zeigt sich, dass die Effizienz von Emulationsplattformen am höchsten ist, wenn durch parallele Datenkanäle eine Erhöhung der Speicherbandbreite erzielt wird ohne im gleichen Maße die Anzahl der Gatter für die HW-Implementierung zu erhöhen. Die in dieser Arbeit konzipierte und realisierte Emulationsplattform SEE stellt sich dabei als effizienteste Plattform für die Emulation eines Netzwerks von pulscodierten Neuronen mit adaptiven Synapsengewichten heraus, wenn man die verfügbare Speicherbandbreite für die Neuronenzustandsberechnung, das Verhältnis von Synapsen zu Neuronen und die verwendete Gatteranzahl berücksichtigt. Ferner stellt SEE die erste Emulationsplattform dar, die für das in [Heit02] beschriebene pulscodierte Neu-

ronenmodell entwickelt wurde. In dieser Arbeit wurde weiterhin gezeigt, dass für die Anforderungen an eine digitale Emulation von pulscodierten neuronalen Netzen programmierbare applikationsspezifische Schaltkreise in Form von FPGAs am besten geeignet sind. Die Vorteile heutiger FPGA-Technologien bestehen neben der Kostenwirksamkeit für Prototypen-Entwürfe und der Flexibilität aufgrund der Rekonfigurierbarkeit vor allem in der hohen Integrationsdichte, der ausreichend hohen Performanz bezüglich der realisierbaren Operationsfrequenzen und der hinreichend nutzbaren Speicherbandbreite.

7.2 Ausblick

Die realisierte Emulationsplattform SEE könnte zur weiteren Erhöhung der Simulationsbeschleunigung ausgebaut werden.

Mögliche Erweiterungen

Eine mögliche Erweiterung der digitalen Emulationsplattform wäre ein weiteres FPGA für die Neuronenzustandsberechnung vorzusehen, um den Parallelitätsgrad der PCNN Emulation durch die weitere Erhöhung der Speicherbandbreite zu erhöhen. Diese Maßnahme würde die Beschleunigung der numerischen Integration etwa um den Faktor 2 erhöhen. Allerdings hat dies einen signifikanten Kostenaufwand zur Folge, da die Entflechtung und Herstellung der Prototypen-Platine aufwendiger wird. Bei der Entflechtung ist darauf zu achten, dass möglichst kurze Leiterbahnen von dem zusätzlichen FPGA zu den zusätzlich erforderlichen 3 SDRAM-DIMM-Speichern realisiert werden, damit die Operationsfrequenzen ausreichend hoch bleiben. Hierfür sind mehr als 300 weitere Signalleitungen vom FPGA an die 3 SDRAM-DIMM-Speicher zu führen, die auch physikalisch aufgrund der Sockelgröße viel Platz auf der Prototypen-Platine benötigen. Die Prototypen-Platine müsste für die Herstellung größer dimensioniert werden und die zusätzlichen SDRAM-DIMM-Speicher müssten sinnvollerweise im Hinblick auf kurze Leiterbahnen rückseitig angebracht werden. Die Lagenanzahl von 10 wäre, wie auch im Anhang L auf Seite 184 ersichtlich, nicht mehr ausreichend.

Digitale Architekturen

Gegenwärtig gibt es einige Architekturtrends von integrierten Digitalsystemen und neue Technologieentwicklungen, die sich für eine Performanzsteigerung der digitalen Emulation von großen pulscodierten neuronalen Netzen als sehr vorteilhaft erweisen.

VIRAM

Eine vielversprechende Zukunftsrichtung ist beispielsweise die Integration von Hauptspeicher und Prozessor auf demselben Chip [Asan03]. Der VIRAM-Prozessor etwa integriert zu einem 64-Bit-MIPS-Prozessor mit 8 KB Befehls- und Daten-Cache insgesamt 8 DRAM-Blöcke (*Embedded DRAM*). Jeder eingebettete DRAM-Block bietet eine Speicherkapazität von 13 MB und eine 256 Bit-breite Schnittstelle für Eingangs- und Ausgangsdaten. Der Vorteil von eingebetteten On-Chip DRAM-Speichern gegenüber externen Off-Chip DRAM-Speichern ist, dass die Speicherbandbreite nicht durch die konstante Bitbreite des Speicher-Busses limitiert ist. Bei On-Chip DRAM-Speichern ist eine Ansteuerung mit wesentlich größeren Bitbreiten möglich, was die Speicherbandbreite entsprechend erhöht. Der VIRAM-Prozessor wird mit einer Taktfrequenz von 200 MHz betrieben und braucht bei einer 0,18 µm-Technologie für die insgesamt 125 Millionen Transistoren eine Chipfläche von 325 mm^2 [Gebi04].

Cell

Eine alternative Zukunftsentwicklung ist die Integration einer heterogenen MIMD/SIMD-Architektur, die in [Hofs05] als nicht-homogene SMP-Architektur (SMP = *Symmetric Multiprocessing*) bezeichnet wird. Der Cell-Prozessor, der von IBM, Sony und Toshiba entwickelt wurde, stellt eine solche Architektur dar. Neben einem 64-Bit-PPC-Prozessor, der über einen L1 Cache-Speicher à 32 KB und einen 512 KB großen L2 Cache-Speicher verfügt und als PPE (*PowerPC Processing Element*) bezeichnet wird, sind auf dem Chip 8 DSPs integriert, die einen integrierten lokalen Speicher von 256 KB für SIMD-Operationen haben und als SPEs (*Synergistic Processing Elements*) gekennzeichnet werden. Das PPE ist mit den 8 SPEs über einen EIB (*Element Interface Bus*) verbunden, der eine Bandbreite von 96 Bytes hat und den simultanen Transfer von 12 Befehlen ermöglicht. Der Cell-Prozessor beinhaltet eine XDR-Speicheransteuerung (XDR = *External Data Representation Standard*) mit 2 unabhängigen Datenkanälen à 32 Bit. XDR-Speicher können pro Takt 8

Daten liefern, und der externe XDR-Bus wird mit 400 MHz betrieben. Dies führt zu einer enorm hohen Speicherbandbreite von 2·8·4 Bytes·400 MHz = 25,6 GB/s. Der Cell-Prozessor kann mit einer Taktfrequenz von über 4 GHz betrieben werden und benötigt bei einer 90 nm-Technologie für die insgesamt 234 Millionen Transistoren eine Chipfläche von 221 mm^2 [Hofs05].

Virtex-4

Eine neue Technologieentwicklung im Bereich der FPGAs stellt das seit Anfang 2005 erhältliche Virtex-4-FPGA von Xilinx dar [Xili05d]. Die Core-Spannung dieses FPGAs liegt bei 1,2 V, und es wird im Rahmen einer 90 nm-Technologie mit 10 Kupfer-Lagen fabriziert. Die beiden wesentlichen Technologiefortschritte sind zum einen die erzielbaren Operationsfrequenzen und zum anderen die neuartige Anordnung der Pads. Ein Virtex-4-FPGA erreicht für die Ansteuerung von internen Logik-, Multiplizierer- und Speicher-Blöcken eine Taktfrequenz von über 500 MHz (bei Verwendung des leistungsfähigsten Speedgrades von -12). Die Anordnung der Pads ist bei den Virtex-4-FPGAs nicht auf die 4 Chipkanten begrenzt, sondern die Pads sind über die ganze Chipfläche verteilt. Registerausgänge zur Ansteuerung externer Speicher, die sich in der Chipmitte befinden, müssen nicht mehr bis an den Chiprand zu einem Pad geführt werden. Diese Registerausgänge können nun in der Chipmitte mit einem Pad verbunden werden, welches zusätzlich kürzere Wege zu den FPGA-Pins hat. Dies führt zu kleineren Leitungsverzögerungen, zu externen Speichern oder sonstigen externen Schaltungsteilen, und die erzielbaren Operationsfrequenzen im Zusammenhang mit der Inbetriebnahme auf einer Prototypen-Platine können erhöht werden.

BEE2

Der Architekturtrend von leistungsfähigen FPGA basierten Emulationsplattformen, die sich auch für die PCNN Emulation eignen, geht immer mehr zu einem kompakteren Platinen-Entwurf (überschaubare Anzahl von FPGAs) und zu der Bereitstellung hoher Speicherbandbreiten. Es zeigt sich also der Ansatz, die Vorteile von Rapid-Prototyping- und ASIC-Prototyping-Plattformen zu nutzen, wie dies auch in dieser Arbeit mit der konzipierten Emulationsplattform SEE versucht wurde. Ein Beispiel hierfür ist die Weiterentwicklung der FPGA basierten Emulationsplattform BEE, die als BEE2 bezeichnet wird [Chan05]. Anstatt der 20 FPGAs bei der BEE-Plattform werden bei der BEE2-Plattform lediglich 5 Virtex-II-Pro-FPGAs (XCVP70-7FF1704) verwendet, die jeweils mit 4 DDR2-SDRAM-DIMM-Speicher verbunden sind. Diese SDRAM-DIMM-Speicher haben jeweils einen 64 Bit breiten Datenbus und liefern bei einer Taktfrequenz von 200 MHz 2 Daten pro Takt. Somit verfügt jedes der 5 FPGAs über eine beachtliche Speicherbandbreite von 4·2·8 Bytes·200 MHz = 12,8 GB/s. Die entworfene BEE2-Platine verwendet 22 Lagen und kommt mit einer Größe von 36×38 cm aus.

PCNN Systeme der Zukunft

Die PCNN Systeme, die in Zukunft für Bildverarbeitungsanwendungen in der Automobil- oder Sicherheitstechnik eingesetzt werden können, müssen neben hohen Performanz-Anforderungen auch bestimmte Kriterien bezüglich des Kostenaufwands (Anzahl der Bauteile und Herstellung der Platine) entsprechen. Diese Anforderungen können am ehesten von hybriden (Digital/Analog) Systemen erfüllt werden, wie das beispielsweise bei Mobilfunksystemen umgesetzt wird. Bei diesen Systemen erfolgt die Verarbeitung von hochfrequenten Daten durch den Analogteil, während die Weiterverarbeitung von empfangenen Daten bzw. die Aufbereitung von zu sendenden Daten durch den Digitalteil realisiert wird. Nach [Hamm03] werden zukünftige PCNN Systeme für Anwendungen in der Bildverarbeitung die Vorteile der Halbleiter-Prozesstechnologie analoger Schaltungstechnik für die Merkmalsextraktion und die Abwicklung einiger elementarer Klassifizierungen nutzen (*Front-End*). Nach einer Digitalwandlung werden dann die kontextabhängigen Prozesse und endgültigen Klassifizierungen der höheren Gerhirnareale mit digitaler Schaltungstechnik bearbeitet (*Back-End*).

Fazit

Die digitale Emulationsplattform SEE ist ein erster und neuartiger Versuch, eine flexible Experimentierplattform für den ganzheitlichen Bildverarbeitungsprozess mit pulscodierten neuronalen Netzen bereitzustellen. Die innovative Kombination von heterogener SIMD/MIMD-Architektur, effizienter Kommunikation zwischen Netzwerktopologie- und Neuronenzustandsberechnung, Hardware-Beschleunigung eines numerischen Integrationsverfahrens, hoher Speicherbandbreite und ausreichender Speicherkapazität ist sowohl für die

Durchführung von Low-Level Operationen als auch von High-Level Operationen im Rahmen einer Emulation von großen pulscodierten neuronalen Netzen geeignet.

Literaturverzeichnis

Stand der WWW (World Wide Web) Links: März 2008

[Abel91] M. Abeles, **Corticonics: Neural Circuits of the Cerebral Cortex**, *Cambridge University Press*, ISBN 0-5213-7476-6, 1991.

[Alte03] Altera Corp., **Excalibur Device Overview Data Sheet**, www.altera.com/literature/ds/ds_arm.pdf, Mai 2003.

[AMBA99] ARM Ltd., **AMBA 2.0 Specification**, www.arm.com/products/solutions/AMBA_Spec.html, Mai 1999.

[Amda67] G. Amdahl, **Validity of single-processor approach to achieving large-scale computing capability**, *American Federation of Information Processing Societies (AFIPS) Conference Proceedings*, Vol. 30, S. 483-485, 1967.

[Attn76] F. Attneave, **Multistability in Perception**, *Recent Progress in Perception (Herausgeber: R. Held und W. Richards)*, *W. H. Freeman*, 1976.

[Asan03] K. Asanovic, **Programmable Neurocomputing**, The Handbook of Brain Theory and Neural Networks (Herausgeber: M. A. Arbib), *MIT Press*, 2nd Edition, ISBN 0-2620-1197-2, 2003.

[Avne05] **Avnet Inc.**, www.avnet.com, 2005.

[Beic93] J. Beichter, N. Brüls, E. Sicheneder, U. Ramacher und H. Klar, **A VLSI Array Processor For Neural Network Algorithms**, *Proceedings of the IEEE Custom Integrated Circuit Conference (CICC)*, S. 4.6.1-4.6.3, Mai 1993.

[Bied87] I. Biederman, **Recognition-by-components: A theory of human image understanding**, *Psychological Review*, Vol. 94, S. 115-147, 1987.

[Birb02] N. Birbaumer und R. F. Schmidt, **Biologische Psychologie**, *Springer Verlag*, 5. Auflage, ISBN 3-5404-3480-1, September 2002.

[Blue04a] **Blue Gene Project**, *IBM Forschungsprojekt mit Biowissenschaftlern am T. J. Watson Research Center*, www.research.ibm.com/bluegene, November 2004.

[Blue04b] **IBM's Blue Gene/L goes on sale**, *ZDNet News*, news.zdnet.com/2100-9584_22-5442285.html, November 2004.

[Blue05] **Blue Brain Project**, *IBM Forschungsprojekt mit dem Brain Mind Institute (BMI) an der École Polytechnique Fédérale de Lausanne (EPFL)*, bluebrainproject.epfl.ch, Juni 2005.

[Bret94] D. Brettle und E. Niebur, **Detailed Parallel Simulation of a Biological Neuronal Network**, *IEEE Computational Science and Engineering*, Issue 1, S. 31-43, 1994.

[Chan03a] C. Chang, K. Kuusilinna, B. Richards und R. W. Broderson, **Implementation of BEE: a Real-time Large-scale Hardware Emulation Engine**, *International Symposium on Field Programmable Gate Arrays (FPGA)*, Monterey (USA), S. 91-99, Februar 2003.

[Chan03b] C. Chang, K. Kuusilinna, B. Richards, A. Chen, N. Chan, R. W. Broderson und B. Nikolic, **Rapid Design and Analysis of Communication Systems Using the BEE Hardware Emulation Environment**, *IEEE 14th Workshop on Rapid System Prototyping (RSP)*, San Diego (USA), S. 148-154, Juni 2003.

[Chan05] C. Chang, J. Wawrzynek und R. W. Broderson, **BEE2: A High-End Reconfigurable Computing System**, *IEEE Design & Test of Computers*, Vol. 22, No. 2, S. 114-125, bwrc.eecs.berkeley.edu/Research/BEE, März/April 2005.

[Core01] IBM Corp., **CoreConnect Bus Architecture**, www-03.ibm.com/chips/products/coreconnect, Mai 2001.

[Daya01] P. Dayan und L. F. Abbott, **Theoretical Neuroscience - Computational and Mathematical Modeling of Neural Systems**, *MIT Press*, ISBN 0-2620-4199-5, 2001.

[Delo01] A. Delorme und S. Thorpe, **Face identification using one spike per neuron: resistance to image degradations**, *Neural Networks*, Vol. 14, S. 795-803, 2001.

[Delo03] A. Delorme und S. Thorpe, **SpikeNET: An Event-driven Simulation Package for Modelling Large Networks of Spiking Neurons**, *Network: Computation in Neural Systems*, Vol. 14, S. 613-627, www.spikenet-technology.com, 2003.

[Dini05] **The Dini Group Inc.**, www.dinigroup.com, 2005.

[Eckh90] R. Eckhorn, H. J. Reitböck, M. Arndt und P. Dicke, **Feature linking via synchronization among distributed assemblies: Simulations of results from cat visual cortex**, *Neural Computation*, Vol. 2, S. 293-307, 1990.

[Eckh99] R. Eckhorn, **Neural Mechanisms of Scene Segmentation: Recordings from the Visual Cortex Suggest Basic Circuits for Linking Field Models**, *IEEE Transactions on Neural Networks*, Vol. 10, No. 3, S. 464-479, Mai 1999.

[Euro05] Europractice IC Service, **ASIC Prototyping (MPW runs)**, www.europractice-ic.com/prototyping.php, 2005.

[Flyn66] M. J. Flynn, **Very High-Speed Computing Systems**, *Proceedings of the IEEE*, Vol. 54, No. 12, S. 1901-1909, Dezember 1966.

[Fran99] G. Frank, G. Hartmann, A. Jahnke und M. Schäfer, **An Accelerator for Neural Networks with Pulse-Coded Model Neurons**, *IEEE Transactions on Neural Networks*, Vol. 10, No. 3, S. 527-538, Mai 1999.

[Fren70] A. S. French und R. B. Stein, **A flexible neural analog using integrated circuits**, *IEEE Transactions on Biomedical Engineering*, Vol. 17, No. 3, S. 248-253, 1970.

[Funk02] K. Funke, Z. F. Kisvárday, M. Volgushev und F. Wörgötter, **Integrating Anatomy and Physiology of the Primary Visual Pathway: From LGN to Cortex**, *Models of Neural Networks IV - Early Vision and Attention (Herausgeber: J. L. van Hemmen, J. D. Cowan und E. Domany)*, *Springer Verlag*, ISBN 0-3879-5105-9, 2002.

[Gebi04] J. Gebis, S. William, C. Kozyrakis und D. Patterson, **VIRAM1: A Media-Oriented Vector Processor with Embedded DRAM**, *41st Design Automation Student Design Contest*, San Diego (USA), iram.cs.berkeley.edu, Juni 2004.

[Geik07] M. Geike, **Hardware/Software Co-Entwurf einer konfigurierbaren Topologie-Einheit für eine digitale Emulationsplattform pulscodierter neuronaler Netze**, *Diplomarbeit im Fachgebiet Mikroelektronik des Instituts für Technische Informatik und Mikroelektronik (TIME) an der Fakultät IV (Elektrotechnik und Informatik) der TU Berlin*, 2007.

[GENE05] **The GENESIS Simulator**, www.genesis-sim.org/GENESIS, 2005.

[Gers98] W. Gerstner, **Spiking Neurons**, Pulsed Neural Networks (Herausgeber: W. Maass und C. M. Bishop), *MIT Press*, ISBN 0-2621-3350-4, S. 1-53 , 1998.

[Gers02] W. Gerstner und W. M. Kistler, **Spiking Neuron Models - Single Neurons, Populations, Plasticity**, *Cambridge University Press*, ISBN 0-5218-1384-0, Oktober 2002.

[Gilo93] W. K. Giloi, **Rechnerarchitektur**, *Springer Verlag*, 2. Auflage, ISBN 3-5405-6355-5, 1993.

[Gold01] D. H. Goldberg, G. Cauwenberghs und A. G. Andreou, **Analog VLSI Spiking Neural Network With Address Domain Probabilistic Synapses**, *IEEE International Symposium on Circuits and Systems (ISCAS)*, Vol. 2, S. 241-244, Mai 2001.

[Gold02] E. B. Goldstein, **Wahrnehmungspsychologie**, *Originaltitel: **Sensation and Perception**, Wadsworth Group, 6. Auflage (aus dem Amerikanischen übersetzt von G. Herbst und M. Ritter)*, *Spektrum Akademischer Verlag*, 2. deutsche Auflage, ISBN 3-8274-1083-5, November 2002.

[Gras98] C. Grassmann und J. K. Anlauf, **Distributed, Event Driven Simulation of Spiking Neural Networks**, *Proceedings of the International ICSC/IFAC Symposium on Neural Computation*, S. 100-105, 1998.

[Graz95] M. S. A. Graziano und C. G. Gross, **The representation of extrapersonal space: A possible role for bimodal, visual-tactile neurons**, The cognitive neurosciences (Herausgeber: M. S. Gazzaniga), *MIT Press*, ISBN 0-2620-7157-6, S. 1021-1034, 1995.

[Grie07] P. Griep, **Implementierung von ARM- und PowerPC-basierten Schnittstellenmodulen für die Emulation pulscodierter neuronaler Netze**, *Diplomarbeit im Fachgebiet Mikroelektronik des Instituts für Technische Informatik und Mikroelektronik (TIME) an der Fakultät IV (Elektrotechnik und Informatik) der TU Berlin*, 2007.

[Hamm95] D. Hammerstrom, **A Digital VLSI Architecture for Real-World Applications**, An Introduction to Neural and Electronic Networks (Herausgeber: S. F. Zornetzer, J. L. Davis, C. Lau und T. McKenna), *Academic Press*, ISBN 0-1278-1882-0, 1995.

[Hamm03] D. Hammerstrom, **Digital VLSI for Neural Networks**, The Handbook of Brain Theory and Neural Networks (Herausgeber: M. A. Arbib), *MIT Press*, 2nd Edition, ISBN 0-2620-1197-2, 2003.

[Hayk99] S. Haykin, **Neural Networks - A Comprehensive Foundation**, *Prentice Hall*, 2nd Edition, ISBN 0-1390-8385-5, 1999.

[Hebb49] D. O. Hebb, **The Organization of Behavior**, *Wiley*, 1949.

[Heem95] J. N. H. Heemskerk, **Overview of Neural Network Hardware**, Neurocomputers for Brain-Style Processing. Design, Implementation and Application, *Dissertation am Departement Psychologie der Universiteit Leiden*, Niederlande, 1995.

[Heit02] A. Heittmann, U. Ramacher, D. Matolin, J. Schreiter und R. Schüffny, **An Analog VLSI Pulsed Neural Network for Image Segmentation using Adaptive Connection Weights**, *International Conference on Artificial Neural Networks (ICANN)*, Madrid (Spanien), S. 1293-1298, August 2002.

[Heit03] A. Heittmann und U. Ramacher, **Correlation-Based Feature Detection Using Pulsed Neural Networks**, *IEEE 13th Workshop on Neural Networks for Signal Processing (NNSP)*, S. 479-488, September 2003.

[Heit04] A. Heittmann und U. Ramacher, **An Architecture for Feature Detection Utilizing Dynamic Synapses**, *Proceedings of the IEEE 47th Midwest Symposium on Circuits and Systems (MWSCAS)*, Hiroshima (Japan), Juli 2004.

[Hell04a] H. H. Hellmich und H. Klar, **SEE: a Concept for an FPGA based Emulation Engine for Spiking Neurons with Adaptive Weights**, *Proceedings of the 5th World Scientific and Engineering Academy and Society (WSEAS) International Conference on Neural Networks and Applications (NNA)*, Udine (Italien), März 2004.

[Hell04b] H. H. Hellmich und H. Klar, **An FPGA based Simulation Acceleration Platform for Spiking Neural Networks**, *Proceedings of the IEEE 47th Midwest Symposium on Circuits and Systems (MWSCAS)*, Hiroshima (Japan), Juli 2004.

[Hell04c] H. H. Hellmich und H. Klar, **Digitalsimulator für pulscodierte neuronale Netze**, *Advances in Radio Science - Kleinheubacher Berichte*, Miltenberg, September 2004.

[Hell05a] H. H. Hellmich, M. Geike, P. Griep, P. Mahr, M. Rafanelli und H. Klar, **Spiking Neural Network Emulation Engine**, *Datenblatt der digitalen Emulationsplattform im BMBF-Förderprojekt "Vision IC"*, Mai 2005.

[Hell05b] H. H. Hellmich, P. Mahr und M. Geike, **Graphical User Interface of the Spiking Neural Network Emulation Engine**, Juni 2005.

[Hell05c] H. H. Hellmich, M. Geike, P. Griep, P. Mahr, M. Rafanelli und H. Klar, **Emulation Engine for Spiking Neuons and Adaptive Synapses**, *IEEE International Joint Conference on Neural Networks (IJCNN)*, Montreal (Kanada), Juli/August 2005.

[Henn03] J. L. Hennessy und D. A. Patterson, **Computer Architecture - A Quantitative Approach**, *Morgan Kaufmann Publishers*, 3rd Edition, ISBN 1-5586-0724-2, 2003.

[Hine03] M. Hines und N. T. Carnevale, **Computer Modeling Methods for Neurons**, The Handbook of Brain Theory and Neural Networks (Herausgeber: M. A. Arbib), *MIT Press*, 2nd Edition, ISBN 0-2620-1197-2, 2003.

[Hodg52] A. L. Hodgkin und A. F. Huxley, **A quantitative description of ion currents and its applications to conduction and excitation in nerve membranes**, *Journal of Physiology*, Vol. 117, S. 500-544, 1952.

[Hofs05] H. P. Hofstee, **Power Efficient Processor Architecture and The Cell Processor**, *11th International Symposium on High-Performance Computer Architecture (HPCA)*, S. 258-262, www.research.ibm.com/cell, Februar 2005.

[Hube62] D. H. Hubel und T. N. Wiesel, **Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex**, *Journal of Physiology*, Vol. 160, S. 106-154, 1962.

[Hube68] D. H. Hubel und T. N. Wiesel, **Receptive fields and functional architecture of monkey striate cortex**, *Journal of Physiology*, Vol. 195, S. 215-243, 1968.

[Ienn96] P. Ienne, **Programmable VLSI Systolic Processors for Neural Network and Matrix Computations**, *Dissertation an der École Polytechnique Fédérale de Lausanne (EPFL)*, Thése No. 1525, Schweiz, 1996.

[Itti98] L. Itti, C. Koch und E. Niebur, **A Model of Saliency-Based Visual Attention for Rapid Scene Analysis**, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, No. 11, S. 1254-1259, Nov. 1998.

[Jahn96] A. Jahnke, U. Roth und H. Klar, **A SIMD/Dataflow Architecture for a Neurocomputer for Spike-Processing Neural Networks (NESPINN)**, *Proceedings of the 5th International Conference on Microelectronics for Neural Networks and Fuzzy Systems (MicroNeuro)*, Lausanne (Schweiz), S. 232-237, Februar 1996.

[Jahn98] A. Jahnke, U. Roth und T. Schönauer, **Digital Simulation of Spiking Neural Networks**, Pulsed Neural Networks (Herausgeber: W. Maass und C. M. Bishop), *MIT Press*, ISBN 0-2621-3350-4, 1998.

[Jesc05] H. Jeschke, **Kosten- und Performance-Modellierung von applikationsspezifischen VLSI-Architekturen**, *Dissertation an der Fakultät für Elektrotechnik und Informatik der Universität Hannover*, www.ims.uni-hannover.de/de/forschung/promotionen/?year=2005, Juni 2005.

[John94] J. L. Johnson, **Pulse-coupled neural nets: translation, rotation, scale, distortion, and intensity signal invariance for images**, *Applied Optics*, Vol. 33, No. 26, S. 6239-6253, September 1994.

[John99] J. L. Johnson und M. L. Padgett, **PCNN models and applications**, *IEEE Transactions on Neural Networks*, Vol. 10, No. 3, S. 480-498, Mai 1999.

[Kand00] E. R. Kandel, J. H. Schwartz und T. M. Jessell, **Principles of Neural Science**, *McGraw Hill*, 4th Edition, ISBN 0-0711-2000-9, Juli 2000.

[Keat02] M. Keating und P. Bricaud, **Reuse Methodology Manual for System-on-a-Chip Designs**, *Kluwer Academic Publishers*, 3rd Edition, ISBN 1-4020-7141-8, 2002.

[Klar04] H. Klar, **Integrierte Digitale Schaltungen MOS/BICMOS**, *Springer Verlag*, 2. Auflage, ISBN 3-5404-0600-X, Dezember 2004.

[Kuhn01] B. Kuhn, **Messungen von Aktionspotentialen in Dendriten von kultivierten und gentechnische veränderten Hippocampusneuronen mit spannungssensitiven Farbstoffen**, *Dissertation an der Fakultät für Physik der Technischen Universität München*, PHY 825d, Mai 2001.

[Krup00] H. Krupnova und G. Saucier, **FPGA-Based Emulation: Industrial and Custom Prototyping Solutions**, *10th International Conference on Field Programmable Logic and Applications (FPL)*, Villach (Österreich), S. 68-77, August 2000.

[Lazz93] J. Lazzaro, J. Wawrzynek, M. Mohawald, M. Sivilotti und D. Gillespie, **Silicon Auditory Processors as Computer Peripherals**, *IEEE Transactions on Neural Networks*, Vol. 4, No. 3, S. 523-528, Mai 1993.

[Lind98a] C. S. Lindsey und T. Lindblad, **Review of Hardware Neural Networks: A User's Perspective**, *Springer-Verlag*, www.particle.kth.se/~lindsey/elba2html/elba2html.html, 1998.

[Lind98b] T. Lindblad und J. M. Kinser, **Image Processing using Pulse-Coupled Neural Networks**, *Springer-Verlag*, ISBN 3-5407-6264-7, 1998.

[MacG87] R. J. MacGregor, **Neural and Brain Modeling**, *Academic Press*, ISBN 0-1246-4261-6, 1987.

[Mahr07] P. Mahr, **Optimierung von externen Kommunikationskanälen eines Digitalsimulators für pulscodierte neuronale Netze**, *Diplomarbeit im Fachgebiet Mikroelektronik des Instituts für Technische Informatik und Mikroelektronik (TIME) an der Fakultät IV (Elektrotechnik und Informatik) der TU Berlin*, 2007.

[Mala00] R. Malaka und S. Buck, **Solving Nonlinear Optimization Problems Using Networks of Spiking Neurons**, *IEEE International Joint Conference on Neural Networks (IJCNN)*, Vol. 6, S. 486-491, Juli 2000.

[Mals81] C. von der Malsburg, **The Correlation Theory of Brain Function**, *Max-Planck-Institute (MPI) of Biophysical Chemistry*, Internal Report 81-2, Göttingen, 1981.

[Mano97] M. M. Mano und C. R. Kime, **Logic and Computer Design Fundamentals**, *Prentice Hall*, ISBN 0-1320-6780-3, 1997.

[Marc93] J. J. Marchesi, **Handbuch der Fotografie**, *Verlag Photographie*, ISBN 3-7231-0024-41, 1993.

[Masc98] M. V. Mascagni und A. S. Sherman, **Numerical Methods for Neuronal Modelling**, Methods of Neuronal Modelling - From Ions to Networks (Herausgeber: C. Koch und I. Segev), *MIT Press*, 2nd Edition, ISBN 0-2621-1231-0, 1998.

[MATL05] **MATLAB Neural Network Toolbox**, www.mathworks.com/products/neuralnet, 2005.

[Mari02] I. Marian, R. Reilly und D. Mackey, **Efficient Event-Driven Simulation of Spiking Neural Networks**, *Proceedings of the 3rd WSEAS International Conference on Neural Networks and Applications (NNA)*, Interlaken (Schweiz), Februar 2002.

[Marr82] D. Marr, **Vision - A Computational Investigation into Human Representation and Processing of Visual Information**, *W. H. Freeman*, ISBN 0-7167-1567-8, 1982.

[Matt00] M. Mattia und P. Del Giudice, **Efficient Event-Driven Simulation of Large Networks of Spiking Neurons and Dynamical Synapses**, *Neural Computation*, Vol. 12, S. 2305-2329, 2000.

[Maxf04] C. M. Maxfield, **The Design Warrior's Guide to FPGAs: Devices, Tools and Flows**, *Elsevier*, ISBN 0-7506-7604-3, 2004.

[Mean94] R. W. Means, **High Speed Parallel Hardware Performance Issues for Neural Network Applications**, *IEEE International Conference on Computational Intelligence*, Vol. 1, Orlando (USA), S. 10-16, 1994.

[Mehr01] N. Mehrtash, D. Jung und H. Klar, **Neuro-Accelerator für Synapsen-plastische Spikende Neuronale Netze (SP²INN)**, *ITG-Tagung der Fachgruppe 8.4.9 - Mikroelektronik neuronaler Netzwerke (Fachbereich 8: Mikro- und Nanoelektronik, Fachausschuss 8.4: Integrierte Systeme)*, 2001.

[Mehr03] N. Mehrtash, D. Jung, H. H. Hellmich, T. Schönauer, V. T. Lu und H. Klar, **Synaptic Plasticity in Spiking Neural Networks (SP²INN): A System Approach**, *IEEE Transactions on Neural Networks*, Vol. 14, No. 5, S. 980-992, September 2003.

[Merc02] D. Mercier und R. Séguier, **Spiking neurons (STANNs) in speech recognition**, *Proceedings of the 3rd WSEAS International Conference on Neural Networks and Applications (NNA)*, Interlaken (Schweiz), Februar 2002.

[Mohr97] K. Mohraz, U. Schott und M. Pauly, **Parallel Simulation of Pulse Coded Neural Networks**, *Proceedings of the 15th World Congress on Scientific Computation, Modelling and Applied Mathematics (IMACS)*, Vol. 6, Berlin, S. 523-528, 1997.

[Morg03] N. Morgan, **Programmable Neurocomputing Systems**, The Handbook of Brain Theory and Neural Networks (Herausgeber: M. A. Arbib), *MIT Press*, 2nd Edition, ISBN 0-2620-1197-2, 2003.

[MPI05] **The Message Passing Interface (MPI) Standard**, www-unix.mcs.anl.gov/mpi, 2005.

[Müll95] U. A. Müller, A. Gunzinger und W. Guggenbühl, **Fast Neural Net Simulation with a DSP Processor Array**, *IEEE Transactions on Neural Networks*, Vol. 6, No. 1, S. 203-213, Januar 1995.

[Murr03] J. M. J. Murre, **Neurosimulators**, The Handbook of Brain Theory and Neural Networks (Herausgeber: M. A. Arbib), *MIT Press*, 2nd Edition, ISBN 0-2620-1197-2, 2003.

[Nall05] **Nallatech Inc.**, www.nallatech.com, 2005.

[NEUR05] **NEURON Simulator: for empirically-based simulations of neurons and networks of neurons**, www.neuron.yale.edu/neuron, 2005.

[Nume05] **Numerical Recipes**, *Books On-Line*, Chapter 16, www.nr.com/nronline_switcher.html, 2005.

[Oliv71] J. Oliver, **The efficiency of extrapolation methods for numerical integration**, *Journal Numerische Mathematik*, Vol. 17, No. 1, S. 17-32, Februar 1971.

[Pent05] **Pentek Inc.**, www.pentek.com, 2005.

[Porr02] M. Porrmann, U. Witkowski, H. Kalte und U. Rückert, **Implementation of Artificial Neural Networks on a Reconfigurable Hardware Accelerator**, *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (PDP)*, Cran Canaria (Spanien), S. 243-250, Januar 2002.

[Pran93] S. J. Prange und H. Klar, **Cascadable Digital Emulator IC for 16 Biological Neurons**, *Proceedings of the 40th IEEE International Solid-State Circuits Conference (ISSCC)*, S. 234-235+294, Februar 1993.

[Prei01] R. Preis, K. Salzwedel, C. Wolff und G. Hartmann, **Efficient Parallel Simulation of Pulse-Coded Neural Networks (PCNN)**, *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2001.

[ProD05] **ProDesign GmbH**, www.prodesign-europe.com, 2005.

[PVM05] **PVM: Parallel Virtual Machine**, www.csm.ornl.gov/pvm, 2005.

[Rafa06] M. Rafanelli, **A PowerPC based digital emulation environment for spiking neural networks**, *Diplomarbeit im Fachgebiet Mikroelektronik des Instituts für Technische Informatik und Mikroelektronik (TIME) an der Fakultät IV (Elektrotechnik und Informatik) der TU Berlin*, 2006.

[Rama95] U. Ramacher, W. Raab, J. Anlauf, U. Hachmann, J. Beichter, N. Brüls, M. Wesseling, E. Sicheneder, J. Gläss, A. Wurz und R. Männer, **SYNAPSE-1: A High-Speed General Purpose Parallel Neurocomputer System**, *Proceedings of the 9th IEEE International Parallel Processing Symposium (IPPS)*, S. 774-781, April 1995.

[Reed03] Reed Business Information, **PLD market will recover as functionality increases**, *Purchasing Magazine*, www.purchasing.com/article/CA329630.html, Oktober 2003.

[Reut02] J. Reutimann, M. Giugliano und S. Fusi, **Event-Driven Simulation of Spiking Neurons embedded in Very Large Networks**, *Proceedings of the World Congress on Neuroinformatics*, S. 468-477, Wien (Österreich), Mai 2002.

[Reyn03] L. M. Reyneri, **Implementation Issues of Neuro-Fuzzy Hardware: Going Toward HW/SW Codesign**, *IEEE Transactions on Neural Networks*, Vol. 14, No. 1, S. 176-194, Januar 2003.

[Rinz85] J. Rinzel, **Excitation dynamics: insights from simplified membrane models**, *Federation Proceedings*, Vol. 44, No. 15, S. 2944-2946, Dezember 1985.

[Rogg03] D. Roggen, S. Hofmann, Y. Thoma und D. Floreano, **Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot**, *The 2003 NASA/DoD Conference on Evolable Hardware (EH)*, S. 199-208, Juli 2003.

[Roth95] U. Roth, A. Jahnke und H. Klar, **Hardware Requirements for Spike-Processing Neural Networks**, *World Congress on Neural Networks (WCNN)*, S. 460-463, Washington (USA), Juli 1995.

[Roth97] U. Roth, F. Eckhardt, A. Jahnke und H. Klar, **Efficient On-Line Computation of Connectivity: Architecture of the Connection Unit of NESPINN**, *Proceedings of the 6th International Conference on Microelectronics for Neural Networks, Evolutionary and Fuzzy Systems (MicroNeuro)*, Dresden, S. 31-38, September 1997.

[Sann03] C. Sannelli, **Analyses of pulse-coded neural networks with adaptive weights**, *Diplomarbeit im Fachgebiet Mikroelektronik des Instituts für Technische Informatik und Mikroelektronik (TIME) an der Fakultät IV (Elektrotechnik und Informatik) der TU Berlin*, SDI-S-114, November 2003.

[Schä99] M. Schäfer und G. Hartmann, **A Flexible Hardware Architecture for Online Hebbian Learning in the Sender-Oriented PCNN-Neurocomputer Spike 128K**, *Proceedings of the 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (MicroNeuro)*, S. 316-323, Granada (Spanien), 1999.

[Schä02] M. Schäfer, T. Schönauer, C. Wolff, G. Hartmann, H. Klar und U. Rückert, **Simulation of Spiking Neural Networks - Architectures and Implementations**, *Neurocomputing*, Elsevier, Vol. 48, S. 647-679, 2002.

[Schö98a] T. Schönauer, A. Jahnke U. Roth und H. Klar, **Digital Neurohardware: Principles and Perspective**, *3. Internationaler Workshop für Neuronale Netze in der Anwendung (NN)*, Magdeburg, Februar 1998.

[Schö98b] T. Schönauer, N. Mehrtash, A. Jahnke und H. Klar, **MASPINN: Novel Concepts for a Neuro-Accelerator for Spiking Neural Networks**, *Workshop on Virtual Intelligence and Dynamic Neural Networks (VIDYNN)*, Stockholm (Schweden), Juni 1998.

[Schö00] T. Schönauer, S. Atasoy, N. Mehrtash und H. Klar, **Simulation of a Digital Neuro-Chip for Spiking Neural Networks**, *IEEE International Joint Conference on Neural Networks (IJCNN)*, Como (Italien), Juli 2000.

[Schö02a] T. Schönauer, S. Atasoy, N. Mehrtash und H. Klar, **NeuroPipe-Chip: A Digital Neuro-Processor for Spiking Neural Networks**, *IEEE Transactions on Neural Networks*, Vol. 13, No. 1, S. 205-213, Januar 2002.

[Schö02b] T. Schönauer, **Ein digitaler Neuroprozessor für biologienahe neuronale Netze**, *Dissertation an der Fakultät IV (Elektrotechnik und Informatik) der TU Berlin*, ISBN 3-8322-0462-8, Juni 2002.

[Schr02] J. Schreiter, U. Ramacher, A. Heittmann, D. Matolin und R. Schüffny, **Analog Implementation for Networks of Integrate-And-Fire Neurons with Adaptive Local Connectivity**, *IEEE 12th Workshop on Neural Networks Signal Processing (NNSP)*, S. 657-666, September 2002.

[Seif02] U. Seifert, **Artificial Neural Networks on Massively Parallel Computer Hardware**, *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*, Brüssel (Belgien), S. 319-330, April 2002.

[Sing95] W. Singer, **Development and Plasticity of Cortical Processing Architectures**, *Science*, Vol. 270, S. 758-764, November 1995.

[Smit97] M. J. S. Smith, **Application-Specific Integrated Circuits**, *Addison Wesley*, ISBN 0-2015-0022-1, 1997.

[SNNS05] **Stuttgart Neural Network Simulator**, www.ra.cs.uni-tuebingen.de/SNNS, 2005.

[Spik05] **SpikeNNS: a Simulator for Spiking Neural Networks**, cortex.cs.may.ie/tools/spikeNNS, 2005.

[Stei67] R. B. Stein, **The frequency of nerve action potentials generated by applied currents**, *Proceedings of the Royal Society*, Vol. B167, S. 227-236, 1967.

[Stöc96] M. Stöcker, H. J. Reitböck und R. Eckhorn, **A neural network for scene segmentation by temporal coding**, *Neurocomputing*, Elsevier, Vol. 11, S. 123-134, 1996.

[Stöc97] M. Stöcker, R. Eckhorn und H. J. Reitböck, **Size and position invariant visual representation supports retinotopic maps via selective backwards paths: A dynamic second**

order neural network model for a possible functional role of recurrent connections in the visual cortex, *Neurocomputing*, Elsevier, Vol. 17, S. 111-132, 1997.

[Stoe00] J. Stoer und R. Bulirsch, **Numerische Mathematik II**, *Springer Verlag*, 4. Auflage, ISBN 3-5406-7644-9, 2000.

[Stre01a] A. Strey, **Spezifikation und parallele Simulation neuronaler Netze**, *Habilitation an der Fakultät für Informatik der Universität Ulm*, ISBN 3-1836-6110-1, Februar 2001.

[Stre01b] A. Strey und M. Bange, **Performance Analysis of Intel's MMX and SSE: A Case Study**, *Proceedings of the European Conference on Parallel Computing (EuroPar)*, Manchester (Großbritannien), August 2001.

[Sund05] **Sundance Multiprocessor Technology Ltd.**, www.sundance.com, 2005.

[Tay92] O. N. Tay und P. D. Noakes, **The Use of Cache Memory in Neurocomputer Design**, *IEEE International Joint Conference on Neural Networks (IJCNN)*, Vol. 2, S. 571-576, Juni 1992.

[Thor96] S. J. Thorpe, D. Fitze und C. Marlot, **Speed of processing in the human visual system**, *Nature*, Vol. 381, S. 520-522, 1996.

[TOP05] **TOP500 Supercomputer Sites**, www.top500.org, 2005.

[Trei80] A. Treisman und G. Gelade, **A feature integration theory of attention**, *Cognitive Psychology*, Vol. 12, S. 97-136, 1980.

[Veen00] H. J. M. Veendrick, **Deep-Submicron CMOS ICs - From Basics to ASICs**, *Kluwer Academic Publishers*, 2nd Edition, ISBN 9-0440-0111-6, 2000.

[Visi01] **VisionIC - Intelligente Vision-Plattformen für den Massenmarkt**, *Vorhabenbeschreibung zum BMBF-Förderprogramm "Innovationen für die Wissenschaftsgesellschaft" mit dem Forschungsschwerpunkt "Entwurf innovativer integrierter Chipsysteme"*, November 2001.

[Watt94] L. Watts, **Event-Driven Simulation of Networks of Spiking Neurons**, *Advances in Neural Information Processing Systems*, Vol. 6, S. 927-934, 1994.

[Wawr96] L. Watts, K. Asanovic, B. Kingsbury, J. Beck, D. Johnson und N. Morgan, **Spert-II: A vector microprocessor system**, *IEEE Computer*, Vol. 29, No. 3, S. 79-86, 1996.

[Weit97] L. Weitzel, K. Kopecz, C. Spengler, R. Eckhorn und H. J. Reitböck, **Contour Segmentation with Recurrent Neural Networks of Pulse-Coding Neurons**, *International Conference on Computer Analysis of Images and Patterns (CAIP)*, S. 337-344, 1997.

[Wolf99] C. Wolff, G. Hartmann und U. Rückert, **ParSPIKE - A Parallel DSP-Accelerator for Dynamic Simulation of Large Spiking Neural Networks**, *Proceedings of the 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (MicroNeuro)*, S. 324-331, Granada (Spanien), 1999.

[Wolf01] C. Wolff, **Parallele Simulation großer pulscodierter neuronaler Netze**, *Dissertation am Fachbereich Elektrotechnik an der Universität-GH Paderborn*, D14-162, Mai 2001.

[XESS05] **XESS Corp.**, www.xess.com, 2005.

[Xili97] Xilinx Inc., **Gate Count Capacity Metrics for FPGAs**, direct.xilinx.com/bvdocs/appnotes/xapp059.pdf, Februar 1997.

[Xili98] Xilinx Inc., **XC3000 Series Field Programmable Gate Arrays**, direct.xilinx.com/bvdocs/publications/3000.pdf, November 1998.

[Xili99] Xilinx Inc., **XC4000E and XC4000X Series Field Programmable Gate Arrays**, direct.xilinx.com/bvdocs/publications/4000.pdf, Mai 1999.

[Xili01] Xilinx Inc., **Virtex 2.5 V Series Field Programmable Gate Arrays**, direct.xilinx.com/bvdocs/publications/ds003.pdf, April 2001.

[Xili02] Xilinx Inc., **Virtex-E 1.8 V Series Field Programmable Gate Arrays**, direct.xilinx.com/bvdocs/publications/ds022.pdf, Juli 2002.

[Xili05a] **Xilinx Inc.**, www.xilinx.com, 2005.

[Xili05b] Xilinx Inc., **Virtex-II Pro Complete Data Sheet**, direct.xilinx.com/bvdocs/publications/ds083.pdf, Web Power Tool: www.xilinx.com/cgi-bin/power_tool/power_Virtex2p, März 2005.

[Xili05c] Xilinx Inc., **Virtex-II Complete Data Sheet**, direct.xilinx.com/bvdocs/publications/ds031.pdf, Web Power Tool: www.xilinx.com/cgi-bin/power_tool/power_Virtex2, März 2005.

[Xili05d] Xilinx Inc., **Special Edition Virtex-4: Achieving Breakthrough Performance at the Lowest Cost**, www.xilinx.com/publications/xcellonline/xcell_52/xc_toc52.htm, *Xcell Journal*, Issue 52, 2005.

[Zell00] A. Zell, **Simulation neuronaler Netze**, *Oldenbourg Verlag*, 3. Auflage, ISBN 3-4862-4350-0, 2000.

Anhang

Anhang B : Lösung Differentialgleichung I

Die Lösung der Differentialgleichung (DGL) erfolgt über die Methode der “Variation der Konstanten”. Die ermittelte Konstante in der homogenen DGL (HDGL) wird bei der Berechnung der inhomogenen DGL (IDGL) als zeitlich veränderlich angenommen. Die Lösung der DGL setzt sich aus der Summe der Lösungen der HDGL und IDGL zusammen sowie der Berücksichtigung der Anfangsbedingungen (AB).

DGL:

$$\frac{du(t)}{dt} + \frac{u(t)}{\tau} = \frac{R \cdot I_0}{\tau}$$

HDGL:

$$\frac{du}{dt} = -\frac{u}{\tau}$$

$$\Leftrightarrow \int \frac{du}{u} = -\int \frac{1}{\tau} dt$$

$$\Leftrightarrow \frac{1}{k} \ln|u| = -\frac{1}{\tau} t \Rightarrow u_H = k \cdot e^{-\frac{t}{\tau}}$$

IDGL:

$$u = k(t) \cdot e^{-\frac{t}{\tau}}$$

$$u' = k(t)' \cdot e^{-\frac{t}{\tau}} - k(t) \cdot \frac{1}{\tau} \cdot e^{-\frac{t}{\tau}}$$

Einsetzen der DGL:

$$k(t)' \cdot e^{-\frac{t}{\tau}} - k(t) \cdot \frac{1}{\tau} \cdot e^{-\frac{t}{\tau}} + \frac{1}{\tau} \cdot k(t) \cdot e^{-\frac{t}{\tau}} = \frac{R \cdot I_0}{\tau}$$

$$k(t)' = \frac{R \cdot I_0}{\tau} \cdot e^{\frac{t}{\tau}}$$

$$\Rightarrow k(t) = R \cdot I_0 \cdot e^{\frac{t}{\tau}} \Rightarrow u_I = R \cdot I_0$$

Lösung der DGL:

$$u = u_H + u_I = k \cdot e^{-\frac{t}{\tau}} + R \cdot I_0$$

Berücksichtigung der AB:

$$u(t_0) = 0$$

$$\Rightarrow u(t_0) = k \cdot e^{-\frac{t_0}{\tau}} + R \cdot I_0$$

$$\Leftrightarrow k = -R \cdot I_0 \cdot e^{\frac{t_0}{\tau}}$$

$$\Rightarrow u(t) = R \cdot I_0 \cdot \left(1 - e^{-\frac{(t - t_0)}{\tau}}\right)$$

Anhang C : Lösung Differentialgleichung II

Die Lösung der Differentialgleichung (DGL) zweiter Ordnung erfolgt über die Berechnung der Eigenwerte (Nullstellen) für die homogene DGL (HDGL). Je nachdem, ob eine doppelte Nullstelle, 2 reelle oder komplexe Nullstellen vorliegen, ergeben sich unterschiedliche Lösungen, bestehend aus Exponential-, Sinus- und Cosinus-Funktionen. Die Lösung der DGL setzt sich aus der Summe der Lösungen der HDGL und IDGL zusammen sowie der Berücksichtigung der Anfangsbedingungen (AB).

DGL:

$$a_K(t)' = W_{KL}(t) \wedge W_{KL}(t)' = -\gamma \cdot W_{KL}(t) - \mu \cdot \left(a_K(t) + \frac{\theta}{2}\right) \wedge \gamma, \mu, \theta > 0$$

$$\Rightarrow a_K(t)'' + \gamma \cdot a_K(t)' + \mu \cdot a_K(t) = \mu \cdot \frac{\theta}{2}$$

HDGL:

$$\lambda^2 + \gamma \cdot \lambda + \mu = 0 \Leftrightarrow \lambda_{1,2} = -\frac{\gamma}{2} \pm \sqrt{\frac{\gamma^2}{4} - \mu} = -\frac{\gamma}{2} \pm i\sqrt{\mu - \frac{\gamma^2}{4}}$$

$$\Rightarrow a_{KH} = c_1 \cdot e^{-\frac{\gamma}{2}t} \cdot \sin(\omega_0 \cdot t) + c_2 \cdot e^{-\frac{\gamma}{2}t} \cdot \cos(\omega_0 \cdot t) \quad | \ \omega_0 = \sqrt{\mu - \frac{\gamma^2}{4}}$$

IDGL (für konstante Lösung gilt):

$$a_K' = a_K'' = 0$$

Einsetzen der DGL:

$$\mu \cdot a_K = \mu \cdot \frac{\theta}{2} \Rightarrow a_{KI} = \frac{\theta}{2}$$

Lösung der DGL:

$$a_K = a_{KH} + a_{KI} = c_1 \cdot e^{-\frac{\gamma}{2}t} \cdot \sin(\omega_0 \cdot t) + c_2 \cdot e^{-\frac{\gamma}{2}t} \cdot \cos(\omega_0 \cdot t) + \frac{\theta}{2}$$

$$a_K' = c_1 \cdot e^{-\frac{\gamma}{2}t} \cdot \left[-\frac{\gamma}{2} \cdot \sin(\omega_0 \cdot t) + \omega_0 \cdot \cos(\omega_0 \cdot t)\right] + c_2 \cdot e^{-\frac{\gamma}{2}t} \cdot \left[-\frac{\gamma}{2} \cdot \cos(\omega_0 \cdot t) - \omega_0 \cdot \sin(\omega_0 \cdot t)\right]$$

Berücksichtigung der AB:

$$a_K(t=0) = 0 \Rightarrow c_2 + \frac{\theta}{2} = 0 \Leftrightarrow c_2 = -\frac{\theta}{2}$$

$$a_K(t=0)' = 0 \Rightarrow c_1 \cdot \omega_0 - c_2 \cdot \frac{\gamma}{2} = 0 \Rightarrow c_1 = -\frac{\theta}{2} \cdot \frac{\gamma}{2 \cdot \omega_0}$$

$$a_K(t) = \frac{\theta}{2} \cdot \left(1 - e^{-\frac{\gamma}{2}t} \cdot \left(\frac{\gamma}{2 \cdot \omega_0} \cdot \sin(\omega_0 \cdot t) + \cos(\omega_0 \cdot t)\right)\right)$$

Anhang D : Lösung Differentialgleichung III

Die Lösung der Differentialgleichung (DGL) erfolgte mit dem Programm Mathematica von Wolfram Research. Es sind die Eingaben in Mathematica und die dazugehörige Lösung (durch Tastenkombination Shift+Return) dargestellt.

Eingabe:

$$\text{Solve}\left[u == (\theta - \theta_0) * e^{\frac{-t}{\tau_\theta}} + \theta_0,\ t\right]$$

Lösung:

$$\left\{\left\{t \to \text{Log}\left[\frac{\theta - \theta_0}{u - \theta_0}\right] \tau_\theta\right\}\right\}$$

Eingabe:

$$M = 2;$$

$$\text{Solve}\left[(u - u_0) * e^{\frac{-t}{M * \tau_\theta}} + u_0 == (\theta - \theta_0) * e^{\frac{-t}{\tau_\theta}} + \theta_0,\ t\right]$$

Lösung:

$$\left\{\left\{t \to \text{Log}\left[\left(u^2 - 2\,u\,u_0 + 2\,\theta\,u_0 + u_0^2 - 2\,\theta\,\theta_0 - 2\,u_0\,\theta_0 + 2\,\theta_0^2 - (u - u_0)\sqrt{u^2 - 2\,u\,u_0 + 4\,\theta\,u_0 + u_0^2 - 4\,\theta\,\theta_0 - 4\,u_0\,\theta_0 + 4\,\theta_0^2}\right) \Big/ \left(2\left(u_0^2 - 2\,u_0\,\theta_0 + \theta_0^2\right)\right)\right] \tau_\theta\right\},$$

$$\left\{t \to \text{Log}\left[\left(u^2 - 2\,u\,u_0 + 2\,\theta\,u_0 + u_0^2 - 2\,\theta\,\theta_0 - 2\,u_0\,\theta_0 + 2\,\theta_0^2 + (u - u_0)\sqrt{u^2 - 2\,u\,u_0 + 4\,\theta\,u_0 + u_0^2 - 4\,\theta\,\theta_0 - 4\,u_0\,\theta_0 + 4\,\theta_0^2}\right) \Big/ \left(2\left(u_0^2 - 2\,u_0\,\theta_0 + \theta_0^2\right)\right)\right] \tau_\theta\right\}\right\}$$

Anhang E : Berechnungsformeln der Neuronenadressen

Jedes Neuron wird aufgrund eines effizienten Zugriffs auf die Markierungsspeicher ETF und FTF in einer codierten Form angegeben. Anhand dieser Neuronencodierung lässt sich für jedes Neuron eine eindeutige Neuronenadresse mit Bitposition in den Markierungsspeichern ermitteln. Die Neuronencodierung bei einer Kaskaden-Architektur besteht im wesentlichen aus Netzwerkschichtnummer, x- und y-Koordinate (siehe auch Abb. 4-20 auf Seite 86), und die Neuronencodierung bei einer Gabor-Architektur setzt sich aus Gabor-Detektornummer, Nummer des rezeptiven Feldes und des Gradientenpaares zusammen (siehe auch Abb. 4-23 auf Seite 88).

Kaskaden-Architektur

Bei der Kaskaden-Architektur sind für die Berechnung der Neuronenadresse, neben der x- und y-Koordinate des Neurons, noch folgende Parameter erforderlich:

- L = Netzwerkschicht des Neurons (L = 0...31)
- $x_{MAX,L}$, $y_{MAX,L}$ = maximale x- und y- Koordinaten der Netzwerkschicht L ($x_{MAX,L}$ = $y_{MAX,L}$ = 1...4095)
- $BADDR_L$ = Neuronenadressen-Offset der Netzwerkschicht L

Neuronenadresse[1,2]:

$$\begin{aligned} POS_{NEURON}(x) &= (x-1)\%32 \\ ADDR_{NEURON}(x,y,L) &= (y-1)\cdot ceil\left(\frac{x_{MAX,L}}{32}\right) + ceil\left(\frac{x}{32}\right) - 1 + BADDR_L \end{aligned} \quad \textbf{(A-1)}$$

Offset der Neuronenadressen:

$$\begin{aligned} BADDR_0 &= 0 \\ BADDR_L &= ADDR_{NEURON}(x_{MAX,L-1}, y_{MAX,L-1}, L-1) + 1 \\ &= ceil\left(\frac{x_{MAX,L-1}}{32}\right)\cdot y_{MAX,L-1} + BADDR_{L-1} \end{aligned} \quad \textbf{(A-2)}$$

Gabor-Architektur

Bei der Gabor-Architektur sind für die Berechnung der Neuronenadressen folgende Parameter erforderlich:

- D = Gabor-Detektor des Neurons (D = 0...7)
- x_{MAX}, y_{MAX} = maximale x- und y- Koordinaten der Eingangsschicht ($x_{MAX} = y_{MAX}$ = 1...4095)
- RF = rezeptives Feld eines positiven (RF = 0) oder negativen Gabor-Detektorprofils (RF = 1)
- NUM_{RF} = Nummer des rezeptiven Feldes eines Gabor-Detektors (NUM_{RF} = 0...32767) [x_{RF}, y_{RF} = x- und y-Werte zur Koordinatendarstellung rezeptiver Felder ($x_{RF} \cdot y_{RF}$ = 1...32768)]
- NUM_{GDP} = Nummer des Gradientenpaares innerhalb eines rezeptiven Feldes eines Gabor-Detektors (NUM_{GDP} = 0...511) [x_{GDP}, y_{GDP} = x- und y-Werte zur Koordinatendarstellung von Gradientenpaaren ($x_{GDP} \cdot y_{GDP}$ = 1...512)]
- $BADDR_{D,i}$ = Neuronenadressen-Offset der unterschiedlichen Netzwerkschichten eines Gabor-Detektors (i = {COR,DEC,SUM,SCOR,SDEC})

Neuronenadresse[3]:

$$\begin{aligned} POS_{NEURON,INP}(x) &= (x-1)\%32 \\ ADDR_{NEURON,INP}(x,y) &= (y-1)\cdot ceil\left(\frac{x_{MAX}}{32}\right) + ceil\left(\frac{x}{32}\right) - 1 \end{aligned} \quad \textbf{(A-3)}$$

$$\left.\begin{aligned} POS_{NEURON,i}(NUM_{GDP}) &= NUM_{GDP}\%32 \\ ADDR_{NEURON,i}(RF, NUM_{RF}, NUM_{GDP}) &= BADDR_{D,i} + (RF \cdot x_{RF,D} \cdot y_{RF,D} + NUM_{RF,D}) \\ &\quad \cdot ceil\left(\frac{x_{GDP,D}\cdot y_{GDP,D}}{32}\right) + floor\left(\frac{NUM_{GDP,D}}{32}\right) \end{aligned}\right| \; i = \{COR, DEC\} \quad \textbf{(A-4)}$$

$$\begin{aligned} POS_{NEURON,SUM}(RF, NUM_{RF}) &= 4\cdot(NUM_{RF}\%8) + RF \\ POS_{NEURON,SCOR}(NUM_{RF}) &= 4\cdot(NUM_{RF}\%8) + 2 \\ POS_{NEURON,SDEC}(NUM_{RF}) &= 4\cdot(NUM_{RF}\%8) + 3 \\ ADDR_{NEURON,i}(NUM_{RF}) &= BADDR_{D,i} + floor\left(\frac{NUM_{RF,D}}{8}\right) \;|\; i = \{SUM, SCOR, SDEC\} \end{aligned} \quad \textbf{(A-5)}$$

Offset der Neuronenadressen:

$$\begin{aligned} BADDR &= ceil\left(\frac{x_{MAX}}{32}\right)\cdot y_{MAX} \\ BADDR_D &= BADDR + \sum_{i=0}^{D-1}\left[4\cdot x_{RF,i}\cdot y_{RF,i}\cdot ceil\left(\frac{x_{GDP,i}\cdot y_{GDP,i}}{32}\right) + ceil\left(\frac{x_{RF,i}\cdot y_{RF,i}}{8}\right)\right] \end{aligned} \quad \textbf{(A-6)}$$

1. % entspricht dem Modulo-Operator, z. B. 14 % 5 = 4
2. ceil() stellt die ganzzahlige Aufrundung dar, z. B. ceil(2,5) = 3 [engl. ceil = Decke]
3. floor() stellt die ganzzahlige Abrundung dar, z. B. ceil(2,5) = 2 [engl. floor = Boden]

$$
\begin{aligned}
BADDR_{D,COR} &= BADDR_D \\
BADDR_{D,DEC} &= BADDR_D + 2 \cdot x_{RF,D} \cdot y_{RF,D} \cdot ceil\left(\frac{x_{GDP,D} \cdot y_{GDP,D}}{32}\right) \\
BADDR_{D,SUM} &= BADDR_D + 4 \cdot x_{RF,D} \cdot y_{RF,D} \cdot ceil\left(\frac{x_{GDP,D} \cdot y_{GDP,D}}{32}\right) \\
BADDR_{D,SCOR} &= BADDR_{D,SUM} \\
BADDR_{D,SDEC} &= BADDR_{D,SUM}
\end{aligned}
\qquad \textbf{(A-7)}
$$

Anhang F : Berechnungsformeln der Neuronennummern

Anhand der Neuronencodierung lässt sich für jedes Neuron ebenfalls eine eindeutige Neuronennummer ermitteln. Die Neuronennummern werden für den effizienten Zugriff auf den Parameterspeicher eines Neurons (NIB) genutzt und aus Zuordnungszwecken für die Visualisierung von Pulsereignissen nach der Durchführung einer Emulation benötigt (siehe auch Kapitel 5.5.4 auf Seite 125).

Kaskaden-Architektur

Bei der Kaskaden-Architektur sind für die Berechnung der Neuronennummer, neben der x- und y-Koordinate des Neurons, noch folgende Parameter erforderlich:

- L = Netzwerkschicht des Neurons (L = 0...31)
- $x_{MAX,L}$, $y_{MAX,L}$ = maximale x- und y- Koordinaten der Netzwerkschicht L ($x_{MAX,L}$ = $y_{MAX,L}$ = 1...4095)
- $BNUM_L$ = Neuronennummer-Offset der Netzwerkschicht L ($BNUM_L = 1^2 ... 4095^2 \cdot L$)

Neuronennummer:

$$
NUM_{NEURON}(x, y, L) = BNUM_L + x_{MAX,L} \cdot (y_{MAX,L} - y) + (x - 1) \qquad \textbf{(A-8)}
$$

Offset der Neuronennummern:

$$
\begin{aligned}
BNUM_0 &= 0 \\
BNUM_L &= BNUM_{L-1} + x_{MAX,L-1} \cdot y_{MAX,L-1} \quad | \; L \geq 1
\end{aligned}
\qquad \textbf{(A-9)}
$$

Gabor-Architektur

Bei der Gabor-Architektur sind für die Berechnung der Neuronennummer folgende Parameter erforderlich:

- D = Gabor-Detektor des Neurons (D = 0...7)
- x_{MAX}, y_{MAX} = maximale x- und y- Koordinaten der Eingangsschicht (x_{MAX} = y_{MAX} = 1...4095)
- RF = rezeptives Feld eines positiven (RF = 0) oder negativen Gabor-Detektorprofils (RF = 1)
- NUM_{RF} = Nummer des rezeptiven Feldes eines Gabor-Detektors (NUM_{RF} = 0...32767) [x_{RF}, y_{RF} = x- und y-Werte zur Koordinatendarstellung rezeptiver Felder ($x_{RF} \cdot y_{RF}$ = 1...32768)]
- NUM_{GDP} = Nummer des Gradientenpaares innerhalb eines rezeptiven Feldes eines Gabor-Detektors (NUM_{GDP} = 0...511) [x_{GDP}, y_{GDP} = x- und y-Werte zur Koordinatendarstellung von Gradientenpaaren ($x_{GDP} \cdot y_{GDP}$ = 1...512)]
- $BNUM_{D,i}$ = Neuronennummer-Offset der unterschiedlichen Netzwerkschichten eines Gabor-Detektors (i = {COR,DEC,SUM,SCOR,SDEC})

Neuronennummer:

$$
NUM_{NEURON,INP}(x, y) = x_{MAX} \cdot (y_{MAX} - y) + (x - 1) \qquad \textbf{(A-10)}
$$

$$
\begin{aligned}
NUM_{NEURON,i}(RF, NUM_{RF}, NUM_{GDP}) = {} & BNUM_{D,i} + RF \cdot (x_{RF,D} \cdot y_{RF,D} \cdot x_{GDP,D} \cdot y_{GDP,D}) \\
& + NUM_{RF,D} \cdot x_{GDP,D} \cdot y_{GDP,D} + NUM_{GDP,D}
\end{aligned}
\quad \Big| \; i = \{COR, DEC\} \qquad \textbf{(A-11)}
$$

$$
NUM_{NEURON,SUM}(RF, NUM_{RF}) = BNUM_{D,SUM} + RF \cdot (x_{RF,D} \cdot y_{RF,D}) + NUM_{RF,D} \qquad \textbf{(A-12)}
$$

$$
NUM_{NEURON,i}(NUM_{RF}) = BNUM_{D,i} + NUM_{RF,D} \quad | \; i = \{SCOR, SDEC\} \qquad \textbf{(A-13)}
$$

Offset der Neuronennummern:

$$\begin{aligned} BNUM &= x_{MAX} \cdot y_{MAX} \\ BNUM_D &= BNUM + \sum_{i=0}^{D-1} [4 \cdot x_{RF,i} \cdot y_{RF,i} \cdot (x_{GDP,i} \cdot y_{GDP,i} + 1)] \end{aligned} \quad \textbf{(A-14)}$$

$$\begin{aligned} BNUM_{D,COR} &= BNUM_D \\ BNUM_{D,DEC} &= BNUM_D + 2 \cdot (x_{RF,D} \cdot y_{RF,D} \cdot x_{GDP,D} \cdot y_{GDP,D}) \\ BNUM_{D,SUM} &= BNUM_D + 4 \cdot (x_{RF,D} \cdot y_{RF,D} \cdot x_{GDP,D} \cdot y_{GDP,D}) \\ BNUM_{D,SCOR} &= BNUM_D + x_{RF,D} \cdot y_{RF,D} \cdot (4 \cdot x_{GDP,D} \cdot y_{GDP,D} + 2) \\ BNUM_{D,SDEC} &= BNUM_D + x_{RF,D} \cdot y_{RF,D} \cdot (4 \cdot x_{GDP,D} \cdot y_{GDP,D} + 3) \end{aligned} \quad \textbf{(A-15)}$$

Anhang G : Skript für Synthese der Hardware

```
# $Source: /home/spinnsoc/REPOSITORY/hrdwr/see/syn/tcl/top_nsc_pre.tcl,v $
#-------------------------------------------------------------------------------
set FPGA "nsc"
# set FPGA "ntc"
# set FPGA "ppc2"
set TOP "top_nsc"
# set TOP "top_ntc"
# set TOP "top_ppc2"
set LIB VIRTEX-II
# set LIB {VIRTEX-II Pro}
set DEVICE 2V4000ff1152
# set DEVICE 2VP20ff1152
set PROC 4
# set PROC 5
set WIRE xcv2-4000-4_avg
# set WIRE xcv2p-20-5_avg
set FREQ 50
set CLK 20
set IDLY 5
set ODLY 5
set IODLY 10
set XDB_DIR ../$TOP
set NET_DIR ../../net
set RPT_DIR ../../rpt
set SRC_DIR ../../src
set WRK_DIR ./$TOP
set SYNNET "v"
#-------------------------------------------------------------------------------
exec mkdir -p $WRK_DIR
exec rm -rf $WRK_DIR/$TOP\.psp
exec rm -rf $WRK_DIR/$TOP\_impl_1
new_project -name $TOP -folder $WRK_DIR -createimpl
#-------------------------------------------------------------------------------
puts "TOP DESIGN: $TOP"
setup_design -manufacturer Xilinx -family $LIB -part $DEVICE -speed $PROC -frequency
$FREQ
source $TOP\_edn.tcl
source $TOP\_vhd.tcl
source $TOP\_v.tcl
compile
current_design $TOP
#-------------------------------------------------------------------------------
```

```
# TIMING CONSTRAINTS
create_clock -name clk -period $CLK
set_input_delay $IDLY -clock clk [all_inputs]
set_output_delay $ODLY -clock clk [all_outputs]
set_max_delay $IODLY -from [all_inputs] -to [all_outputs]
set_wire_load_model -name $WIRE
#-------------------------------------------------------------------------------
# SYNTHESIZE
synthesize
#-------------------------------------------------------------------------------
# REPORTS
report_area $RPT_DIR/syn_area_pre_$TOP.rpt -cell_usage -all_leafs
report_timing $RPT_DIR/syn_delay_pre_$TOP.rpt -summary -num_paths 10 -critical_paths -
clock_frequency
#-------------------------------------------------------------------------------
# NETLISTS
if {$SYNNET == "v"} { auto_write -format Verilog $NET_DIR/$TOP\_synnet.v
} else { auto_write -format VHDL $NET_DIR/$TOP\_synnet.vhd }
# auto_write -format SDF $NET_DIR/$TOP\_synnet.sdf
auto_write -format EDIF $NET_DIR/$TOP.edf
exec mv $NET_DIR/$TOP.xdb $XDB_DIR
exec rm -f $NET_DIR/$TOP\_synnet.xdb
if {[file exists ../precision.pl]} { puts ">>> Found Perl-Script \"precision.pl $FPGA\"
<<<"; exec perl ../precision.pl $FPGA}
if {[file exists ../see.pl]} { puts ">>> Found Perl-Script \"see.pl\" <<<"; exec perl
../see.pl $FPGA }
save_impl
close_project
```

Anhang H : Skript für Platzierung und Verdrahtung der Hardware

```
eval 'exec perl -w -S $0 ${1+"$@"}'
  if 0;
#-------------------------------------------------------------------------------
# $Source: /home/spinnsoc/REPOSITORY/hrdwr/see/par/pl/top_ppc2.pl,v $
#-------------------------------------------------------------------------------
$TOP = "top_ppc2";
# $TOP = "top_ntc";
# $TOP = "top_nsc";
chomp ($PWD = `pwd`);
$WORK = "$PWD/par";
$DEVICE = 'xc2vp20-ff1152-5';
# $DEVICE = 'xc2v4000-ff1152-4';
$RPT_DIR = '../../rpt';
$NET_DIR = '../../net';
$UCF_DIR = '../ucf';
$UCF  = $TOP; $UCF =~ s/top_//g;
$NGD  = $ARGV[0];
if (@ARGV eq "1") {
  # look for EDIF netlist
  if (!-f "$PWD/net/$TOP.edf") {
    print "Sorry, but cannot find EDIF netlist net/$TOP.edf!\n";
    exit;
  }
  # look for UCF file
```

```
  if (-f "$WORK/ucf/$UCF.ucf") {
    print("cat par/ucf/$UCF.ucf > par/ucf/$TOP.ucf\n");
    system("cat $WORK/ucf/$UCF.ucf > $WORK/ucf/$TOP.ucf");
  }
  if ((-f "sim/mem/memory_init0.ucf") && (-f "sim/mem/memory_init1.ucf") && (-f "par/
ucf/memory.pl") && ($TOP eq "top_ppc2")) {
    do "par/ucf/memory.pl";
    print("cat sim/mem/memory_init0.ucf >> par/ucf/$TOP.ucf\n");
    system("cat sim/mem/memory_init0.ucf >> $WORK/ucf/$TOP.ucf");
    print("cat sim/mem/memory_init1.ucf >> par/ucf/$TOP.ucf\n");
    system("cat sim/mem/memory_init1.ucf >> $WORK/ucf/$TOP.ucf");
  }
  # create and change to WORK directory
  if (!-d "$WORK/$TOP") {
    print "mkdir par/$TOP;\n";
    system("mkdir $WORK/$TOP");
  }
  print "cd par/$TOP;\n";
  chdir("$WORK/$TOP");
  #-------------------------------------------------------------------------
  # start PAR process
  if (-f "$UCF_DIR/$TOP.ucf") {
    print ("UCF file $TOP.ucf found in par/ucf!");
    system("ngdbuild -aul -uc $UCF_DIR/$TOP.ucf -quiet -p $DEVICE $NET_DIR/$TOP.edf
$TOP.ngd");
  } else {
    system("ngdbuild -quiet -i -p $DEVICE $NET_DIR/$TOP.edf $TOP.ngd");
  }
  system("map -quiet -p $DEVICE -cm area -pr b -k 4 -c 100 -tx off -o $TOP\_map.ncd
$TOP.ngd $TOP.pcf");
  system("par -w -ol 2 -t 1 $TOP\_map.ncd $TOP.ncd $TOP.pcf");
  system("trce -a -v 3 -l 3 -xml $TOP $TOP.ncd -o $TOP.twr $TOP.pcf");
  system("cp $TOP.par $RPT_DIR/par_area_$TOP.rpt");
  system("cp $TOP.twr $RPT_DIR/par_delay_$TOP.rpt");
} else {
  print "cd par/$TOP;\n";
  chdir("$WORK/$TOP");
}
#-------------------------------------------------------------------------
# create PAR netlist
if ($NGD eq "v") {
  print "cp $TOP\_map.ngm $TOP.ngm;\n";
  system("cp $TOP\_map.ngm $TOP.ngm");
  system("netgen -ofmt verilog -w -ne -sim -ngm $TOP.ngm -pcf $TOP.pcf $TOP -dir
$NET_DIR $TOP\_parnet.v");
  $netlist = "$NET_DIR/$TOP\_parnet.v";
} elsif ($NGD eq "vhd") {
  print "cp $TOP\_map.ngm $TOP.ngm;\n";
  system("cp $TOP\_map.ngm $TOP.ngm");
  system("netgen -ofmt vhdl -w -ne -sim -ngm $TOP.ngm -pcf $TOP.pcf $TOP -dir $NET_DIR
$TOP\_parnet.v");
  $netlist = "$NET_DIR/$TOP\_parnet.vhd";
}
#-------------------------------------------------------------------------
# $sdf_annotate
local (*SAVESTDOUT,*SAVESTDERR);
open(SAVESTDOUT, ">&STDOUT") or die "couldn't dup STDOUT: $!";
open(SAVESTDERR, ">&STDERR") or die "couldn't dup STDERR: $!";
if (-f $netlist) {
  open (FILE, "<$netlist") or die "Can't open $netlist: $!\n";
```

```
    @lines = <FILE>;
  close FILE;
  # open same file for writing, reusing STDOUT
  open (STDOUT, ">$netlist") or die "Can't open $netlist: $!\n";
  foreach $line (@lines) {
    $line =~ s/initial \$*.*/initial \$sdf_annotate(\"..\/..\/net\/
$TOP\_parnet.sdf\")\;/o;
    print $line;
  }
  close STDOUT;
}
open(STDOUT, ">&SAVESTDOUT") || die "couldn't dup SAVESTDOUT: $!";
open(STDERR, ">&SAVESTDERR") || die "couldn't dup SAVESTDERR: $!";
```

Anhang I : Befehle und Skript für die FPGA Konfiguration

```
bitgen -g startupclk:Cclk -g userid:0x02140215 -w top_ppc2.ncd
bitgen -g startupclk:Cclk -g userid:0x02140215 -w top_ntc.ncd
bitgen -g startupclk:Cclk -g userid:0x02140215 -w top_nsc.ncd

promgen -o see -w -u 0 top_ppc2.bit top_ntc.bit top_nsc.bit

impact -batch see.lpt

# $Source: /home/spinnsoc/REPOSITORY/hrdwr/see/par/see.lpt,v $
#-------------------------------------------------------------------------------
setMode -ss
setCable -port lpt1
# setCable -port usb1 -baud auto
addDevice -p 1 -file see.mcs
program -showprogress -p 1
quit
```

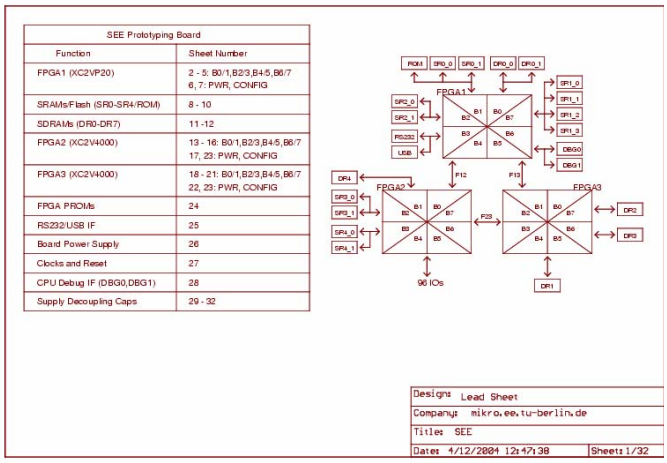
Anhang J : Makefile zum Compilieren/Linken der Software

```
# $Source: /home/spinnsoc/REPOSITORY/sftwr/see/Makefile,v $
#-------------------------------------------------------------------------------
# Define general settings
#-------------------------------------------------------------------------------
ROOT            = $(XILINX_GNUTOOLS)
ROOTBIN         = $(ROOT)/bin
PREFIX          = powerpc-eabi
ROOTLIB         = $(ROOT)/$(PREFIX)/lib
RM              = rm -f
CP              = cp
FIND            = find
MKDIR           = mkdir -p
MAKE            = make
OBJDUMP         = $(ROOTBIN)/$(PREFIX)-objdump
SOFTDIR         = $(SW_DIR)
BSPDIR          = $(SOFTDIR)/libs/bsp
BSPINCLUDEDIR   = $(BSPDIR)/include
TBDIR           = $(HW_PATH)/sim/tb
DEBUG           = $(PROJECT)/bin/debug
```

```
PPC0DIR         = $(SW_PATH)/ppc0
PPC1DIR         = $(SW_PATH)/ppc1
INCDIR          = $(SW_DIR)/see/params
INCDIR0         = $(PPC0DIR)/inc
INCDIR1         = $(PPC1DIR)/inc
#-------------------------------------------------------------------------
# Define location of board support packages (BSP)
#-------------------------------------------------------------------------
BSP_DEBUG_V2PDK= $(BSPDIR)/v2pdk/debug/bsp.a
BSP_RELEASE_V2PDK= $(BSPDIR)/v2pdk/release/bsp.a
#-------------------------------------------------------------------------
# Define build directory (object files and final ELF file)
#-------------------------------------------------------------------------
BUILDDIR_V2PDK= v2pdk
#-------------------------------------------------------------------------
# Define compiler settings
#-------------------------------------------------------------------------
CC                    = $(ROOTBIN)/$(PREFIX)-gcc
CFLAGS_DEBUG_V2PDK0  = -Wall -gdwarf -msdata=eabi -mcpu=405 \
                      -I$(BSPINCLUDEDIR) -I$(INCDIR) -I$(INCDIR0) -DV2PDK -DDEBUG
CFLAGS_DEBUG_V2PDK1  = -Wall -gdwarf -msdata=eabi -mcpu=405 \
                      -I$(BSPINCLUDEDIR) -I$(INCDIR) -I$(INCDIR1) -DV2PDK -DDEBUG
CFLAGS_RELEASE_V2PDK0  = -Wall -O3 -msdata=eabi -mcpu=405 \
                        -I$(BSPINCLUDEDIR) -I$(INCDIR) -I$(INCDIR0) -DV2PDK
CFLAGS_RELEASE_V2PDK1  = -Wall -O3 -msdata=eabi -mcpu=405 \
                        -I$(BSPINCLUDEDIR) -I$(INCDIR) -I$(INCDIR1) -DV2PDK
#-------------------------------------------------------------------------
# Define linker settings
#-------------------------------------------------------------------------
LDSFILE_DEBUG_V2PDK0   = $(SOFTDIR)/mapfiles/mapfile_ppc0
LDSFILE_RELEASE_V2PDK0 = $(SOFTDIR)/mapfiles/mapfile_ppc0
LDSFILE_DEBUG_V2PDK1    = $(SOFTDIR)/mapfiles/mapfile_ppc1
LDSFILE_RELEASE_V2PDK1  = $(SOFTDIR)/mapfiles/mapfile_ppc1
LDFLAGS                  = -T $(LDSFILE) -u _boot
#-------------------------------------------------------------------------
# Define archiver settings
#-------------------------------------------------------------------------
AR          = $(ROOTBIN)/$(PREFIX)-ar
LD          = $(ROOTBIN)/$(PREFIX)-ld
#-------------------------------------------------------------------------
# Suffixes
#-------------------------------------------------------------------------
.SUFFIXES:      .c .o .S
#-------------------------------------------------------------------------
# Compiler rules for C programs
#-------------------------------------------------------------------------
$(BUILDDIR0)/%.o: $(PPC0DIR)/%.c Makefile
      $(CC) $(CFLAGS) -o $@ -c $<
$(BUILDDIR1)/%.o: $(PPC1DIR)/%.c Makefile
      $(CC) $(CFLAGS) -o $@ -c $<
#-------------------------------------------------------------------------
# Object files
#-------------------------------------------------------------------------
OBJECTS0 =  $(BUILDDIR0)/ppc0_main.o \
            $(BUILDDIR0)/ppc0_menu.o \
            $(BUILDDIR0)/ppc0_write.o \
            $(BUILDDIR0)/ppc0_read.o \
            $(BUILDDIR0)/ppc0_see.o \
            $(BUILDDIR0)/ppc0_memory.o \
            $(BUILDDIR0)/ppc0_func.o
```

```
OBJECTS1 =  $(BUILDDIR1)/ppc1_main.o \
            $(BUILDDIR1)/ppc1_menu.o \
            $(BUILDDIR1)/ppc1_write.o \
            $(BUILDDIR1)/ppc1_read.o \
            $(BUILDDIR1)/ppc1_see.o \
            $(BUILDDIR1)/ppc1_memory.o \
            $(BUILDDIR1)/ppc1_func.o
#-------------------------------------------------------------------------------
# Target rules
#-------------------------------------------------------------------------------
all:                    ppc_debug_v2pdk ppc_release_v2pdk
ppc_debug_v2pdk:        ppc_debug_v2pdk_0 ppc_debug_v2pdk_1
ppc_release_v2pdk:      ppc_release_v2pdk_0 ppc_release_v2pdk_1
ppc_debug_v2pdk_0:      $(BUILDDIR_V2PDK)/debug
            $(MAKE) $(BUILDDIR_V2PDK)/debug/ppc0.elf \
            BUILDDIR0=$(BUILDDIR_V2PDK)/debug \
            BSP=$(BSP_DEBUG_V2PDK) \
            CFLAGS="$(CFLAGS_DEBUG_V2PDK0)" \
            LDSFILE=$(LDSFILE_DEBUG_V2PDK0)
ppc_release_v2pdk_0:    $(BUILDDIR_V2PDK)/release
            $(MAKE) $(BUILDDIR_V2PDK)/release/ppc0.elf \
            BUILDDIR0=$(BUILDDIR_V2PDK)/release \
            BSP=$(BSP_RELEASE_V2PDK) \
            CFLAGS="$(CFLAGS_RELEASE_V2PDK0)" \
            LDSFILE=$(LDSFILE_RELEASE_V2PDK0)
ppc_debug_v2pdk_1:      $(BUILDDIR_V2PDK)/debug
            $(MAKE) $(BUILDDIR_V2PDK)/debug/ppc1.elf \
            BUILDDIR1=$(BUILDDIR_V2PDK)/debug \
            BSP=$(BSP_DEBUG_V2PDK) \
            CFLAGS="$(CFLAGS_DEBUG_V2PDK1)" \
            LDSFILE=$(LDSFILE_DEBUG_V2PDK1)
ppc_release_v2pdk_1:    $(BUILDDIR_V2PDK)/release
            $(MAKE) $(BUILDDIR_V2PDK)/release/ppc1.elf \
            BUILDDIR1=$(BUILDDIR_V2PDK)/release \
            BSP=$(BSP_RELEASE_V2PDK) \
            CFLAGS="$(CFLAGS_RELEASE_V2PDK1)" \
            LDSFILE=$(LDSFILE_RELEASE_V2PDK1)
$(BUILDDIR0)/ppc0.elf: $(BSP) $(OBJECTS0) $(LDSFILE)
            @echo "Linking application $@"
            $(LD) $(LDFLAGS) -o $@ $(OBJECTS0) $(BSP)
            $(OBJDUMP) --source $@ > $(BUILDDIR0)/ppc0.lst
$(BUILDDIR1)/ppc1.elf: $(BSP) $(OBJECTS1) $(LDSFILE)
            @echo "Linking application $@"
            $(LD) $(LDFLAGS) -o $@ $(OBJECTS1) $(BSP)
            $(OBJDUMP) --source $@ > $(BUILDDIR1)/ppc1.lst
$(BSP):
            $(MAKE) -C $(BSPDIR)
%/debug:
            -$(MKDIR) $@
%/release:
            -$(MKDIR) $@
BUILD-V2PDK:
            $(MAKE) megaclean all
clean:
      -$(RM) -r $(BUILDDIR_ML2) $(BUILDDIR_V2PDK)
      -$(FIND) . -name '*'_dbg.'*' -print -exec $(RM) {} \;
megaclean: clean
      -$(RM) *~
      -$(RM) .cvsignore
```

Anhang K : Schaltplan der Prototypen-Platine

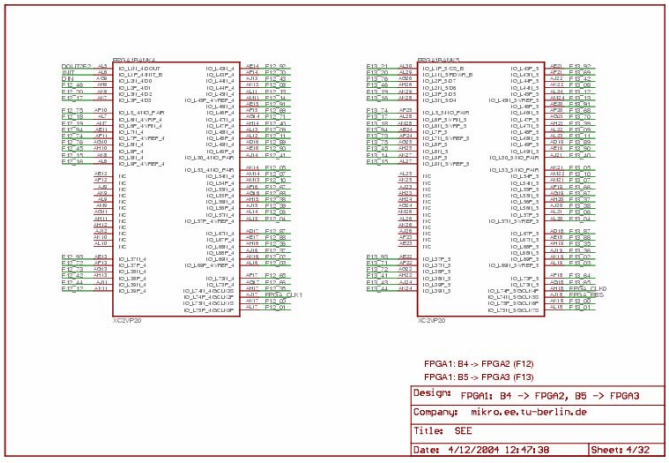


SEE Prototyping Board	
Function	Sheet Number
FPGA1 (XC2VP20)	2 - 5: B0/1,B2/3,B4/5,B6/7 6, 7: PWR, CONFIG
SRAMs/Flash (SR0-SR4/ROM)	8 - 10
SDRAMs (DR0-DR7)	11 -12
FPGA2 (XC2V4000)	13 - 16: B0/1,B2/3,B4/5,B6/7 17, 23: PWR, CONFIG
FPGA3 (XC2V4000)	18 - 21: B0/1,B2/3,B4/5,B6/7 22, 23: PWR, CONFIG
FPGA PROMs	24
RS232/USB IF	25
Board Power Supply	26
Clocks and Reset	27
CPU Debug IF (DBG0,DBG1)	28
Supply Decoupling Caps	29 - 32

Sheet 1/32

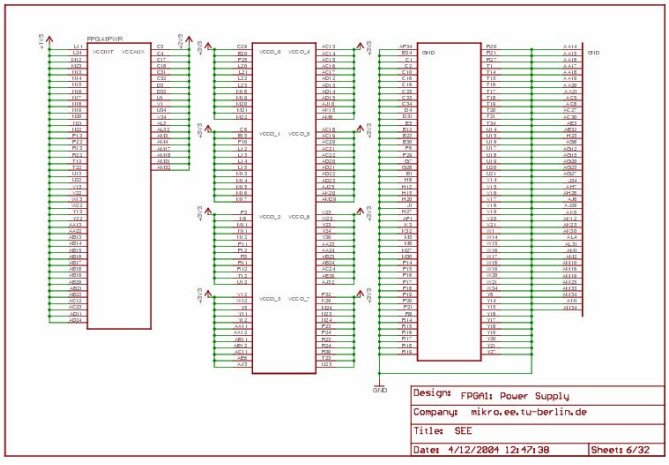
Sheet 2/32

Sheet 3/32

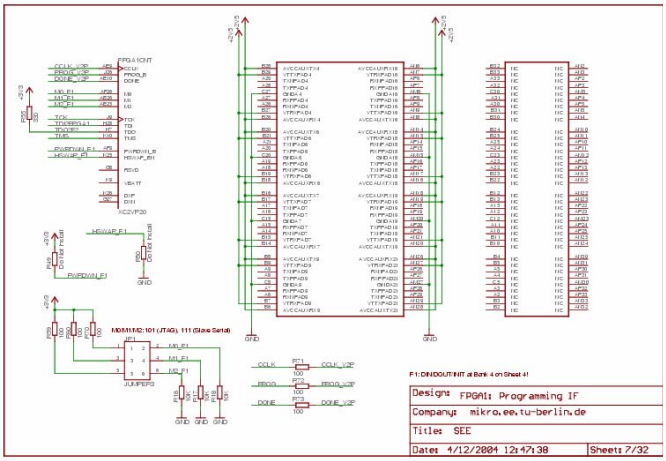


Sheet 4/32

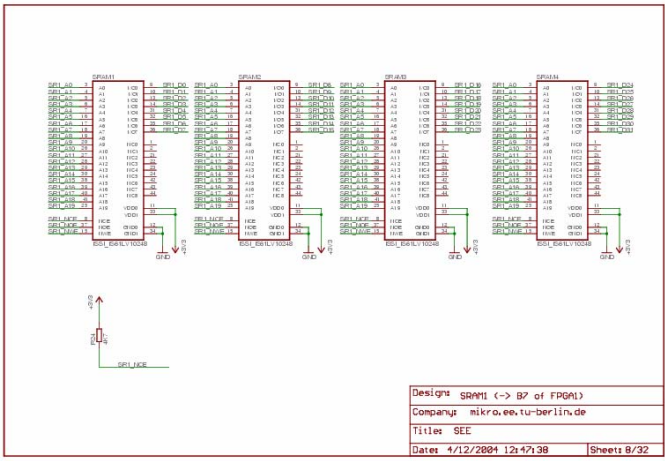
Sheet 5/32



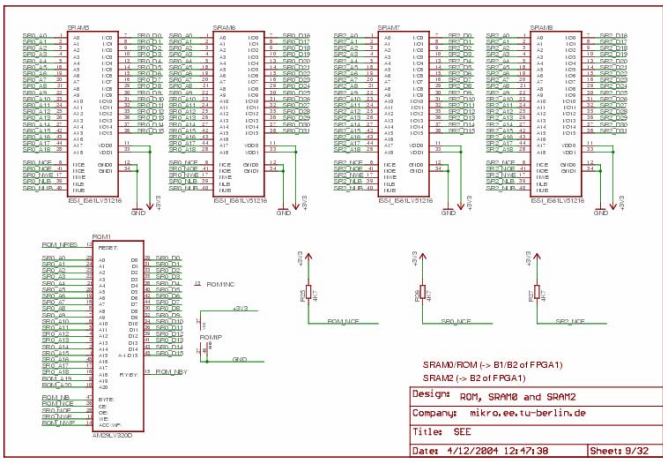
Sheet 6/32



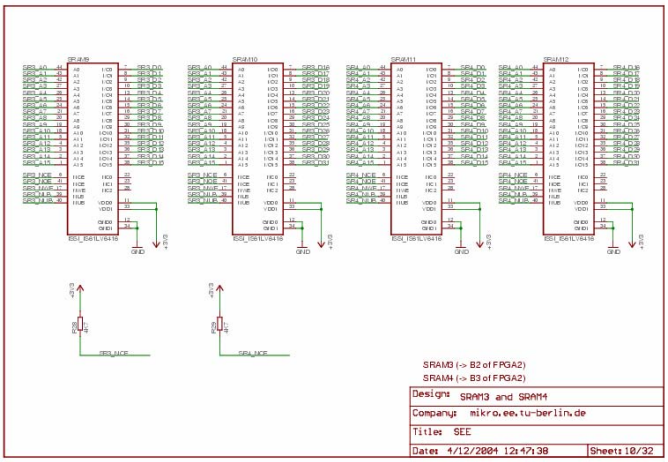
Sheet 7/32



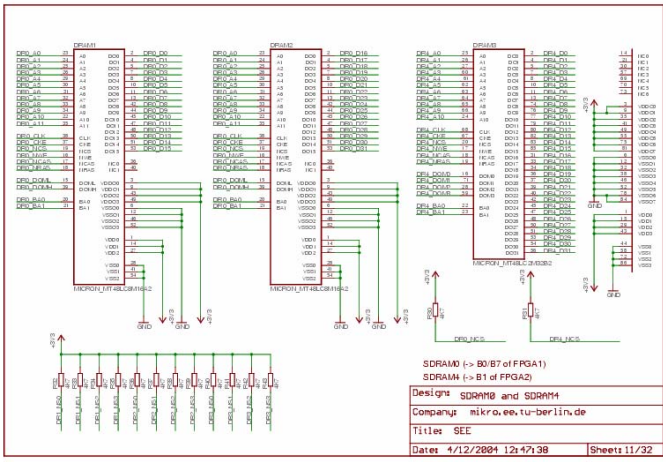
Sheet 8/32



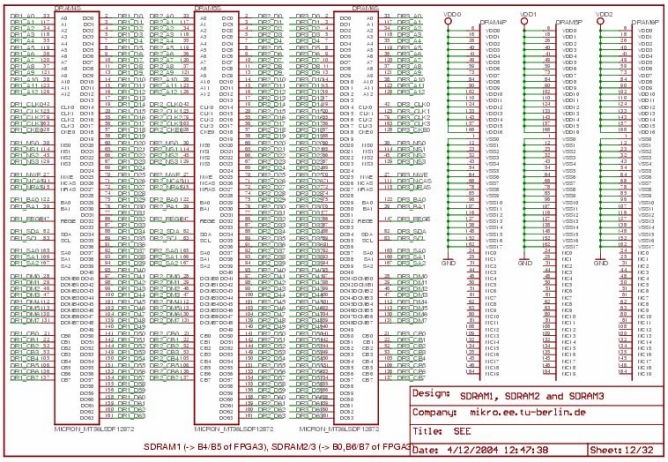
Sheet 9/32



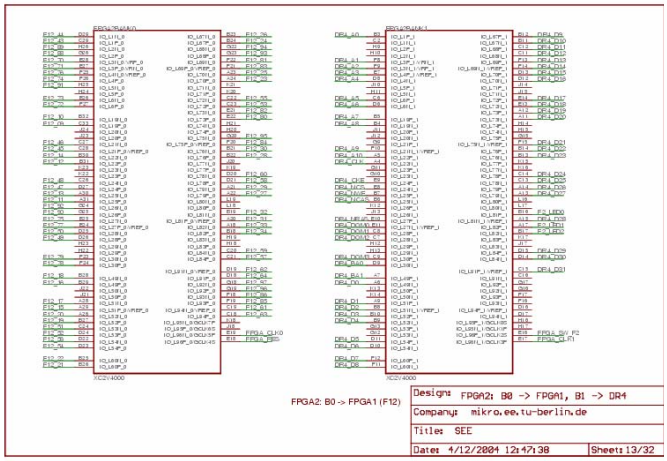
Sheet 10/32



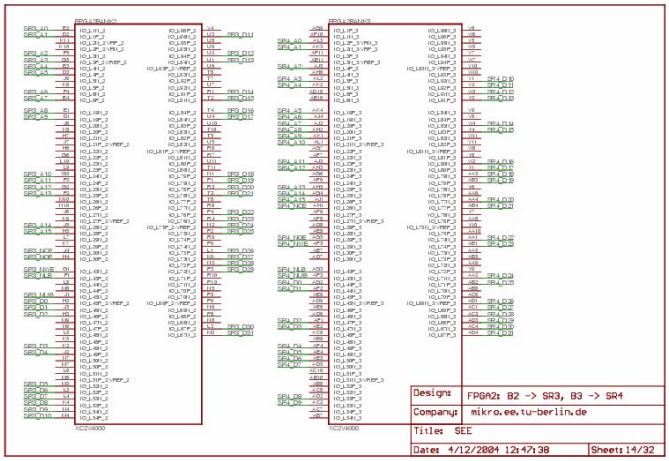
Sheet 11/32



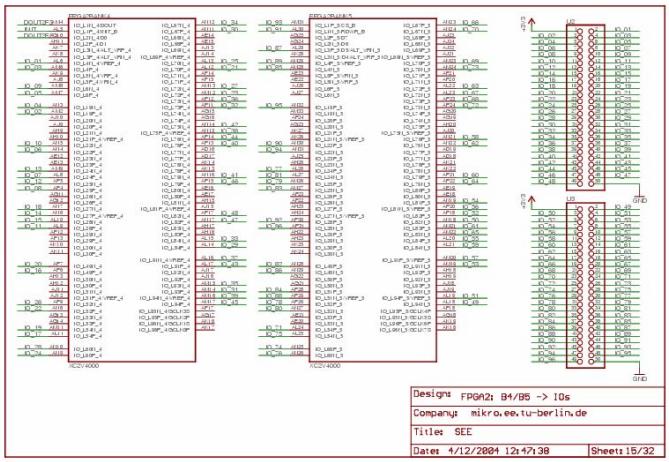
Sheet 12/32



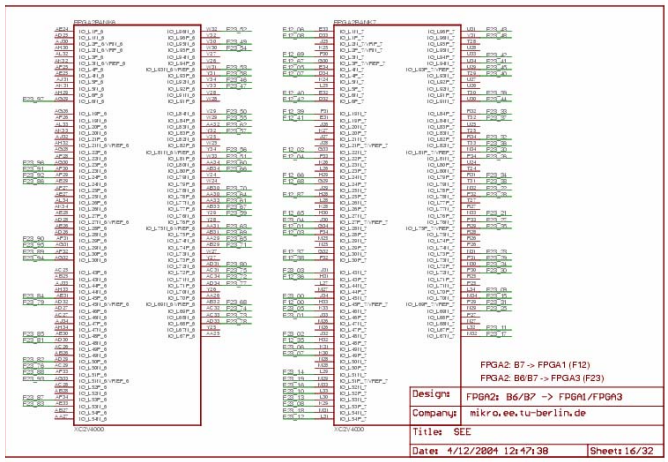
Sheet 13/32



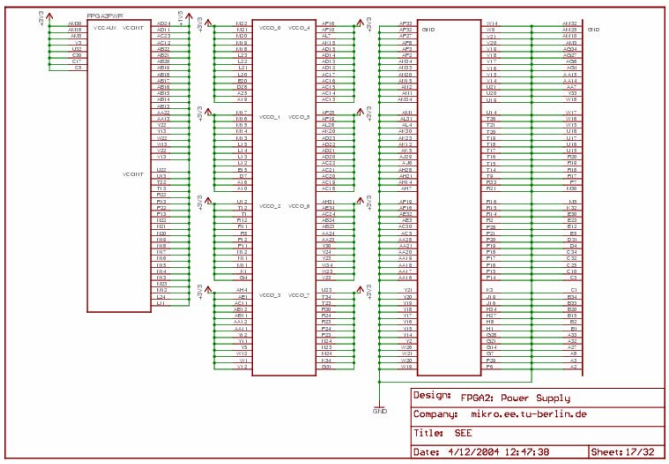
Sheet 14/32



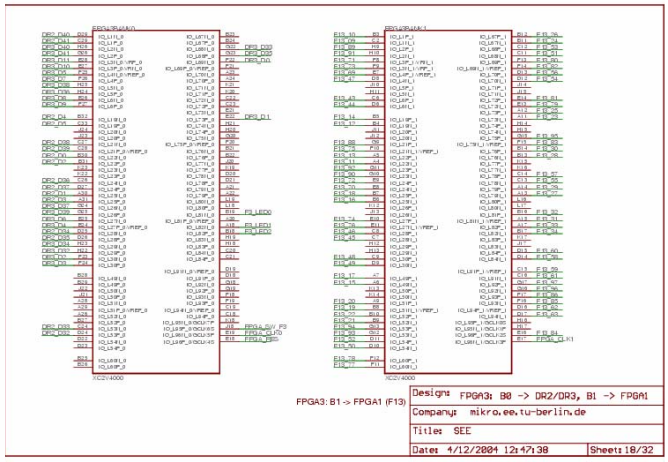
Sheet 15/32



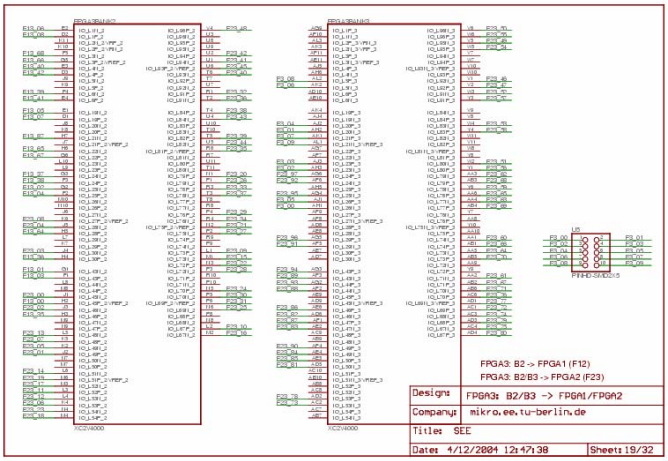
Sheet 16/32



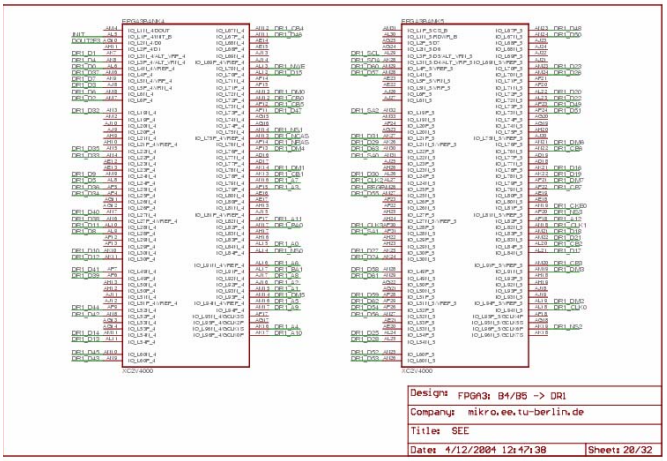
Sheet 17/32



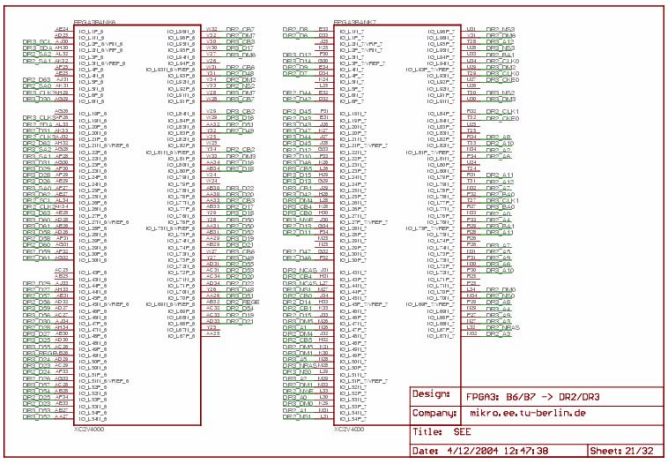
Sheet 18/32



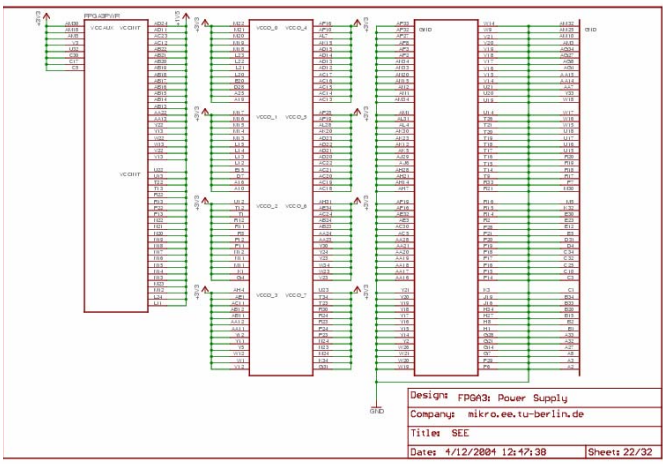
Sheet 19/32



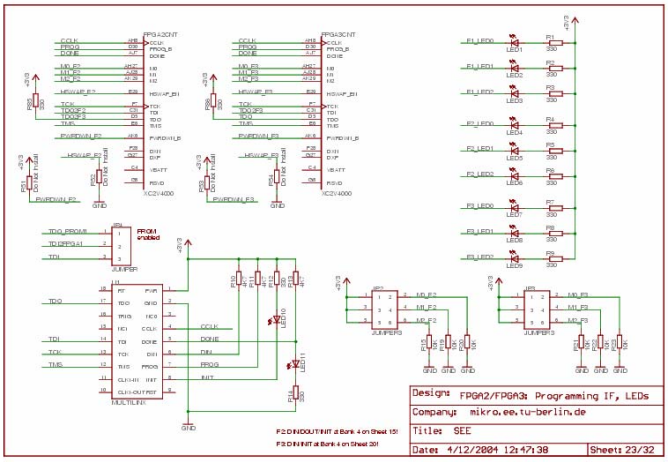
Sheet 20/32



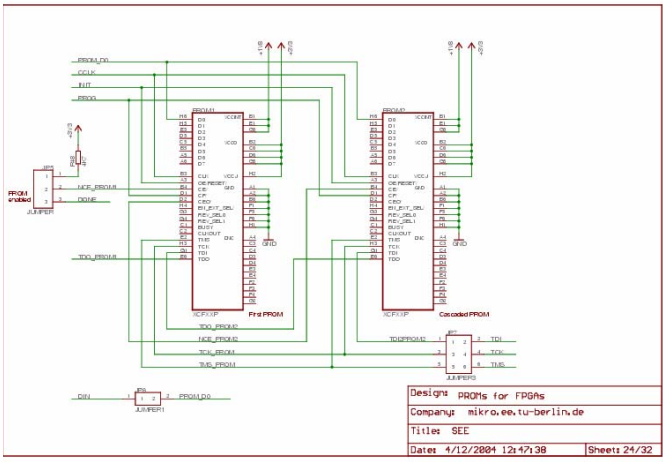
Sheet 21/32



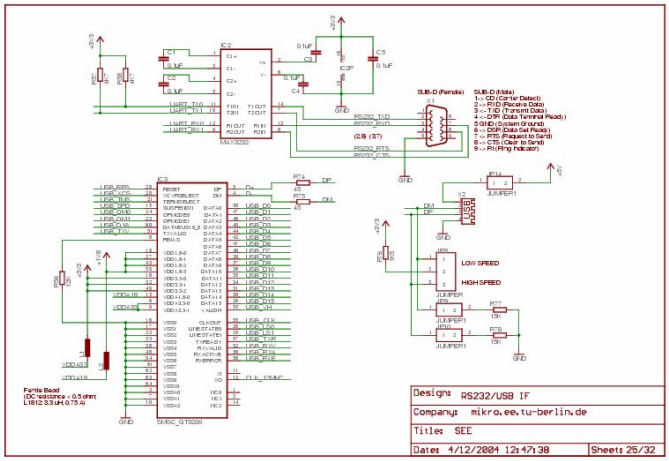
Sheet 22/32



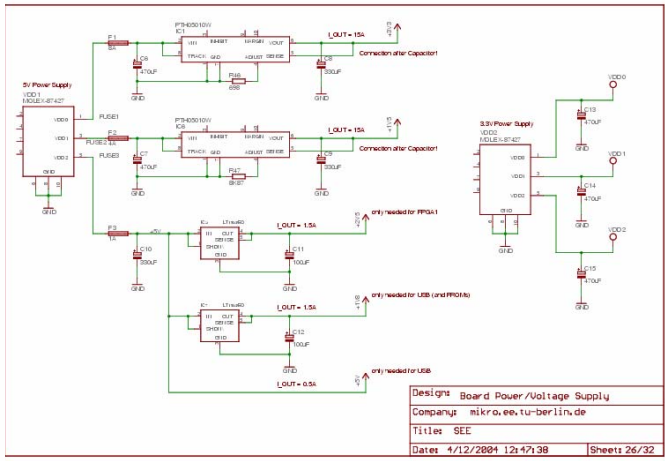
Sheet 23/32



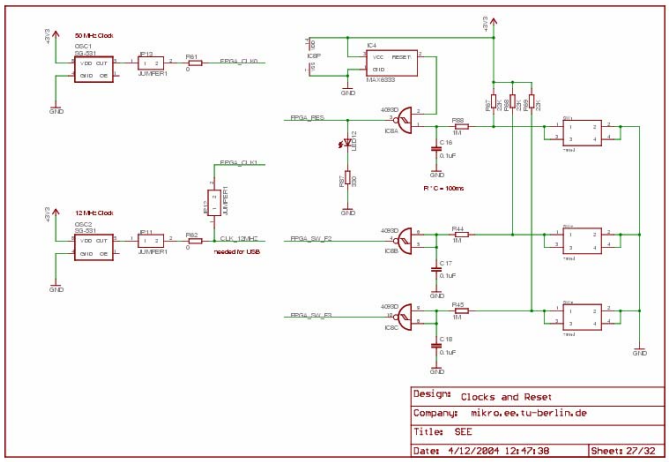
Shet 24/32



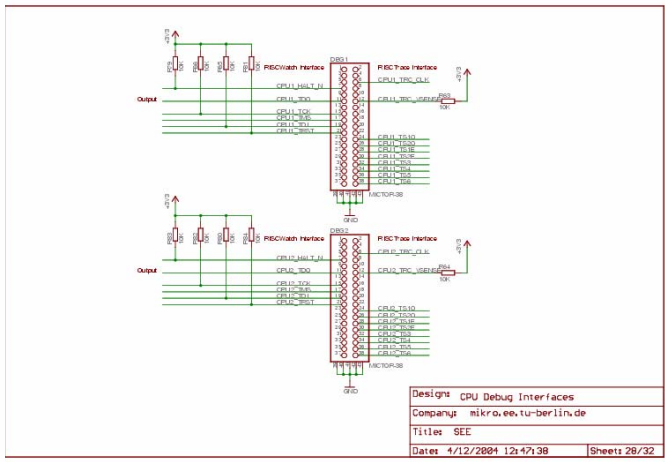
Sheet 25/32



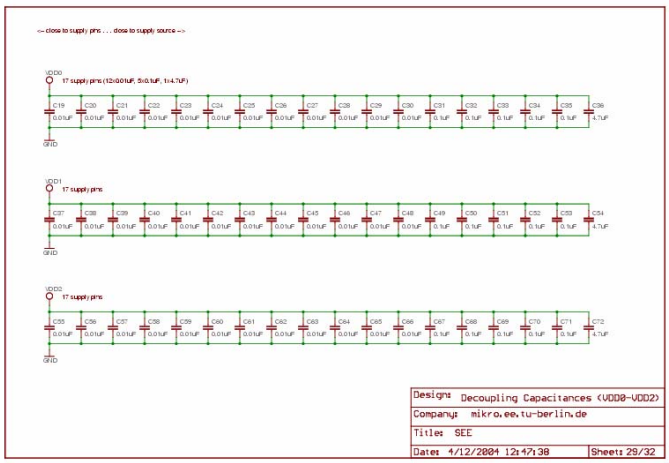
Sheet 26/32



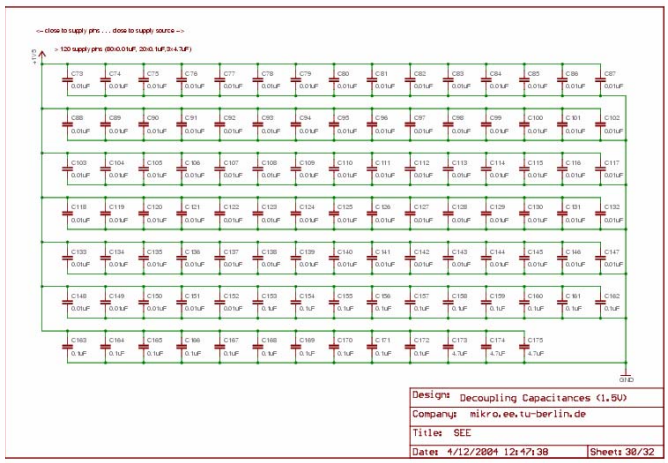
Sheet 27/32



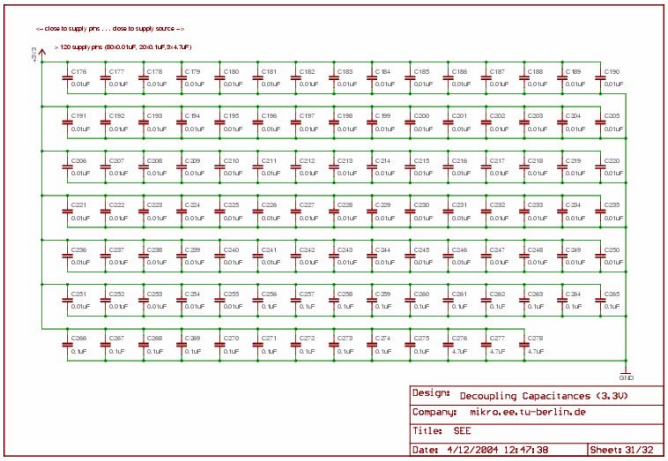
Sheet 28/32



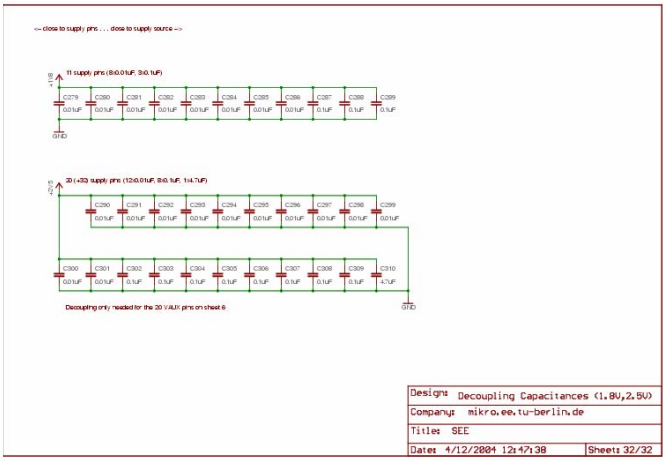
Sheet 29/32



Sheet 30/32

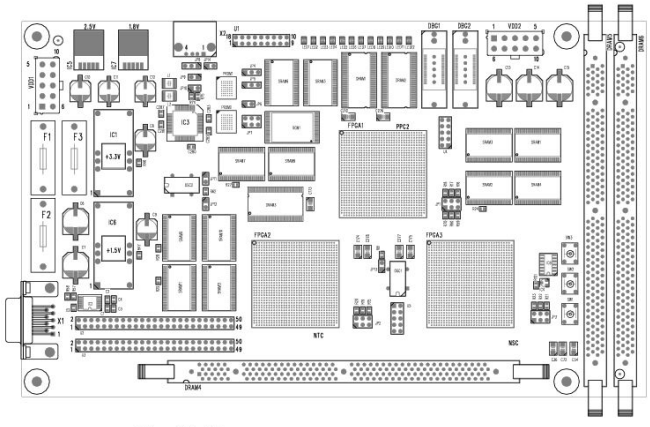


Sheet 31/32

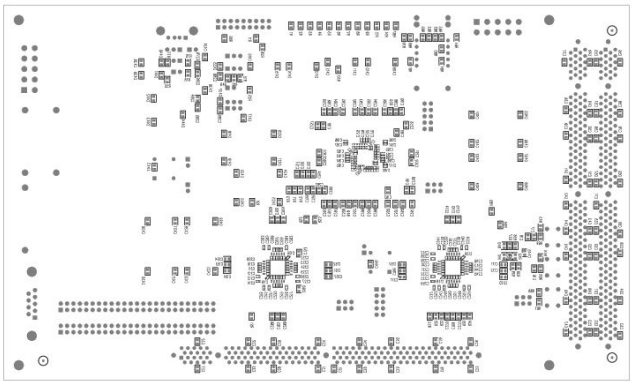


Sheet 32/32

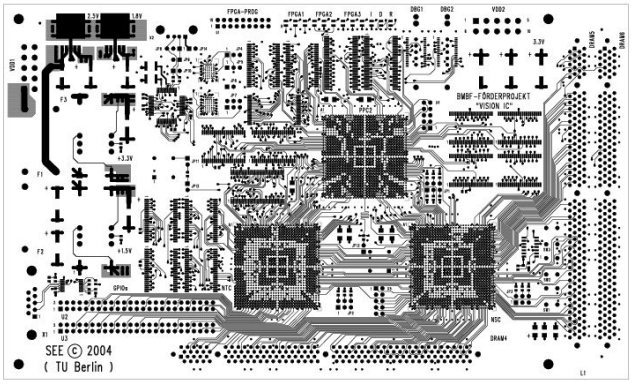
Anhang L : Entflechtung der Prototypen-Platine



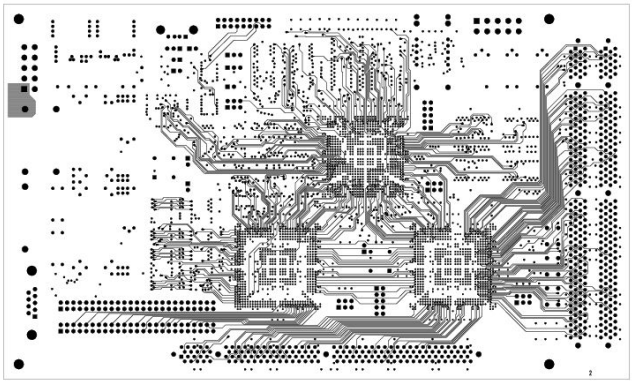
Vorderseite



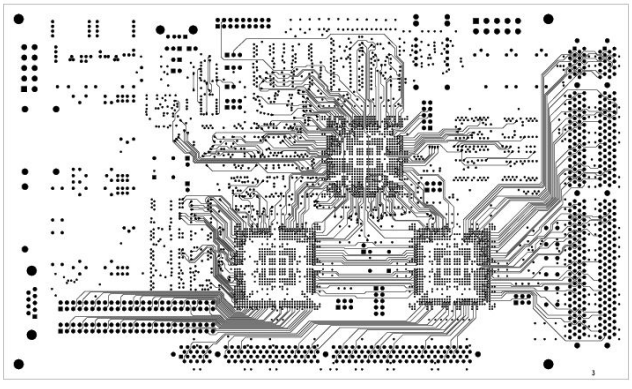
Rückseite



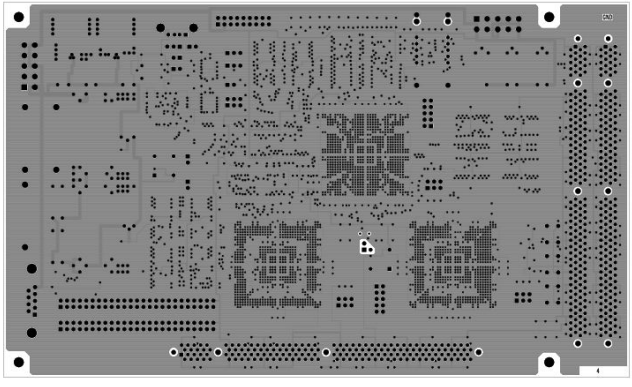
Lage 1/10



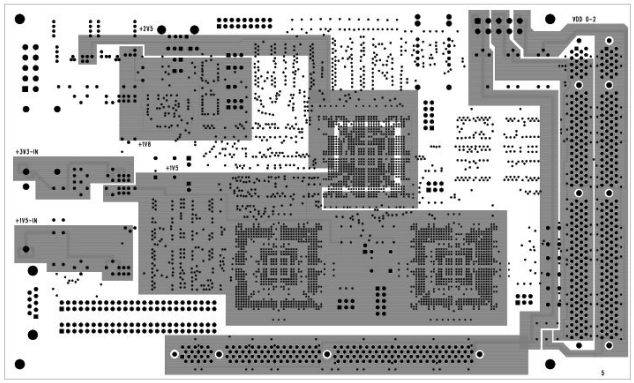
Lage 2/10



Lage 3/10

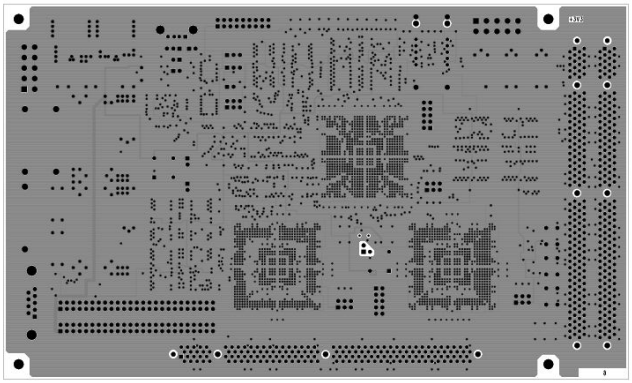


Lage 4/10



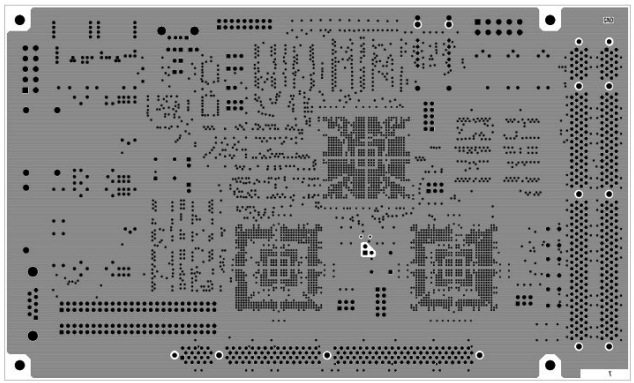
Platinenmass 350,00 x 150,00 mm
CAD-UL TUB01/0 18-06-04 LAYER 5 POW

Lage 5/10



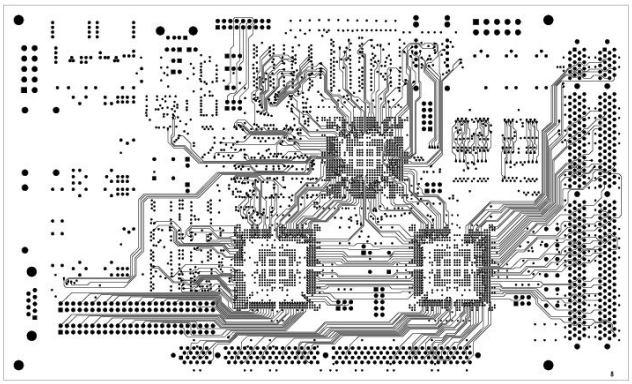
Platinenmass 350,00 x 150,00 mm
CAD-UL TUB01/0 18-06-04 LAYER 6 +3V3

Lage 6/10



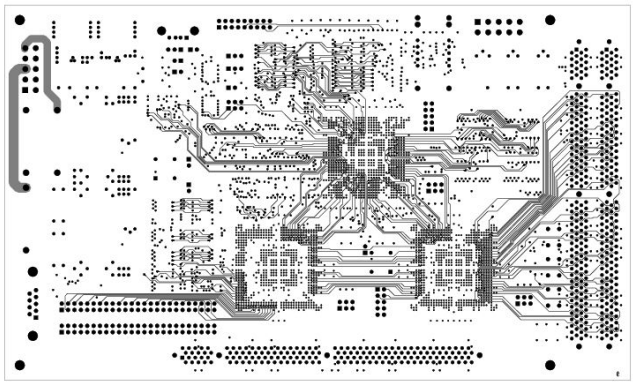
Platinenmass 350,00 x 150,00 mm
CAD-UL TUB01/0 18-06-04 LAYER 7 GND

Lage 7/10



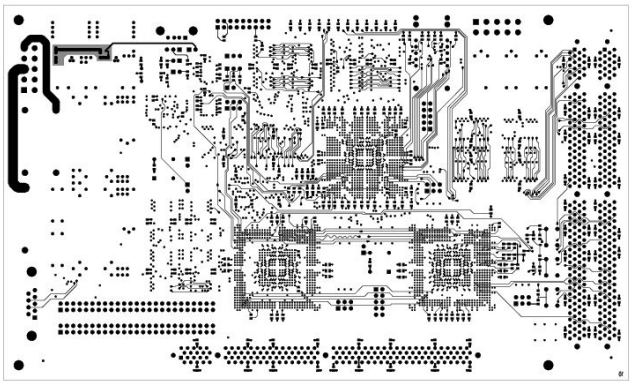
Platinenmass 350,00 x 150,00 mm
CAD-UL TUB01/0 18-06-04 LAYER 8 SIG4

Lage 8/10



Platinenmass 350,00 x 150,00 mm
CAD-UL TUB01/0 18-06-04 LAYER 9 SIG5

Lage 9/10



Platinenmass 350,00 x 150,00 mm
CAD-UL TUB01/0 18-06-04 LAYER 10 LOET/SIG6

Lage 10/10

Danksagung

Diese Dissertation entstand im Rahmen des vom BMBF (Bundesministerium für Bildung und Forschung) geförderten Forschungsprojektes “Vision-IC“ unter der Projektleitung der Forschungsabteilung CPR ST (Corporate Research System Technology) der Infineon Technologies AG. In diesem Forschungsprojekt waren neben der Infineon Technologies AG 6 weitere Industrievertreter (Viisage Technology AG, EUROS Embedded Systems GmbH, Regocnitec Gesellschaft für digitale Bildverarbeitung mbH, GeSiM (Gesellschaft für Signalverarbeitung und Mustererkennung) mbH, Technische Beratung Dr.-Ing. Giora Baum und Software-Entwicklung Stefan Giessler), 2 Institute der Fraunhofer Gesellschaft (IPK (Institut für Produktionsanlagen und Konstruktionstechnik) und HHI (Institut für Nachrichtentechnik Heinrich-Hertz-Institut)) und 5 Universitäten (Universität Bochum (Prof. C. von der Malsburg), Universität Ulm (Prof. H.-J. Pfleiderer, Prof. E. Kohn), Universität-GH-Paderborn (Prof. U. Rückert), Technische Universität Dresden (Prof. R. Schüffny) und Technische Universität Berlin (Prof. H. Klar)) involviert. Das gemeinsame Projektziel bestand in der Erforschung von intelligenten und flexiblen “Vision”-Plattformen für den Einsatz in Massenmarkt-Anwendungen.

Mein besonderer Dank gilt meinem Doktorvater Prof. Dr.-Ing. Heinrich Klar, der mir die Möglichkeit gab, auf einem sehr interdisziplinären, herausfordernden und zukunftweisenden Forschungsgebiet im Bereich des Entwurfs integrierter Digitalsysteme tätig zu sein. Prof. Dr.-Ing. Ulrich Kleine danke ich für die Bereitschaft zur Übernahme des Zweitgutachtens.

Prof. Dr. rer. nat. Ulrich Ramacher und Dr.-Ing. Arne Heittmann von der Infineon Technologies AG danke ich für ihr Engagement und ihre klaren Zielvorgaben im Rahmen des BMBF-Projektes. Die zahlreichen Treffen im kleineren Kreis zusammen mit der Universität-GH-Paderborn und der Technischen Universität Dresden in Berlin und Dresden boten stets Gelegenheit für einen regen Wissensaustausch und für eine regelmäßige erforderliche Darstellung der eigenen Forschungsarbeiten.

Ganz herzlich möchte ich mich bei meinen Kollegen an der TU Berlin René Hartmann, Uwe Voß und Werner Eschenberg bedanken, die mir stets mit Rat und Tat bei technischen Fragen und Problemen zur Seite standen. Meinem Kollegen bei der DR. JOHANNES HEIDENHAIN GmbH Dr.-Ing. Richard Ižák danke ich sehr für die fachlichen Korrekturvorschläge aus der Sicht der analogen Schaltungsintegration von pulscodierten Neuronen und dem Interesse an der Fertigstellung dieser Arbeit.

An dieser Stelle möchte ich auch den Diplomanden Claudia Sannelli, Marco Rafanelli, Martin Geike, Patrick Griep und Philipp Mahr danken, die im Rahmen dieser Dissertation mitgewirkt haben. Es war eine große Unterstützung zu wissen, dass man als Gruppe und nicht nur als Einzelperson an technischen Problemstellungen arbeitet. Die zahlreichen gemeinsamen Diskussionen vor der weißen Tafel haben mir stets Freude bereitet und meine Ansicht bestärkt, dass sich Lösungen häufig im Verlauf eines Gruppengespräches ergeben.

Ich danke dem BMBF für die Finanzierung meiner Forschungsarbeiten, die sowohl Reisen zu Konferenzen und Fachtagungen als auch die Realisierung der Prototypen-Platine beinhaltete. Für die Unterstützung bei der Entflechtung und Bestückung bin ich Willi Maurer und Dirk Backhaus von der CAD-UL GmbH und für die Herstellung der Prototypen-Platine der Rinde GmbH zu Dank verpflichtet.

Schließlich möchte ich mich noch bei meiner Familie bedanken, die immer Verständnis für meine zeitintensiven Forschungsarbeiten hatte und regelmäßiges Interesse an dem Fortschritt der Arbeiten zeigte. Mein Dank gilt insbesondere meiner Frau Claudia, die mich mit ihrem Korrekturlesen bei der Anfertigung dieser Dissertation sehr unterstützt hat.

Lebenslauf

PERSÖNLICHE DATEN:

Name	Heik Heinrich HELLMICH
Geburtsdatum	21. März 1973
Geburtsort	Muroran/Hokkaido (Japan)
Eltern	Dipl.-Ing. Hans Hellmich und Kaoru Hellmich geb. Ikemi
Staatsangehörigkeit	Deutsch
Familienstand	seit 08/1998 verheiratet mit Claudia Hellmich geb. Hagen, 4 Kinder

SCHULISCHER WERDEGANG:

1979 - 1981	Grundschule in Siegburg/Kaldauen (Rhein-Sieg-Kreis)
1981 - 1987	Deutsche Schule in Tokyo (Japan)
1987 - 1992	Dietrich-Bonhoeffer-Gymnasium in Wiehl (Oberbergischer Kreis)
1992	Abitur

ZIVILDIENST:

1992 - 1993	Rettungssanitäter bei der Johanniter-Unfall-Hilfe

STUDIUM:

1993 - 1998	Elektrotechnik an der Technischen Universität Berlin
03/1998 - 07/1998	Diplomarbeit mit dem Titel “Analysis and design of source-coupled logic gates suitable for high-frequency, low noise applications” am Dipartimento di Elettronica, Informatica e Sistemistica an der Universität von Bologna (Italien, Prof. G. Baccarani), finanziert durch das ERASMUS-Stipendium der TU Berlin

BERUFLICHER WERDEGANG:

11/1998 - 04/2002	Wissenschaftlicher Mitarbeiter (mit Lehraufgaben) am Institut für Mikroelektronik und Festkörperelektronik im Fachbereich 12 der Technischen Universität Berlin (Prof. O. Manck) Betreute Lehrveranstaltungen: • Komponenten digitaler Systeme : 2 SWS UE • System-on-Chip Mikroprozessor (ARM) Projekt : 4 SWS PJ • Praktikum Digitale Systeme (Mikrocontroller-Labor) : 4 SWS PJ • Grundlagen der Elektrotechnik I - Teil A+B : 4 SWS UE+PR
03/2000 - 08/2000	Forschungsaufenthalt im Rahmen des Forschungsprojektes “Low Power DSP IP for an ARM based SoC architecture“ am Department of Electronics and Electrical Engineering an der Universität von Edinburgh (Schottland, Dr. T. Arslan), finanziert durch den Erwin-Stephan-Preis der TU Berlin
05/2002 - 04/2005	Wissenschaftlicher Mitarbeiter (ohne Lehraufgaben) im Fachgebiet Mikroelektronik des Instituts für Technische Informatik und Mikroelektronik an der Fakultät IV der Technischen Universität Berlin im Rahmen des vom BMBF geförderten Forschungsprojektes “Vision-IC“ (Prof. H. Klar)
seit 07/2005	Entwicklungsingenieur bei der DR. JOHANNES HEIDENHAIN GmbH in Traunreut (Landkreis Traunstein)