INAUGURAL-DISSERTATION
zur
Erlangung der Doktorwürde

der
Naturwissenschaftlich-Mathematischen
Gesamtfakultät

der
Ruprecht-Karls-Universität
Heidelberg

vorgelegt von
Dipl.-Phys. Tillmann Schmitz
aus Kiel
Tag der mündlichen Prüfung: 31.05.2006

Evolution in Hardware

Eine Experimentalplattform zum parallelen Training analoger neuronaler Netzwerke

Gutachter: Prof. Dr. Karlheinz Meier
Prof. Dr. Reinhard Männer

Evolution in Hardware - Eine Experimentalplattform zum parallelen Training analoger neuronaler Netzwerke

Künstliche neuronale Netzwerke eignen sich aufgrund der in den Neuronen und Synapsen verteilten Rechenoperationen für eine parallele Implementierung in hochintegrierten Schaltungen. Durch die Kombination aus analoger Berechnung und digitaler Kommunikation können mit geringem Stromverbrauch hohe Rechengeschwindigkeiten skalierbar erreicht werden. Diese Geschwindigkeit ist aber nur praktisch nutzbar, wenn die neuronale Hardware in ein System eingebettet wird, das hohe Geschwindigkeiten sowohl bei der Ansteuerung als auch bei der Implementierung der Trainingsalgorithmen sicherstellt.

Diese Arbeit stellt eine Experimentalplattform vor, die einen Mikroprozessor zur Ausführung der Trainingsalgorithmen mit programmierbarer Logik und einem gemischt analog-digitalen neuronalen Netzwerkchip kombiniert. Die erste Hardwaregeneration basiert auf einer PCI-Karte, die in einem Standard-PC betrieben wird. Die zweite Generation bietet durch Verwendung eines FPGAs mit integriertem Mikroprozessor die gleiche Funktionalität auf einer einzigen Leiterplatte. Bis zu 16 dieser Netzwerkmodule können, verbunden über eine Backplane, parallel betrieben werden.

Durch die Implementierung des Trainings in Software sind beliebige Algorithmen realisierbar. Um das Training mit evolutionären Algorithmen zusätzlich zu beschleunigen, wurde ein Koprozessor entwickelt, der die daten- und rechenintensiven Teile des Trainings unter vollständiger Kontrolle durch die Software innerhalb der programmierbaren Logik ausführt. Anhand von praxisrelevanten Klassifikationsproblemen konnte gezeigt werden, dass die Ergebnisse dieses evolutionären Trainings sehr gut mit anderen Lösungen auf der Basis neuronaler Netzwerke konkurrieren können.

Evolution in Hardware - An Experimental Platform for the Parallel Training of Analog Neural Networks

Due to the fact that computing processes can be carried out in parallel in both neurons and synapses artificial neural networks are well suited for a parallel implementation in highly integrated circuits. By combining analog computing with digital communication, it is possible to build a scalable network with a high computation speed and low power consumption. To take full advantage of the neural hardware, it has to be embedded in a system that provides for a high-speed access as well as a fast implementation of the training algorithms.

This thesis describes an experimental platform consisting of a microprocessor to carry out the training algorithms, programmable logic and a mixed-signal neural network microchip. The first generation uses a PCI-Adapter within a standard PC. By utilizing an FPGA with an embedded microprocessor, the second generation offers the same functionality on a single printed circuit board. Up to 16 of those network modules can be used in one backplane in parallel.

Since the training is implemented in software, any kind of algorithms can be used. To accelerate the training with evolutionary algorithms, a specific coprocessor was developed to perform the computing intensive parts of the training in programmable logic and under full software control. By solving real-world classification tasks it was demonstrated that the results obtained with evolutionary algorithms using this platform are comparable to other neural network solutions.

Inhaltsverzeichnis

Einleitung

Biologische Nervensysteme sind gegenwärtigen Computern in vielen Aufgaben deutlich überlegen. Diese Überlegenheit zeigt sich nicht nur bei einzelnen Problemstellungen wie etwa der Mustererkennung, sondern vor allem auch im Vergleich weiterer Eigenschaften wie Fehlertoleranz, physikalische Größe, Energieverbrauch und Lernfähigkeit. Mit dem Entwurf von *künstlichen neuronalen Netzwerken* (KNN) wird versucht, den Aufbau und die Arbeitsweise und damit auch die Vorteile von biologischen Nervensystemen nachzubilden.

Die erste wichtige Arbeit zu KNN wurde 1943 von Warren McCulloch und Walter Pitts veröffentlicht [45]. Sie stellten ein mathematisches Modell auf und konnten zeigen, dass Netze, die aus binären Neuronen aufgebaut sind, jede logische Funktion berechnen können. Die Vorstellung eines deutlich stärker biologisch orientierten Netzwerkes, des *Perzeptrons*, von Rosenblatt 1958 [60] führte zu einem gesteigerten Interesse an künstlichen neuronalen Netzwerken. Später wurden die Einschränkungen von einschichtigen Perzeptrons [50] nachgewiesen, aber auch gezeigt, dass mehrschichtige Netzwerke jede kontinuierliche Funktion approximieren können [8].

Im Gegensatz zu herkömmlichen Computern, die auf jede Problemstellung neu programmiert werden müssen, sind KNN lernfähig: Sie sind in der Lage, die Lösung zu einer gegebenen Aufgabe automatisch anhand von Beispieldatensätzen zu finden. Da dieses Training im Allgemeinen in einem hoch iterativen Prozess abläuft, ist die Geschwindigkeit sowohl des Netzwerkes als auch der Trainingsalgorithmen von großer Bedeutung.

Die Mehrheit der Anwendungen simulieren KNN in Software auf einem Standard Computer. Alternativ werden KNN in Hardware implementiert, wodurch die inhärente Parallelität der in den Neuronen und Synapsen verteilten Rechenoperation besser ausgenutzt werden kann. Durch Einsatz von vielen einfachen Verarbeitungseinheiten kann so eine hohe Geschwindigkeit erreicht werden.

In der Electronic Vision(s) Gruppe am Kirchhoff-Institut für Physik in Heidelberg wurde eine Realisation eines neuronalen Netzwerkes in Hardware entwickelt: Der Netzwerkchip HAGEN (*Heidelberg AnaloG Evolvable neural Network*) versucht, die Vorteile von digitaler und analoger *Very-Large-Scale Integration* (VLSI) Technologie [65] zu verbinden. Hauptsächlich werden evolutionäre Algorithmen zum Training des HAGEN Chips eingesetzt. Da diese Optimierungsverfahren nach dem Vorbild der natürlichen biologischen Evolution keinerlei Gradienteninformation benötigen, können sie auch bei diskontinuierlichen Neuronen verwendet werden. Zudem können durch den hoch iterativen Einsatz im *Chip-in-the-loop*-Verfahren einerseits die analogen Variationen des Herstellungsprozesses ausgeglichen werden, andererseits kann sich der Algorithmus auf das analoge Rauschen einstellen.

Um die Geschwindigkeiten der spezialisierten Hardware praktisch nutzen zu können, muss der HAGEN Chip in ein Hardwaresystem eingebunden werden, das ihn mit Daten versorgt und die berechneten Ergebnisse analysiert und speichert.

Da neuronale Netzwerkchips auf jede Problemstellung neu trainiert werden, muss das umgebende System nicht nur die reine Ansteuerung, sondern auch die Implementation der Trainingsalgorithmen sicherstellen. Da aber sowohl die Entwicklung der Netzwerkchips, als auch die Trainingsalgorithmen selbst Gegenstand aktueller Forschung sind, sollte das Hardwaresystem möglichst modular, erweiterbar und variabel in der Benutzung sein.

In dieser Arbeit werden zwei Experimentalplattformen vorgestellt, die jeweils die Geschwindigkeit des gemischt analog-digitalen Netzwerkchips, die Vorteile von programmierbarer Logik und die Variabilität von Software verbinden: Die erste Generation, das *Darkwing*-System, vereint einen FPGA, SDRAM-Speicher und die Anschlüsse für den Netzwerkchip auf einer PCI-Karte, die in einem Standard-PC betrieben wird. Die zweite Generation, das *Nathan*-System, nutzt einen FPGA mit eingebettetem Mikroprozessor. Dadurch kann die gesamte Funktionalität eines PC-Systems der ersten Generation auf einer Leiterplatte zusammengefasst werden. Durch Portierung von Linux [55] auf den eingebetteten Mikroprozessor steht wie im Darkwing-System ein vollständiges Betriebssystem zur Verfügung. Bis zu 16 dieser *Nathan Netzwerkmodule* werden in einer Backplane parallel betrieben und können über serielle Hochgeschwindigkeitsverbindungen Datenpakete mit hoher Rate und niedriger Latenz untereinander austauschen. Dadurch ist es möglich, mehrere Netzwerkchips miteinander zu verbinden und so größere neuronale Netzwerke aufzubauen.

Beide Experimentalplattformen wurden innerhalb der Electronic Vision(s) Gruppe entwickelt. Im Rahmen dieser Arbeit wurde ausgehend von den in [5] [25] und [69] beschriebenen Leiterplattenlayouts und der Elektronik die Systemintegration, die programmierbare Logik und die hardwarenahe Software für beide Plattformen weiterentwickelt. Dabei werden im Nathan-System die in [55] dokumentierten Module zur Kommunikation zwischen den einzelnen Nathan Netzwerkmodulen verwendet. Die Software sowie die Durchführung der evolutionären Trainingsexperimente ist in [29] dokumentiert.

Ziel bei der Entwicklung der programmierbaren Logik und der Ansteuerungssoftware war eine größtmögliche Modularität und Erweiterbarkeit. Durch die Anbindung des PC-Prozessors bzw. des eingebetteten Mikroprozessors im Nathan-System können beliebige Trainingsalgorithmen variabel in Software beschrieben werden. Über spezielle Linux-Kernel-Treiber ist eine direkte Kommunikation zwischen der Software und der programmierbaren Logik möglich. Für das Training mit evolutionären Algorithmen wurde zudem eine spezielle Koprozessorarchitektur entworfen, welche die daten- und rechenintensiven Aufgaben in der programmierbaren Logik ausführt, dabei aber vollständig von der Trainingssoftware gesteuert wird.

Sowohl die Ansteuerungslogik für den Netzwerkchip, der evolutionäre Koprozessor als auch der Hauptspeicherzugriff des Mikroprozessors verwenden den gleichen externen Speicherbus. Ein modulares Design ist nur möglich, wenn diese Zugriffe unabhängig voneinander implementiert werden können. Daher wurde eine Speicheransteuerung entwickelt, die es einer beliebigen Anzahl von logischen Speicherschnittstellen erlaubt, simultan und mit hoher Transferrate die vorhandenen Speicherbausteine zu nutzen.

Neben einer Adaption von *Liquid Computing* [41] an die Möglichkeiten des Netzwerkchips [69] wird die vorgestellte Plattform und der HAGEN Chip zur Bildverarbeitung und Mustererkennung eingesetzt [19] [18]. Anhand von standardisierten Aufgabenstellungen für Klassifikationsprobleme konnte zudem gezeigt werden [29], dass die Ergebnisse des evolutionären Trainings mit Hilfe des Koprozessors mit anderen Lösungen neuronaler Netzwerke mehr als vergleichbar sind.

Diese Arbeit teilt sich in fünf Kapitel: Die Grundlagen und die Theorie neuronaler Netze, evolutionärer Algorithmen und programmierbarer Logik werden in Kapitel eins behandelt, zudem werden die Hardwarevoraussetzung der beiden Experimentalplattformen vorgestellt. Das zweite Kapitel gibt einen Überblick über die beiden Systeme und die Arbeitsaufteilung zwischen Software, programmierbarer Logik und Hardware. Kapitel drei beschreibt die Implementation der programmierbaren Logik sowie den Aufbau der hardwarenahen Software. Das vierte Kapitel fasst die vollständige Plattform zusammen und diskutiert die Leistungen, abschließend werden in Kapitel fünf die durchgeführten Experimente vorgestellt.

Kapitel 1

Grundlagen

1.1 Künstliche Neuronale Netzwerke

Künstliche Neuronale Netze sind Modelle, die den biologischen Nervensystemen nachempfunden sind. Dabei können zwei Zielsetzungen unterschieden werden: Zum einen ist das Interesse groß, durch künstliche Modelle oder Computersimulationen mehr über die Funktionsweise der biologischen Informationsverarbeitung zu lernen. Zum anderen wird versucht, durch Verwendung von biologischen Prinzipien künstliche Systeme zur Informationsverarbeitung zu entwerfen, die bessere Eigenschaften als herkömmliche Computer aufweisen.

1.1.1 Neuronenmodelle

Die erste wichtige Arbeit zu künstlichen neuronalen Netzen wurde 1943 von Warren McCulloch und Walter Pitts veröffentlicht [45]. Sie stellten ein vereinfachtes Neuronenmodell vor, dass beliebig viele binäre Eingänge, genau einen binären Ausgang und einen reellwertigen Schwellwert enthält. Die Eingänge können wie in biologischen Systemen erregend oder hemmend wirken. Sollte die Anzahl der aktiven, erregenden Eingänge den Schwellwert überschreiten und kein hemmender Eingang aktiv sein, so wird der Ausgang des Neurons auf aktiv geschaltet. McCulloch und Pitts konnten zeigen, dass Netze, die aus diesen "McCulloch-Pitts"-Neuronen aufgebaut sind, jede logische Funktion berechnen können.

Frank Rosenblatt erweiterte das Modell und prägte den Begriff *Perzeptron* [61]: Jeder Eingang wird mit einem reellwertigen Gewicht verknüpft, wobei hemmende Eingänge durch ein negatives Gewicht dargestellt werden (siehe Abb. 1.1). Obwohl beide Beschreibungen mathematisch äquivalent sind[1], ist es im Falle des Perzeptrons einfacher, das Netzwerk an Problemstellungen anzupassen, d.h. zu trainieren. Hier können neben der Topologie, d.h. der Art der Vernetzung, und den Schwellwerten auch die Gewichte verändert werden.

Sowohl McCulloch und Pitts als auch Rosenblatt verwendeten binäre Ein- und Ausgänge. Die Aktivierungsfunktion, nach der das Neuron entscheidet, welchen Wert der

[1] In [59] wird gezeigt, dass gewichtete Netze mit rationalen Gewichten und relativ hemmenden Verbindungen in ungewichtete Netze mit absolut hemmenden Verbindungen überführt werden können und umgekehrt.

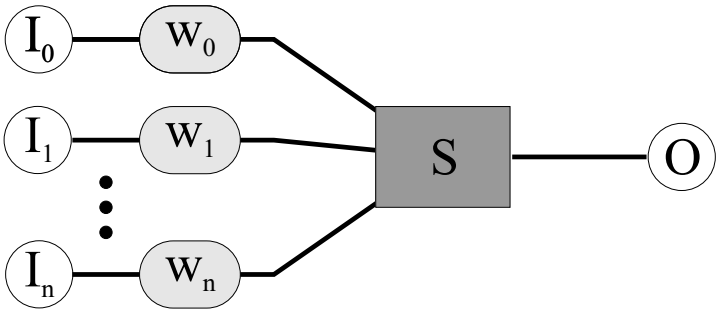

Abbildung 1.1: Definition eines Neurons im klassischen Perzeptron. Sowohl der Ausgang als auch alle Eingänge sind binär ($I_i,\ O \in \{0,1\}$), die Gewichte und der Schwellwert sind reell ($w_i,\ S \in \mathbb{R}$), der Ausgang nimmt genau dann den Wert 1 an, wenn gilt: $\sum_{i=1}^{n} I_i w_i > S$, sonst wird der Ausgang auf 0 geschaltet.

Ausgang annehmen soll, ist dann die Schwellwertfunktion. Dieses Modell kann auf kontinuierliche Ein- und Ausgänge erweitert werden, wobei die Aktivierungsfunktion z.B. schrittweise linear oder sigmoid sein kann.

Diese Arbeit beschäftigt sich mit den klassischen neuronalen Netzwerken und mehrschichtigen Perzeptrons. Die stärker biologisch orientierten pulsgekoppelten neuronalen Netzwerke [24] werden nicht behandelt.

1.1.2 Topologie

Eine größere Auswirkung auf die Fähigkeiten eines neuronalen Netzwerkes als das Neuronenmodell hat die Topologie, die Art der Vernetzung. Ein neuronales Netzwerk ist ein gerichteter Graph, die Neuronen stellen die Knoten und die Verbindungen zwischen Neuronen die Kanten dar. Es werden drei Arten von Neuronen unterschieden:

- Die Eingabeneuronen beziehen ihre Eingabe nur von außerhalb des Netzwerkes.
- Die Ausgabeneuronen übermitteln ihre Ausgabe nur nach außerhalb.
- Die versteckten Neuronen kommunizieren nur mit anderen Neuronen des Netzes, nicht mit der Außenwelt.

Vorwärtsgerichtete Netzwerke

Für ein vorwärtsgerichtetes Netzwerk gilt: Der zugrunde liegende gerichtete Graph hat keine gerichteten Zyklen, d.h. es gibt keinen Pfad, der vom Ausgang eines Neurons – evtl. über Umwege über andere Neuronen – zum Eingang desselben Neurons führt. Ohne Beschränkung der Allgemeinheit kann angenommen werden, dass ein vorwärtsgerichtetes Netzwerk in Schichten aufgebaut ist (Abb. 1.2): Neuronen der Schicht S_n erhalten nur Eingabedaten aus vorherigen Schichten, d.h. nur aus einer Schicht S_m, wenn $m < n$ gilt. Im weiteren wird die Eingabeschicht in der Zählung der Schichten nicht mitgerechnet, demzufolge zeigt Abb. 1.2 ein 3-schichtiges Netzwerk.

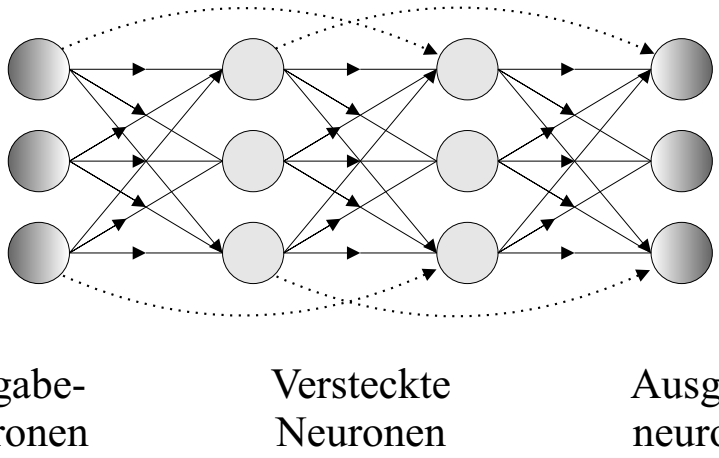

Abbildung 1.2: Netzwerktopologie eines 3-schichtigen vorwärtsgerichteten Netzwerkes. Die gepunkteten Verbindungen überspringen jeweils eine Schicht.

Rückgekoppelte Netzwerke

Ein rückgekoppeltes Netzwerk enthält Zyklen. Dadurch verwendet ein Neuron den eigenen Ausgang – evtl. über andere Neuronen – als Eingang. Mit der Annahme, dass alle Neuronen eine feste Zeit t_0 für ihre Berechnungen benötigen, ist der Ausgang des Netzwerkes nicht allein abhängig von der gegenwärtigen Eingabe, sondern es muss zusätzlich die Historie, d.h. die Zustände der Vergangenheit, betrachtet werden.

1.1.3 Der neuronale Netzwerkchip

Am Kirchhoff-Institut für Physik wurde in der Electronic Vision(s) Gruppe der neuronale Netzwerkchip HAGEN (*Heidelberg AnaloG Evolvable neural Network*) entwickelt [65]. Er implementiert ein neuronales Netzwerk und verbindet dabei die Vorteile analoger Berechnung mit der Zuverlässigkeit und Schnelligkeit digitaler Kommunikation. Vor der Ausführung der Netzwerkoperation muss das neuronale Netzwerk konfiguriert werden, indem die Topologie festgelegt und die Synapsengewichte geschrieben werden. Der HAGEN Chip ist darauf ausgelegt, sowohl die Konfiguration, als auch die anschließende Netzwerkberechnung mit hoher Geschwindigkeit auszuführen. Dabei können große neuronale Netzwerke mit niedrigem Stromverbrauch modelliert werden. Eine Mikrofotografie des HAGEN Chips zeigt Abb. 1.3.

Interne Blockstruktur

Der HAGEN Chip besteht aus vier Blöcken, jeder dieser Blöcke stellt ein vollständig verbundenes einschichtiges Perzeptron mit 128 binären Eingabe- und 64 binären Ausgabeneuronen dar. Die Signale der Ausgabeneuronen eines jeden Blockes können auf die Eingabeneuronen desselben oder eines anderen Blockes zurückgeführt werden. Dadurch ist es möglich, variabel nahezu beliebige mehrschichtige und auch rückgekoppelte Topologien zu realisieren.

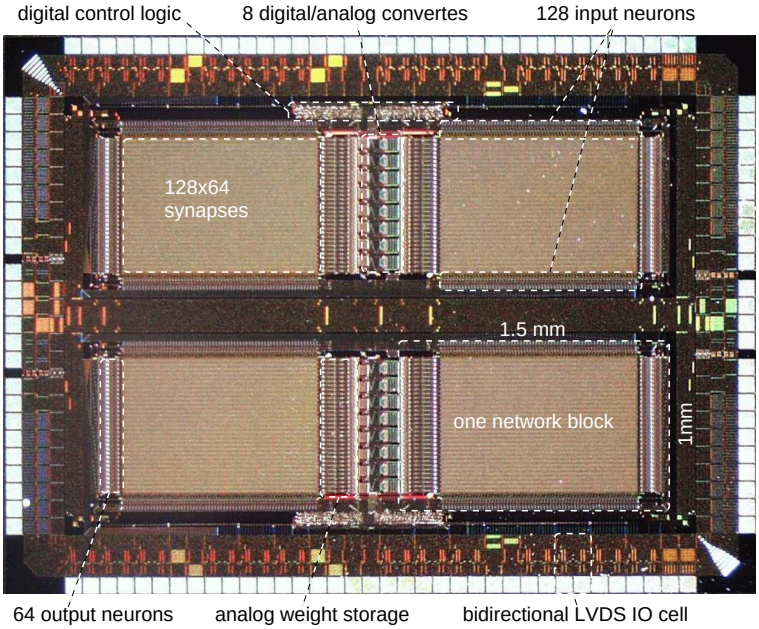

Abbildung 1.3: Mikrofotografie des HAGEN Chips, die vier Netzwerkblöcke mit je 8192 Synapsen sind deutlich zu erkennen [65].

Die Eingabesignale I_j und auch die Ausgabesignale O_i sind binär, die zugehörigen Gewichte ω_{ij} sind durch analoge Implementation kontinuierlich. Die Transferfunktion der Neuronen $g(x)$ ist durch die Stufenfunktion $\Theta(x)$ gegeben:

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

Die Beschränkung auf binäre Signale und damit auf eine relativ einfache Transferfunktion bietet mehrere Vorteile:

- Die Funktion der Synapsen und Neuronen ist ressourcensparend ohne Multiplikationen implementierbar. Das Neuron muss nur über alle Gewichte mit aktivem Eingabesignal summieren und die Schwellwertfunktion ausführen.

- Durch die binären Ein- und Ausgänge ist die Schnittstelle jedes Blockes zu den benachbarten Blöcken und auch die Schnittstelle des HAGEN Chips zu externen Bauteilen rein digital. Das vereinfacht die Kommunikation stark. Im Gegensatz zu analogen Signalen lassen sich digitale Signale leicht speichern, weiterverarbeiten und transportieren. Dadurch ist es möglich, mehrere Blöcke eines HAGEN Chips oder auch mehrere Chips parallel zu betreiben, um große komplexe Netzwerke aufzubauen. Die rein digitale Schnittstelle macht folglich das Netzwerk skalierbar.

- Wird für eine Problemstellung eine höhere Auflösung benötigt, so können mehrere Binärsignale zusammengefasst werden, um Integerneuronen zu realisieren [67].

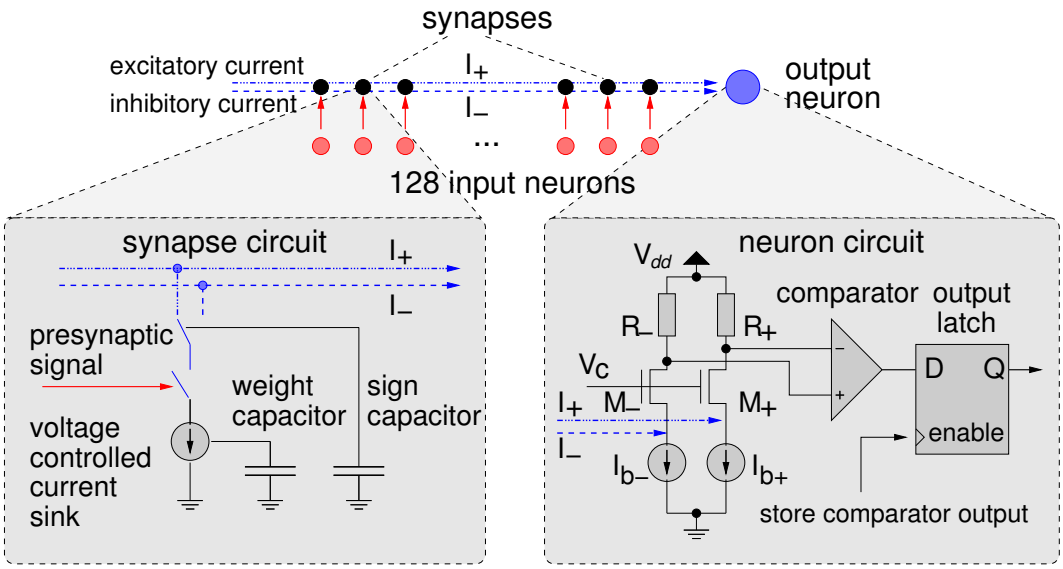

Abbildung 1.4: Operationsprinzip der Synapsen und Neuronen des HAGEN Chips[65]

Das Operationsprinzip der analogen Synapsen und Neuronen ist in Abb. 1.4 dargestellt. Synapsengewicht und -vorzeichen werden während der Konfiguration des Netzwerkes geschrieben und sind auf Kapazitäten gespeichert. Bei aktivem präsynaptischen Signal wird ein Strom, der proportional zur Spannung am Gewichtskondensator ist, von der erregenden oder hemmenden Leitung gezogen. Welche Leitung gewählt wird, entscheidet das Synapsenvorzeichen. Die Ströme aller erregenden und aller hemmenden Synapsen werden auf separaten Signalleitungen addiert. Das Neuron vergleicht die erregende mit der hemmenden Leitung und schaltet das Ausgaberegister bei stärkerem erregendem Strom auf 1, ansonsten bleibt es 0. Die Netzwerkoperation erfolgt taktgesteuert mit einer maximalen Frequenz $f_{net} = 1/\Delta t = 50$ MHz.

Programmierung und Speicherung der Gewichte

Für die Speicherung der Gewichte innerhalb der analogen Blöcke sind verschiedene Mechanismen möglich. Wie aus Abb. 1.4 ersichtlich, wurde die dynamische Speicherung als Ladung auf dem Gewichtskondensator gewählt. Nachteilig ist die Tatsache, dass jeder Gewichtskondensator mit der Zeit durch Leckströme seine Ladung verliert. Deshalb müssen die Gewichte außerhalb des HAGEN Chips gesichert und die Gewichtsmatrix in regelmäßigen Zeitabständen neu geschrieben, sozusagen aufgefrischt werden. Da diese Zeitabstände aber mit $10-100\ ms$ um viele Größenordnungen länger sind als die Dauer eines Netzwerktaktes, beeinflusst diese Auffrischung die Geschwindigkeit der Netzwerkoperation nicht nennenswert.

Um die Schnittstelle des HAGEN Chips digital zu halten, sind die *Digital-Analog-Converter* (DAC) zur Umwandlung der digitalen Werte in analoge Spannungen innerhalb des HAGEN Chips realisiert. Abb. 1.5 zeigt eine Hälfte des HAGEN mit den zugehörigen Anschlüssen und der DAC-Einheit. Sie besteht aus 8 DACs mit einer Umwandlungszeit von

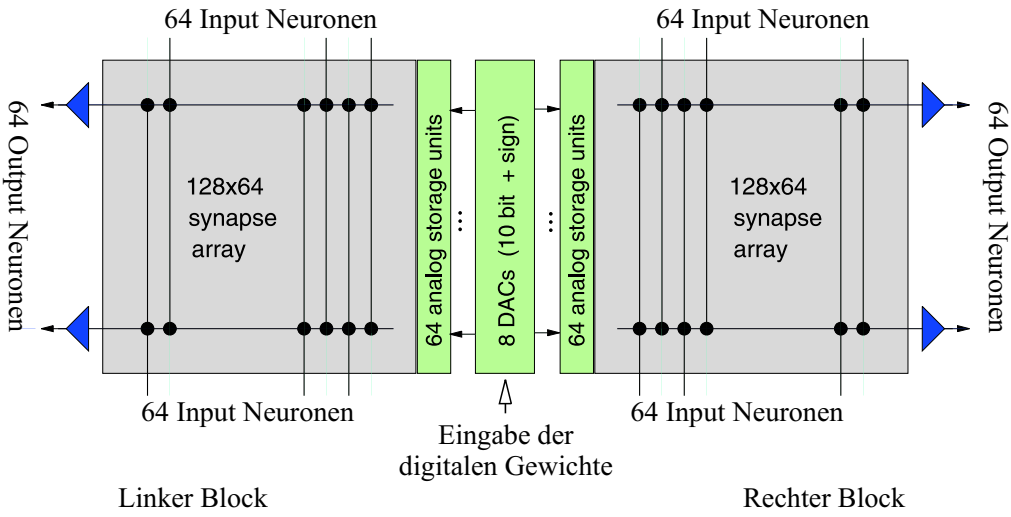

Abbildung 1.5: Schema der unteren Hälfte des HAGEN Chips mit zwei Blöcken, die sich eine DAC Einheit teilen [65].

je 40 *ns* und einer Genauigkeit von 10 Bit. Zusätzlich wird das Vorzeichen, das entscheidet, ob die Synapse erregend oder hemmend wirkt, verarbeitet. Eine Reihe von digitalen und analogen Zwischenregistern ermöglicht während der Konfiguration der Gewichte ein hohes Maß an Parallelität, so können z.B. bereits neue digitale Daten zur DAC-Einheit geschrieben werden, während die DACs bereits die vorherigen Werte umwandeln. Dadurch kann eine kontinuierliche Auslastung der DACs erreicht werden.

Schnittstelle

Der HAGEN Chip teilt sich in eine obere und eine untere Hälfte, beide können unabhängig voneinander konfiguriert und betrieben werden. Die physikalische Schnittstelle verwendet den *Low Voltage Differential Signal* (LVDS) Signalstandard. Jede der beiden Hälften verfügt über 8 bidirektionale Datenleitungen, 3 Adress- und eine Taktleitung, die in Richtung zum HAGEN Chip betrieben werden, sowie eine Taktleitung, die vom HAGEN zurück getrieben wird. Datentransfer zum Netzwerkchip wird im Folgenden als *Schreiben*, Datentransfer vom HAGEN Chip als *Lesen* bezeichnet. Eine umfangreiche Beschreibung der elektrischen Eigenschaften der physikalischen Schnittstelle findet sich in [69]. Eine intensive Betrachtung der logischen Schnittstelle und der Implementation in programmierbarer Logik erfolgt in Kapitel 3.
Tabelle 1.1 fasst die wichtigsten Daten des HAGEN Chips zusammen:

Leistungsfähigkeit

Im Folgenden soll eine grobe Abschätzung der Leistungsfähigkeit des HAGEN Netzwerkchips gegeben werden. Die Konfigurationszeit des Netzwerkes, d.h. die Aktualisierung der 4 Gewichtsmatrizen der 4 Blöcke ist durch die Umwandlungszeit der DACs begrenzt. Die je 8 DACs der unteren und der oberen Hälfte können in 40 ns zusammen 16 analoge Ge-

Prozess Eigenschaften	0.35 μm, 1 poly, 3 metal
Anzahl Blöcke / Neuronen / Synapsen	4 / 256 / 32768
Netzwerk Frequenz f_{net}	max. 50 MHz
Konfigurationsrate	max. 400 Millionen Gewichte/s
Transferrate der Schnittstelle	max. 11.4 Gigabit/s
Chipgröße	4.1×3 mm^2
Versorgungsspannung	3.3 V
Netzwerkverbindungen/s	1.64 TeraCPS (Connections per Second)
Genauigkeit der Gewichte	10 bit (nominal) + sign

Tabelle 1.1: Eigenschaften und Betriebsdaten des HAGEN Chips

wichte schreiben, damit beträgt die minimale Programmierzeit für alle 32768 Gewichte 82 μs und die Rate, mit der die Gewichte aktualisiert werden können, ist 400 Millionen Gewichte/s. Werden die Auffrischzyklen, die aufgrund der dynamischen Speicherung der Gewichte auf Kapazitäten notwendig sind, im Zeitabstand von 10 ms durchgeführt, sinkt dadurch die Verfügbarkeit des Netzwerkes um weniger als 1 %.

Um die Leistungsfähigkeit des HAGEN Chips einzuordnen, wird ein grober Vergleich mit der Rechenleistung eines aktuellen Prozessors angestellt, dabei wird im Wesentlichen der Argumentation aus [29] gefolgt. Die maximale Frequenz der Netzwerkberechnung des HAGEN Chips beträgt $f_{net} = 50$ MHz, die 4 Blöcke berechnen jeweils

$$128 \cdot 64 \cdot 50 \text{ MHz} = 4,1 \cdot 10^{11} \textit{ Connections Per Second} \text{ (CPS)}. \tag{1.2}$$

Unter der Annahme, dass ein aktueller Mikroprozessor die Funktionsweise des HAGEN Chips simuliert und in der Lage ist, 8 Integerrechenoperationen mit 16 Bit Genauigkeit pro Takt bei einer Taktrate von 3,6 GHz auszuführen, summiert sich das auf

$$8 \cdot 3,6 \text{ GHz} = 2,9 \cdot 10^{10} \text{ CPS}. \tag{1.3}$$

Dieser Vergleich zeigt, dass ein Block des HAGEN Chips gegenüber einem aktuellen Mikroprozessor einen Geschwindigkeitsvorteil von etwa einer Größenordnung besitzt, und das, obwohl der Mikroprozessor in der Fertigung einen 0,08 μm Prozess verwendet, während die Strukturbreite des HAGEN Chips von 0,35 μm in den Jahren 1995-97 aktuell war, inzwischen also für Mikroprozessoren veraltet ist. Der Vergleich vernachlässigt allerdings folgende Punkte:

- Die Leistungszahlen des HAGEN Chips beziehen sich auf ein Netzwerk, dass vollständig vernetzt ist, d.h. alle Synapsen verwendet. Für ein spärlich kodiertes Netzwerk sinkt die Rechenleistung entsprechend.
- Die Netzwerkverbindungen des HAGEN Chips verwenden eine Genauigkeit von 11 Bit im Vergleich zu 16 Bit des Mikroprozessors.
- In beiden Architekturen ist die Zeit vernachlässigt worden, die der Datentransfer sowohl der Eingabe- als auch der Ausgabedaten benötigt, ebenso die nötige Zeit zur Aktualisierung der Gewichte.

Deutlich besser als ein herkömmlicher Mikroprozessor schneidet der HAGEN Chip beim Vergleich der Leistungsaufnahme ab, gegenüber etwa 100 W eines Pentium IV Prozessors verbraucht ein Block des Netzwerkchips nur etwa 1 W [69].

1.2 Evolutionäres Training neuronaler Netzwerke

Im Gegensatz zu herkömmlichen Computern, die für eine Problemstellung programmiert werden, müssen bei neuronalen Netzwerken die Topologie und die Gewichte angepasst werden. Dies kann - wie in der Darstellung von McCulloch und Pitts - durch Konstruktion erfolgen, allerdings ist diese Möglichkeit für größere Netzwerke nicht praktikabel. Neuronale Netzwerke können mit Hilfe eines Lernalgorithmus und einer Reihe von Beispieldatensätzen die Lösung zu einer gestellten Aufgabe selbständig finden, d.h. lernen. Dieser Vorgang wird auch als *Training* bezeichnet. Dabei gehen die meisten Trainingsalgorithmen von einer festgelegten Topologie aus und optimieren nur die Gewichte der einzelnen Verbindungen bzw. die Schwellwerte der Neuronen.

Im Folgenden wird nur das *überwachte Lernen* betrachtet, dabei ist die richtige Ausgabe zu einer Reihe von Testdaten bekannt, der Trainingsalgorithmus kann folglich immer das Ergebnis der Netzwerkberechnung überprüfen und die Konfiguration der Gewichte entsprechend ändern. Das *unüberwachte Lernen* dagegen benötigt keinen Lehrer, dem die richtige Ausgabe bekannt ist. Das Netzwerk sucht selbständig nach wiederkehrenden Mustern in den Eingabedaten, d.h. ordnet die Eingabedaten verschiedenen Gruppen zu, man spricht von automatischer Klassifikation [36] [22].

Werden Netzwerk-Implementierungen in analoger Hardware betrachtet, so stellt sich ein Problem: Die genaue Funktion der Neuronen und Synapsen ist durch Variationen im Herstellungsprozess und durch zeitliches Rauschen der analogen Signale nicht genau bekannt. Folglich können gradientenbasierte Trainingsalgorithmen, die die genaue Form der Neuronenaktivierungsfunktion und den exakten Wert der Synapsengewichte benötigen, wie z.B. der Backpropagation-Algorithmus [80] [63], nur sehr eingeschränkt verwendet werden. Um solche analogen neuronalen Netzwerke zu trainieren, bietet sich das *Chip-in-the-loop* Verfahren an [78] [51]:

Die Trainingsdaten bestehen aus einer Reihe von *Eingabedaten*, zu denen die jeweils erwünschten Antworten des Netzwerkes, die *Solldaten*, bekannt sind. Mit diesen Datensätzen wird trainiert: Anfangs wird das Netzwerk mit zufälligen Gewichten initialisiert. Danach werden die Eingabedaten nacheinander präsentiert und die Ergebnisdaten des Netzwerkes beobachtet und mit den Solldaten verglichen. Falls die Ergebnisdaten falsch sind, werden die Gewichte entsprechend des Lernalgorithmus verändert und es werden wiederum die Eingabedaten präsentiert. Dies wird so lange ausgeführt bis die Fehler einen Zielwert unterschreiten oder die maximale Anzahl an Iterationen erreicht ist.

Da der Algorithmus im Chip-in-the-loop Training mit den echten Ergebnisdaten des neuronalen Netzwerkes arbeitet, kann er automatisch die Variationen des Herstellungsprozesses korrigieren bzw. sich auch auf das zeitliche analoge Rauschen einstellen. Durch die häufigen Iterationen und die beständige Neukonfiguration der Synapsengewichte ist allerdings ein hoher Datentransfer zwischen dem neuronalen Netzwerk und dem Trainingsalgorithmus vonnöten.

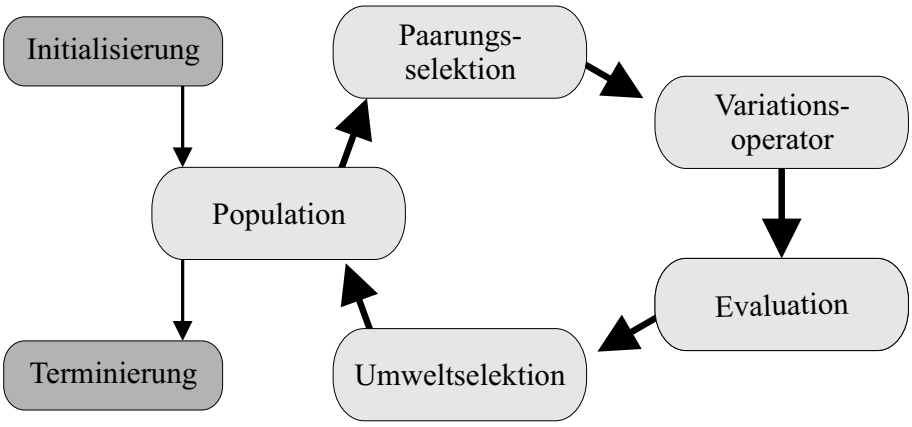

Abbildung 1.6: Funktionsprinzip eines evolutionären Algorithmus

Ein wichtiger Chip-in-the-loop Trainingsalgorithmus, der auf Prinzipien der natürlichen, biologischen Evolution basiert, wird im folgenden Abschnitt erläutert.

Evolutionäre Algorithmen

Das Prinzip der natürlichen Evolution wurde zuerst von Charles Darwin in [15] formuliert: Da Organismen sich beständig vermehren, der verfügbare Lebensraum aber begrenzt ist, setzen sich in einer Population diejenigen Individuen durch, die an den Lebensraum am besten angepasst sind und werden statistisch die meisten Nachkommen hervorbringen. Dadurch dass die erzeugten Nachkommen keine genauen Kopien ihre Eltern sind, sondern durch Rekombination und Mutation entstehen, entwickelt sich die Population weiter in Richtung der optimalen Anpassung. Die natürliche Evolution hat auf diese Weise nicht nur die Vielfältigkeit und Mannigfaltigkeit der Organismenwelt auf der Erde, sondern auch das bislang leistungsfähigste informationsverarbeitende System, das menschliche Gehirn, erschaffen. In den 60er Jahren entstanden drei verschiedene Methoden, um das erfolgreiche Prinzip der Evolution auf Optimierungsprobleme anzuwenden, aber seit Ende der 90er Jahre werden *evolutionary programming* [21], *genetic algorithm* [16] [32] und *evolutionary strategies* [57] [4] als drei Dialekte derselben Technologie angesehen.

Überblick Die Übertragung des Zyklus aus Reproduktion und Selektion auf die künstliche Evolution ist in Abb. 1.6 dargestellt. Es wird nicht ein Lösungskandidat sondern eine *Population* von *Individuen*, die mögliche Lösungen repräsentieren, betrachtet und optimiert. Die Güte jedes Individuums wird über eine Fitnessfunktion bestimmt. Die erste Population wird im Allgemeinen zufällig initialisiert. Für jedes Individuum der Anfangspopulation wird die Fitness bestimmt. Ausgehend von diesen Fitnesswerten werden in der Paarungsselektion (*parent selection*) diejenigen Individuen selektiert, die als Eltern für die Erzeugung der nächsten Generation verwendet werden. Durch Anwendung der Variationsoperatoren Mutation und Rekombination entstehen neue Lösungskandidaten, die – nach Ermittlung ihrer Fitness – miteinander und mit den Eltern um einen Platz in der neuen Generation konkurrieren. Da im Allgemeinen die Anzahl der Individuen innerhalb

der Population konstant gehalten wird, muss in einem zweiten Selektionsprozess, der Umweltselektion (*survivor selection*), entschieden werden, welche Individuen überleben und für die neue Generation ausgewählt werden. Ein evolutionärer Algorithmus setzt sich aus verschiedenen Komponenten zusammen:

Repräsentation Der erste Schritt zur Formulierung eines evolutionären Algorithmus zu einer gegebenen Problemstellung ist die Wahl einer geeigneten Repräsentation. Der *Phänotyp* beschreibt den Lösungskandidaten im Kontext der Problemstellung, wie er mit der Umwelt, d.h. den Problem- oder Testdaten, wechselwirkt. Als *Genotyp* wird dagegen die Kodierung des Lösungskandidaten innerhalb des evolutionären Algorithmus bezeichnet. Im Falle des Trainings eines neuronalen Netzwerkes mit n synaptischen Verbindungen und einer festgelegten Topologie ist der Zusammenhang zwischen Genotyp und Phänotyp einfach: Der Phänotyp ist das konfigurierte Netzwerk, während als Genotyp die Kodierung, d.h. die im Speicher abgelegten Netzwerkeinstellungen, bezeichnet wird. Diese Repräsentation kann durch einen Vektor $w_1 \cdots w_n$ erfolgen, in dem die Synapsengewichte z.B. als 16 Bit Integerwerte abgelegt sind, oder als Bitvektor $b_1 \cdots b_{16n}$, der die gleiche Information enthält. Beide Repräsentationen unterscheiden sich darin, wie die Variationsoperatoren Rekombination und Mutation (s.u.) wirken. Analog zur Biologie werden die Begriffe *Gen* für die kleinste Einheit an Erbinformation, *Chromosom* für eine zusammengehörige Gruppe an Genen und *Individuum* oder *Genom* für die gesamte Erbinformation eines Lösungskandidaten verwendet.

Evaluation Die Evaluation ist die Basis der Selektion, sie bestimmt die Güte oder Fitness der einzelnen Individuen aus der Population. Dabei wird meist ein möglichst hoher Fitnesswert angestrebt, obwohl die Suche nach Minima mathematisch äquivalent ist. Im Falle des überwachten Trainings neuronaler Netzwerke ist das Ziel der Evolution bereits bekannt: Das Netzwerk soll für jeden der n Eingabevektoren $\vec{I}_i$ der Trainingsdaten den zugehörigen Soll-Ergebnisvektor $\vec{S}_i$ liefern[2]. Die tatsächliche Ausgabe des Netzwerkes sei $\vec{A}_i$, wobei der Index $i = 1 \ldots n$ durch die Anzahl der verfügbaren Trainingsvektoren zählt. Die Fitnessfunktion

$$F = F(\vec{S}_1 \ldots \vec{S}_n, \vec{A}_1 \ldots \vec{A}_n) \tag{1.4}$$

sollte nicht nur die vollständig korrekte Ausgabe mit einem Maximalwert honorieren, sondern auch Teillösungen belohnen. Eine sehr einfache Fitnessfunktion beispielsweise vergleicht nur die Ergebnisdaten mit den Solldaten und vergibt für jeden übereinstimmenden Bitwert einen Punkt.

Als Vorstellungshilfe wird gerne eine Fitnesslandschaft oder Fitnessoberfläche verwendet: In einer dreidimensionalen Auftragung bezeichnen die Parameter x und y die Eigenschaften der Individuen und jedem Punkt dieser Fläche ist eine Höhe z, die Fitness, zugeordnet. Gute und einfach zu optimierende Fitnesslandschaften bestehen aus einem breiten Bergkegel, während eine stark zerklüftete Hochgebirgsregion die Gefahr birgt, in lokalen Minima stecken zu bleiben. Die geschickte Wahl einer günstigen Fitnessfunktion ist mit entscheidend für den Erfolg des evolutionären Trainings.

[2]Die Dimension der Ein- und Ausgabevektoren ist problemspezifisch und entspricht der Anzahl der Eingabe- bzw. Ausgabeneuronen des zugrunde liegenden Netzwerkes.

Werden neuronale Netzwerke trainiert, so kann die Evaluation als zweistufiger Vorgang angesehen werden: Im ersten Teil, der *Netzwerkberechnung*, arbeitet das Netzwerk. Es wird mit den genetischen Daten des zu testenden Individuums konfiguriert, dann werden sukzessive die Eingabevektoren $\vec{I}_1$ bis $\vec{I}_n$ präsentiert, die Netzwerkberechnung ausgeführt und die Ergebnisdaten $\vec{A}_1$ bis $\vec{A}_n$ zurück gelesen und gespeichert. Der zweite Teil der Evaluation, die eigentliche Fitnessberechnung, erfolgt außerhalb des Netzwerkes, indem die gespeicherten Ergebnisdaten $\vec{A}_i$ mit den Solldaten $\vec{S}_i$ verglichen werden und die Fitnessfunktion 1.4 ausgeführt wird.

Selektion Während Repräsentation und Evaluation stark abhängig von der gewählten Problemstellung sind, kann die Selektion, ausgehend von den ermittelten Fitnesswerten, allgemein formuliert werden. Die Konkurrenz innerhalb einer Population kann mit zwei Modellen ausgedrückt werden. Das *generational model* erneuert in jeder Iteration jeweils die gesamte Population, während das *steady-state model* jeweils nur einen Teil der Individuen ersetzt.

Die Selektion erfolgt üblicherweise stochastisch in einer Weise, die Individuen mit höherer Fitness bevorzugt. Die Wahrscheinlichkeit, dass ein Individuum sich als Elternteil durchsetzt bzw. die Umweltselektion überlebt, kann entweder proportional zur normalisierten Fitness, oder anhand einer Fitness-Rangfolge gewählt werden. In der *Turnierselektion* treten n zufällig ausgewählte Individuen gegeneinander an, nur das Individuum mit der höchsten Fitness wird selektiert. Je kleiner n ist, desto größer wird der stochastische Anteil der Selektion. Aufgrund der stochastischen Natur garantieren diese Selektionsmechanismen nicht, dass sich das Individuum mit der höchsten Fitness im Selektionsprozess durchsetzt. Wenn zusätzlich außerhalb des normalen Selektionsprozesses das Individuum mit der höchsten Fitness automatisch ausgewählt wird, spricht man von *Elitism*.

Variationsoperatoren Es werden zwei Klassen von Variationsoperatoren unterschieden: Die *Mutation* verwendet nur jeweils ein Elternindividuum und erzeugt einen Nachkommen, während die *Rekombination* die genetische Information von zwei oder mehr Eltern kombiniert, um ein oder mehrere Nachkommen zu erzeugen.

Mutationsoperator Die Mutation bewirkt eine zufällige Veränderung der Erbinformation, womit auch neue, bisher noch nicht aufgetretene Gene entstehen können. Im Allgemeinen wird für jedes Gen separat entschieden, ob es mutiert werden soll oder nicht. Dadurch wird die Gesamtzahl an Mutationen nur von der Verteilung der gezogenen Zufallszahlen abhängig. Die Mutation ist abhängig von der Repräsentation: Bei binärer Kodierung der Gene bedeutet eine Mutation, dass ein *Bitflip* ausgeführt wird, d.h. dass das Bit von 0 auf 1 bzw. von 1 auf 0 geändert wird. Bei einer Kodierung der Erbinformation mit Integerzahlen gibt es mehrere verschiedene Möglichkeiten, die Mutation durchzuführen: Die *gleichverteilte Mutation* ersetzt den bisherigen Wert durch einen neuen, zufällig erzeugten Wert. Sie entspricht dem Bitflip der bitweisen Kodierung, da der alte Wert verworfen wird. Um allerdings graduelle Änderungen zu ermöglichen, wird die *Creep Mutation* verwendet. Im Falle einer Mutation wird zum bisherigen Gen ein kleiner (positiver oder negativer) Wert hinzu addiert. Dieser neue Wert wird aus einer um Null zentrierten Verteilung ausgewählt, sodass meist, aber nicht immer, kleine Werte Verwendung finden. Wird die Gaußverteilung verwendet, so spricht man von *gaußscher Mutation*.

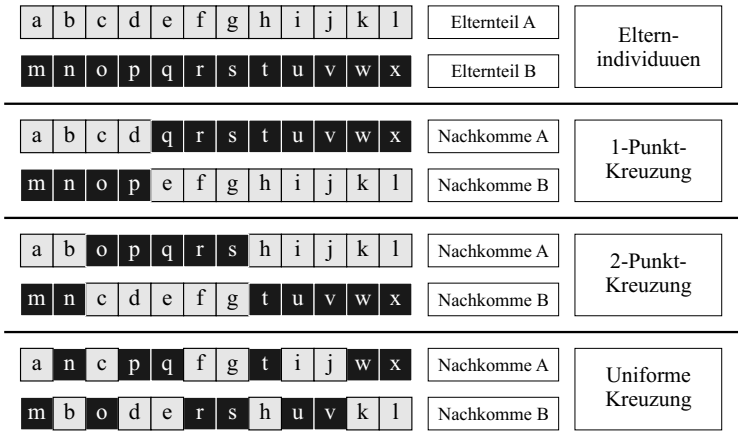

Abbildung 1.7: Der n-Punkt Kreuzungsoperator teilt die Elternchromosomen in (n+1) Teile und verteilt diese Teile auf den Nachkommen, hier ein Beispiel für $n = 1$ und $n = 2$. Der uniforme Kreuzungsoperator entscheidet für jedes Gen separat, von welchem Elternteil es genommen wird.

Durch Eigenschaft der Gaußverteilung, dass etwa $\frac{2}{3}$ der Werte im Bereich von einer Standardabweichung σ um den Nullpunkt liegen, lassen sich die Auswirkungen der Mutation mit dem Faktor σ skalieren.

Rekombinations- oder Kreuzungsoperator Die am häufigsten angewendete Form des Kreuzungsoperators erzeugt aus zwei Elternindividuen zwei neue Nachkommen. Die Implementation ist wie beim Mutationsoperator von der Repräsentation abhängig. Im Falle der binären Kodierung stellt ein Bit, bei Integerkodierung eine Integerzahl ein Gen dar. Der Kreuzungsoperator arbeitet auf einer geordneten Menge von Genen. Die verschiedenen Kreuzungsoperatoren sind in Abb. 1.7 dargestellt. Die Kreuzungspunkte werden dabei zufällig ausgewählt und die genetische Information der Eltern auf beide Nachkommen aufgeteilt. Eine Erweiterung der 1-Punkt-Kreuzung ist die *N-Punkt-Kreuzung*, die für $N = 2$ dargestellt ist. Dabei werden N Kreuzungspunkte ausgewählt und $(N + 1)$ Chromosomenstücke ausgetauscht. Bei *Uniformer Kreuzung* wird für jedes Gen separat entschieden, ob es vom ersten oder vom zweiten Elternteil übernommen wird. 1-Punkt- und N-Punkt-Kreuzung trennen genetische Information, die an benachbarten Stellen gespeichert sind, weniger häufig als die uniforme Kreuzung. Im Gegensatz dazu werden allerdings die Gensequenzen an den beiden Rändern des Chromosomes durch 1-Punkt-Kreuzung mit hoher Wahrscheinlichkeit getrennt, gleiches gilt für N-Punkt-Kreuzung für ungerade N.

Evolutionäres Training des HAGEN Chips Die Repräsentation des künstlichen neuronalen Netzwerkes im Sinne des evolutionären Algorithmus sieht folgendermaßen aus. Ein Synapsengewicht des HAGEN Chips hat eine Genauigkeit von 10 Bit, zusätzlich wird durch das Vorzeichen entschieden, ob es sich um eine hemmende oder eine erregende Synapse handelt. Ein Synapsengewicht des HAGEN Chips wird im Folgenden als ein *Gen*

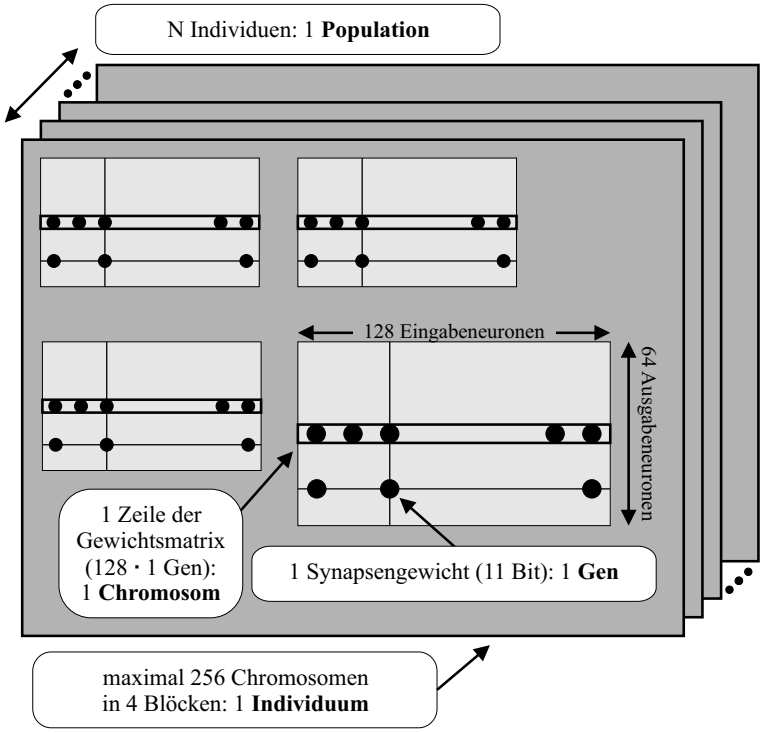

Abbildung 1.8: Definition der Repräsentation bezogen auf den Netzwerkchip HAGEN

bezeichnet (vgl. Abb. 1.8) und beansprucht 11 Bit Speicherplatz. Alle 128 Synapsengewichte einer Zeile innerhalb eines Blockes bilden ein *Chromosom*. Ein *Individuum* oder Genom enthält maximal die Information, die zur Konfiguration aller 4 Blöcke notwendig ist, d.h. bis zu 32768 Gene bzw. bis zu 256 Chromosomen. Eine Population schließlich enthält eine Anzahl N an Individuen, wobei N vom Trainingsalgorithmus gewählt wird.

Der Kreuzungsoperator arbeitet auf einzelnen Genen, d.h. auf Integerzahlen mit der Genauigkeit von 11 Bit. Es wird für jedes Gen separat entschieden, ob eine Mutation auftritt und entsprechend der Art der Mutation der Wert dieses Gens verändert. Gleichverteilte Mutation verwirft den bisherigen Wert und schreibt ein neues, zufälliges Synapsengewicht. Die Gaußsche Mutation dagegen addiert einen zufälligen Wert, der einer gaußschen Verteilung entnommen wird, auf den bisherigen Wert auf. Wird dabei der zulässige Maximalwert überschritten, so wird das Gen auf den Maximalwert gesetzt. Für Minimalwerte wird entsprechend verfahren.

Die Rekombination wird auf der Basis von Chromosomen ausgeführt, der Kreuzungsoperator verknüpft jeweils Chromosomen von jedem Elternteil miteinander. Es kommen sowohl 1-Punkt-Kreuzung als auch 2-Punkt-Kreuzung zum Einsatz.

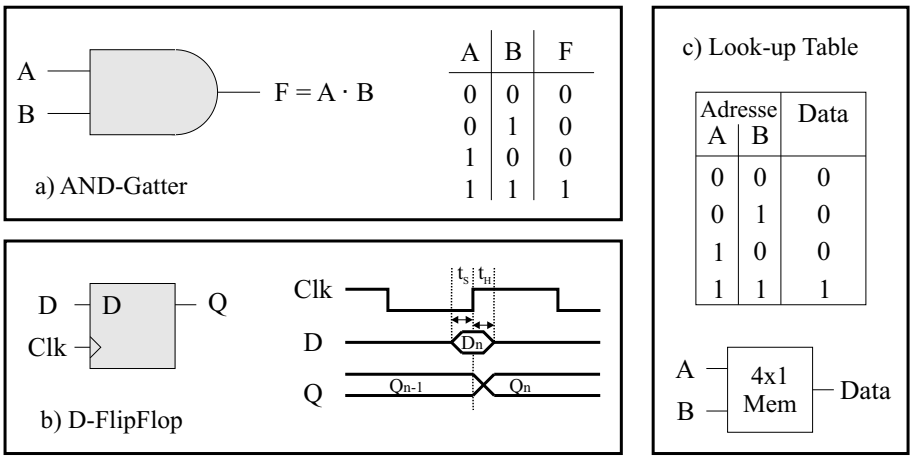

Abbildung 1.9: (a) Schaltsymbol und Wahrheitstabelle für ein AND-Gatter
(b) Symbol und Zeitverhalten eines D-Flip-Flop-Speicherelements. Die *Setup-Zeit* t_s und die *Hold-Zeit* t_h legen fest, wie lange vor bzw. nach der Taktflanke der Eingang stabil und gültig sein muss.
(c) Funktionsweise einer 4x1 Speicherzelle als Look-up Table

1.3 Technologien und Werkzeuge

In diesem Abschnitt werden die notwendigen Technologien und Werkzeuge beschrieben, die verwendet werden, um die Experimentalplattform zu entwickeln und zu betreiben. Begonnen wird mit einer kurzen Einführung über digitale Logik, daraufhin wird das Prinzip der programmierbaren Logik vorgestellt. Die für die Arbeit wichtigen Eigenschaften und Fähigkeiten der verwendeten FPGA-Bausteine, vor allem des Virtex-II Pro, werden zusammengefasst. Schließlich werden die Eigenschaften des im FPGA eingebetteten PowerPC-Prozessors und die Eigenschaften von statischen und dynamischen Speicherbausteinen zusammengefasst.

1.3.1 Digitale Logik

Die kleinste Einheit einer digitalen Schaltung ist ein *Gatter*, es besteht aus einem oder mehreren Eingängen und einem einzigen Ausgang. Im Falle von digitaler Logik können die Ein- und Ausgänge nur einen von zwei Zuständen {`0`, `1`} annehmen. Die Definition der Eigenschaften eines Gatters kann über eine Wahrheitstabelle erfolgen, Abb. 1.9a zeigt Schaltsymbol und Wahrheitstabelle für ein AND-Gatter. Digitale Gatter und logische Funktionen können mit den Methoden der *Boolschen Algebra* beschrieben werden, unter anderem kann gezeigt werden, dass sich jede Boolsche Funktion nur mit Hilfe von NAND- (oder NOR-) Gattern ausdrücken lässt [12].

Wird der Ausgang eines Gatters direkt oder auf einem Umweg über andere Gatter wieder auf den Eingang desselben Gatters geleitet, so spricht man von einer rückgekoppelten Schaltungen. Rückgekoppelte Schaltungen können Zustände speichern, ein Beispiel eines solchen Speicherelements ist ein *D-Flip-Flop* (FF) (vgl. Abb 1.9b). Das D-Flip-Flop ist flankengesteuert, nur während des Wechsels des Taktsignals von `0` auf `1`, d.h. während

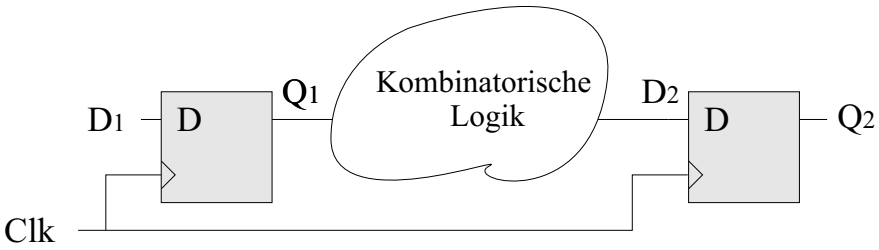

Abbildung 1.10: Taktsynchrone Schaltungen bestehen aus getakteten Speicherelementen, die durch kombinatorische Logik miteinander verknüpft sind.

der steigenden Taktflanke, muss der Eingang `D` gültig sein. Der Ausgang `Q` ändert sich nur synchron zum Takt `Clk` und bleibt bis zur nächsten Taktflanke erhalten. Die Zeitdauer t_s, die das Datensignal vor der Taktflanke stabil sein muss, wird als *Setup-Zeit*, die Zeitdauer t_h, die das Datensignal nach der Taktflanke noch gehalten werden muss, wird als *Hold-Zeit* bezeichnet.

Mittels *Look-up Tables* (LUT) können Speicherelemente eingesetzt werden, um logische Funktionen auszudrücken. Ein Beispiel ist in Abb. 1.9 gegeben. Eine 4x1 Speicherzelle verfügt über 2 Adresseingänge und speichert dementsprechend 4 Werte. Werden die Speicherwerte gemäß der Abbildung initialisiert, so stellt der Ausgang eine Und-Verknüpfung der beiden Adresseingänge dar. Ein 16x1 LUT-Speicherelement kann dementsprechend eine beliebige boolsche Funktion mit bis zu vier Eingabewerten ausdrücken.

Taktsynchrone Schaltwerke sind aus flankengesteuerten Speicherelementen aufgebaut, die durch kombinatorische Logik in Form von Gattern ohne Rückkopplungen verbunden sind (vgl. Abb. 1.10).

1.3.2 Programmierbare Logik

Field Programmable Gate Arrays (FPGAs) sind frei programmierbare Logikbausteine. Sie sind nicht anwendungsspezifisch und werden in großen Stückzahlen hergestellt. FPGAs bestehen im Wesentlichen aus konfigurierbaren Logik-Zellen, die durch ebenfalls programmierbare Verbindungsleitungen miteinander verknüpft werden. Die Logik-Zellen sind in einer Matrix-Struktur angeordnet. Jede Zelle verfügt einerseits über Speicherelemente, die als D-Flip-Flops konfiguriert werden können, als auch über die Möglichkeit, kombinatorische Logik mittels LUTs zu realisieren. Über ebenfalls programmierbare Verbindungsleitungen können die einzelnen Logik-Zellen miteinander verbunden und so komplexe Schaltungen und Schaltwerke aufgebaut werden. Im Folgenden wird am Beispiel des Virtex-II Pro-FPGAs der Firma Xilinx [86] die Struktur programmierbarer Logikbausteine beschrieben.

Elementares Logik-Element Das elementare Logik-Element (*Slice*) enthält zwei Speicherzellen, die i.A. als D-Flip-Flops konfiguriert werden und zwei Funktionsgeneratoren bzw. LUTs. Die LUTs des FPGAs besitzen 4 Steuereingänge und damit 16 Speicherstellen. Jede LUT kann also entweder eine beliebige boolsche Funktion mit bis zu

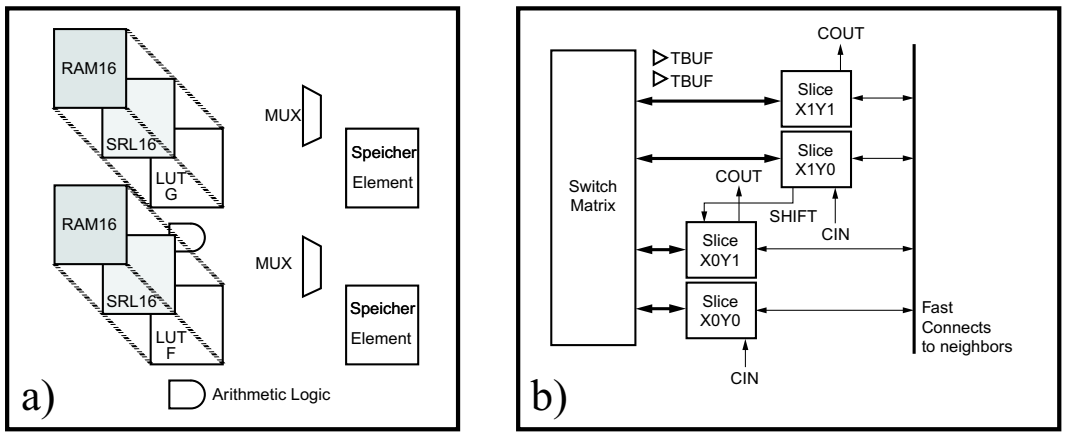

Abbildung 1.11: (a) Übersicht über ein Logik-Element (*Slice*), die 2 Funktionsgeneratoren können als LUT, als Schieberegister (*SRL*) oder als 16 Bit Speicherzelle konfiguriert werden. Die 2 Speicherelemente werden i.A. als D-Flip-Flops verwendet [86].
(b) Vier Logik-Elemente bilden einen CLB, der mit benachbarten CLBs und mit der globalen Routing-Matrix verbunden werden kann [86].

vier Eingabewerten ausdrücken oder als 16-Bit Speicherelement bzw. als 16-Bit Schieberegister verwendet werden. Abb. 1.11a zeigt eine Übersicht über eine Logik-Zelle, neben den bereits beschriebenen 2 Speicherzellen und 2 Funktionsgeneratoren enthält jede Zelle schnelle vertikale Signalleitungen (*Carry-Logic*), arithmetische Logik und zwei zusätzliche Multiplexer[3].

Vernetzung Jeweils 4 Logik-Elemente bilden einen *Configurable Logic Block* (CLB). Es stehen sowohl lokale Routing-Ressourcen innerhalb eines CLBs als auch Verbindungen zu den benachbarten CLBs zur Verfügung. Zusätzlich ist jeder CLB über eine Verbindungseinheit (*Switching Matrix*) mit der globalen Routing-Matrix verbunden und kann so die chipweiten vertikalen und horizontalen Verbindungsleitungen verwenden.

Interne Speicher Um kleine bis mittlere Datenmengen innerhalb des FPGAs zwischenzuspeichern, bietet der Virtex-II Pro zwei Möglichkeiten: Einerseits können die LUTs in den Logik-Elementen jeweils als 16-Bit Speicherelemente verwendet werden. Diese Speichermethode wird als *Distributed Ram* bezeichnet. Andererseits stellt der FPGA spezielle Blockspeicher zur Verfügung. Diese Blockspeicher, im Folgenden *Blockrams* genannt, haben je eine Größe von 18 kBit bei einer maximalen Datenbreite von 36 Bit und sind in 4 senkrechten Spalten über den FPGA verteilt. Jedes Blockram verfügt über zwei voneinander unabhängige Anschlüsse, es können also zwei Prozesse gleichzeitig jeden Speicherblock nutzen.

[3]Ein 2-Bit Multiplexer ist eine Datenweiche mit 2 Daten-, einem Steuereingang und einem Datenausgang. Abhängig vom Steuereingang wird genau einer der beiden Dateneingänge auf den Ausgang gelegt.

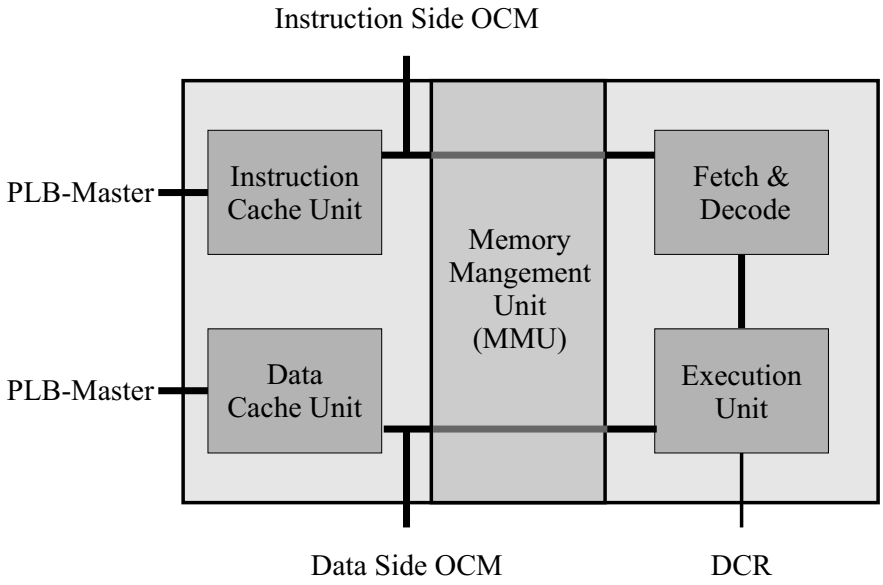

Abbildung 1.12: Übersicht über den PowerPC Prozessorkern mit den Schnittstellen zur programmierbaren Logik PLB, OCM und DCR

Takterzeugung und Taktverteilung Der FPGA ist auf die Versorgung mit einem externen Takt angewiesen, stellt allerdings mit den *Digitalen Clock Managern* (DCM) Bausteine zur Manipulation von Taktsignalen zur Verfügung. Jeder DCM erwartet einen Eingabetakt, der entweder von einem externen Oszillator oder auch von einem anderen DCM stammen kann, und kann daraus verschiedene Ausgabetakte, z.B. mit doppelter oder halber Frequenz, generieren. Für die Synchronisation von Datenübertragungen zwischen dem FPGA und externen Bausteinen ist eine weitere Eigenschaft der DCMs sehr wichtig: Gesteuert über eine einfache Schnittstelle kann die Phase der Ausgabetakte eines DCMs während des Betriebs verschoben werden, die Auflösung ist dabei $\frac{1}{256}$ der Taktperiode.

Eingebetteter PowerPC

Der verwendete FPGA verfügt über einen integrierten Mikroprozessor. Im Folgenden wird ein Überblick über die Eigenschaften dieses PowerPCs gegeben und die Kommunikationskanäle zwischen dem Prozessor und der programmierbaren Logik beschrieben, die Details finden sich sich in [84].

Der eingebettete PowerPC ist eine 32-bit Harvard Architektur [53], für den Virtex-II Pro implementiert in 0.13 μm Technologie und kann mit Frequenzen bis 350 MHz betrieben werden. Der Prozessorkern enthält neben den Einheiten zum Laden, Dekodieren und Ausführen der Befehle eine *Memory Management Unit* (MMU) und jeweils 16 kByte Daten- und Instruktionscache. Beide Caches sind 2-fach assoziativ [26]. Zur Kommunikation mit der umgebenden FPGA-Logik stehen die folgenden drei Möglichkeiten zur Verfügung (vgl. Abb. 1.12).

On-Chip Memory (OCM) Die OCM-Controller verbinden den Prozessorkern direkt mit Blockrams, die sich im Hauptteil des FPGAs befinden. Es gibt zwei unabhängige OCM-Controller: der DS-OCM kann Daten in 32 Bit Breite schreiben und lesen, der IS-OCM liest Instruktionen in 64 Bit Breite. Da jedes Blockram zwei voneinander unabhängige Anschlüsse unterstützt, kann der jeweils zweite Anschluss mit der programmierbaren Logik verbunden werden. Auf diese Weise wird über den DS-OCM eine bidirektionale Datenkommunikation zwischen dem Prozessorkern und der programmierbaren Logik aufgebaut. Zugriffe auf den OCM werden nicht über den Cache abgewickelt.

Processor Local Bus (PLB) Der Prozessorkern ist kompatibel mit der *CoreConnect* Bus Architektur [17]. Sowohl der Datencache als auch der Instruktionscache besitzen eine eigene Schnittstelle zur programmierbaren Logik über den PLB, beide fungieren als PLB-Master. Der PLB-Standard ist in [14] dokumentiert.

Device Control Register (DCR) Das Device Control Register bietet einen Adressraum von 10 Bit für Komponenten außerhalb des Prozessorkerns. Über das DCR können Daten in 32 Bit breite mit der programmierbaren Logik ausgetauscht werden. Um verschiedene Komponenten an das DCR anzuschließen, werden die Datenleitungen in einer Kette verschaltet, die einzelnen Komponenten werden dann über die 10 Bit Adresse selektiert. Über das DCR kann der Prozessorkern mit langsamer Rate einzelne Register von unterschiedlichen Komponenten der programmierbaren Logik lesen und schreiben.

Serielle Multi-Gigabit-Transceiver (MGT)

Zur schnellen Datenübertragung sind serielle Multi-Gigabit-Transceiver in die frei konfigurierbare Logik des FPGAs eingebettet. Jeder MGT kann Daten seriell mit bis zu 3,125 Gbit/s übertragen.

1.3.3 Schaltungsentwurf von FPGAs

Bevor ein FPGA verwendet werden kann, muss die interne Logik, d.h. die Eigenschaften der Funktionsgeneratoren, die Aufgaben der Register und die interne Vernetzung zwischen allen Komponenten programmiert werden. Technisch geschieht dies, indem ein Bitstream, der diese Informationen enthält in den FPGA geladen wird. Um diesen Bitstream zu erhalten, muss zuerst in der *Designeingabe* die gewünschte Applikation in einer geeigneten Hochsprache oder mittels eines Schaltplans beschrieben werden. In der *Synthese* wird aus dieser Beschreibung eine Netzliste für die Zielarchitektur erzeugt. Schließlich folgt das *Place & Route*, die Platzierung und Verdrahtung spezifisch für den gewählten FPGA-Typ. Dieser Weg von der Designeingabe bis zum vollständigen Bitstream ist in Abb. 1.13 dargestellt.

Designeingabe Für komplexere Entwürfe wird eine visuelle Designeingabe mittels eines Schaltplans schnell unübersichtlich. Für die textuelle Beschreibung haben sich seit Ende der 80er Jahre die beiden *Hardwarebeschreibungssprachen* VHDL und Verilog durchgesetzt. Die Beschreibung kann auf verschiedenen Entwurfsebenen stattfinden. Auf der

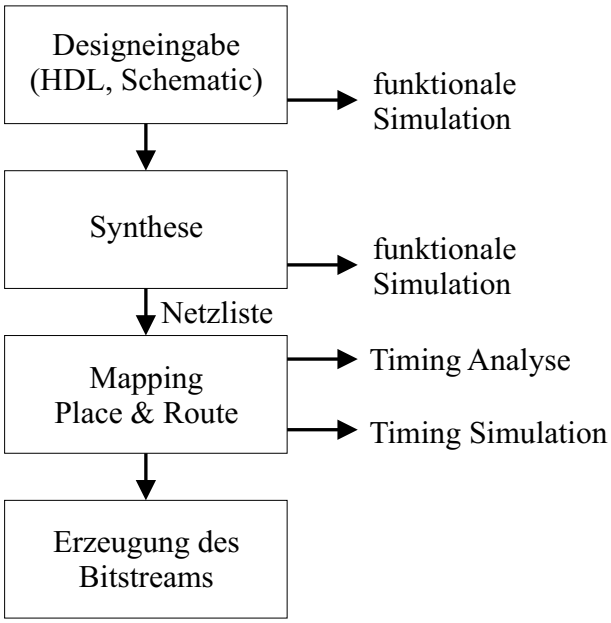

Abbildung 1.13: Ablauf eines FPGA Schaltungsentwurfs von der Designeingabe bis zur Erzeugung des Bitstreams, mit dem der FPGA konfiguriert werden kann. Auf allen Ebenen ist eine Simulation der Schaltung möglich.

Logik-Ebene werden die durch eine Bibliothek bereitgestellten Grundelemente (wie beispielsweise Gatter oder D-Flip-Flops) zusammengeschaltet. Die *Register-Transfer-Ebene* beschreibt Elemente wie Addierer oder Multiplexer und den Transfer der Daten über Signale. Die *Algorithmische Ebene* schließlich beschreibt größer Module und Algorithmen. Während die Beschreibung auf der Logik-Ebene und der Register-Transfer-Ebene im Allgemeinen synthetisierbar sind, sich also in eine Netzliste und damit eine reale Schaltung überführen lassen, wird die Verhaltensbeschreibung auf algorithmischer Ebene hauptsächlich zur Simulation und zur Definition von Prüfstrukturen verwendet. Eine Einführung in VHDL ist in [38], eine stärker an der Praxis orientierte Beschreibung in [13] zu finden.

Simulation Mittels Simulator-Programmen wie beispielsweise ModelSim [46] kann die VHDL-Beschreibung der synthetisierbaren Schaltung zusammen mit einer umgebenden Testbench simuliert und so mit Hilfe geeigneter Testvektoren auf logische Korrektheit überprüft werden. Der Zeitverlauf der verwendeten Signale und Variablen kann dabei grafisch angezeigt werden.

Synthese Die *Synthese* besteht aus zwei Schritten. Zuerst wird die VHDL-Schaltungsbeschreibung auf herstellerunabhängige Komponenten wie Register, Addierer und Multiplexer abgebildet und eine von der Zielarchitektur unabhängige Netzliste erzeugt. In einem zweiten Schritt werden dann Eigenschaften der Zielarchitektur, im Falle des FPGAs beispielsweise die Besonderheiten der Register oder die Größe der Funktionsgeneratoren, berücksichtigt. Es stehen verschiedene Anbieter von Synthesewerkzeugen zur

Verfügung, unter anderem bietet Xilinx mit der ISE [88] ein Programmpaket für den gesamten Schaltungsentwurf an. Um die funktionale Korrektheit der Netzliste zu überprüfen, kann sie ähnlich der Beschreibung in VHDL simuliert werden.

Mapping Die weiteren Schritte nach der Generation der Netzliste werden von Werkzeugen durchgeführt, die Abhängig von der Zieltechnologie sind und von den FPGA-Herstellern zur Verfügung gestellte werden. Das *Mapping* führt eine Optimierung der vollständigen Schaltung durch und bildet die optimierte Netzliste komplett auf die Zieltechnologie ab. Die Optimierung entfernt einerseits redundante Logik, andererseits können auch Elemente vervielfacht werden, um bessere Routingeigenschaften zu erreichen. Nach dem Mapping steht fest, wie viele Ressourcen wie beispielsweise I/O-Pins, I/O-Zellen, Register, Funktionsgeneratoren oder Blockrams die Schaltung benötigt.

Place & Route Die Verteilung der durch das Mapping erzeugten Elemente auf die physikalisch vorhandenen Logik-Blöcke bzw. I/O-Zellen und ihre Verdrahtung wird im *Place & Route*-Schritt durchgeführt. Dabei können neben der eigentlichen Schaltung weitere Nebenbedingungen (*Constraints*) angegeben werden. Einerseits müssen über *Location Constraints* die physikalischen Pins des verwendeten Bausteins gemäß der vorliegenden Leiterplatte angegeben werden, andererseits können auch Elemente innerhalb des FPGAs fest an einen physikalischen Ort oder in einem Bereich des FPGAs platziert werden.

Nachdem alle Elemente platziert worden sind, werden die Verbindungen zwischen ihnen geschaltet. Erst nach diesem *Routing* steht fest, wie viel Laufzeit die einzelnen Logikpfade benötigen. Dabei setzt sich diese Laufzeit aus den Verzögerungszeiten der Logikelemente zuzüglich der Signallaufzeit der Verbindungen zusammen. Die *Timing Constraints*, die der Router erfüllen muss, ergeben sich aus der maximalen Taktperiode. Um die Verdrahtung zu vereinfachen, können für spezielle Pfade, wie beispielsweise die Reset-Logik, auch längere Signallaufzeiten erlaubt werden. Dies muss aber in der Designeingabe berücksichtigt werden. Das vollständige Design einschließlich der Logik- und der Verbindungslaufzeiten kann in einer *Timing Simulation* auf Korrektheit überprüft werden. Das manuelle Eingreifen sowohl in die Platzierung als auch in die Konfiguration der Verbindungen ist jederzeit möglich und wird als *Floorplanning* bezeichnet. Nachdem das Design komplett platziert und die Verdrahtung spezifiziert ist, kann eine Konfigurationsdatei erzeugt werden, die den Bitstream enthält, der in den FPGA geladen wird.

1.3.4 Speicherbausteine

Der vorgestellte FPGA verfügt zwar über interne Speicher, diese sind aber in ihrer Größe begrenzt. Werden die Funktionsgeneratoren der Logik-Elemente verstärkt als *Distributed Ram* eingesetzt, so steht weniger Logik zur Realisierung der Schaltungen zur Verfügung. Der gesamte Blockram-Speicher des verwendeten Virtex-II Pro beträgt 88 kByte und würde gerade zur Speicherung von 2 Individuen (vgl. Abschnitt 1.2) des HAGEN Chips ausreichen. Aus diesem Grund müssen externe Speicherbausteine eingesetzt werden. Es wird unterschieden zwischen statischem und dynamischem Speicher (SRAM bzw. DRAM).

SRAM-Speicherzellen bestehen aus Flip-Flops und erhalten ihre gespeicherte Information, solange die Betriebsspannung aktiv ist. DRAMs dagegen speichern die Information

als elektrische Ladung auf einer Kapazität. Dies hat den Vorteil, dass deutlich weniger Chipfläche pro Speicherzelle benötigt wird, allerdings verlieren die Kapazitäten durch Leckströme mit der Zeit ihre Ladung und müssen in Abständen von einigen Millisekunden aufgefrischt, d.h. gelesen und wieder geschrieben werden. *Synchrones DRAM* (SDRAM) bezeichnet Dynamisches RAM, das über eine synchrone Schnittstelle angesprochen wird.

Auch bezüglich der Ansteuerung unterscheiden sich die beiden Speichertechnologien. Bei SRAM können die Speicherzellen in einer beliebigen Reihenfolge oder auch zufällig adressiert werden. SDRAM dagegen verwendet ein zweistufiges Adressierungsschema: Bei SDRAMs ist der verfügbare Speicherbereich in Zeilen und Spalten aufgeteilt. Um eine Speicherzelle anzusprechen, muss erst die Zeile adressiert werden. Diese Zeile wird dann aktiviert, d.h. die Inhalte aller Speicherzellen dieser Zeile werden im Speicherbaustein intern zwischengespeichert. Erst nachdem diese Aktivierung abgeschlossen ist, können die einzelnen Spalten in dieser Zeile adressiert werden.

Um größere Flexibilität zu gewähren, bestehen SDRAM-Module aus insgesamt 4 oder 8 Speicherbänken, wobei pro Bank jeweils eine Zeile aktiv sein kann. Bei einer Gesamtgröße des Speichermoduls von 512 MByte und einer Organisation in 8 Bänken enthält eine Bank folglich 64 MByte, es gibt 8192 Zeilen, jede Zeile enthält 1024 Speicherzellen von jeweils 8 Byte Größe.

Für die Ansteuerung hat diese interne Organisation zwei Konsequenzen: Aufeinander folgende Zugriffe, die Daten aus einer Bank, aber von unterschiedlichen Zeilen adressieren, können nur sehr langsam bearbeitet werden. Bei jedem Zeilenwechsel muss die nicht mehr benötigte Zeile deaktiviert und die neue Zeile aktiviert werden, ein Vorgang, der 7-10 Takte beansprucht. Erst dann kann der nächste Zugriff ausgeführt werden. Als zweite Konsequenz des zweistufigen Adressierungsschemas kann bei Verwendung von SDRAM keine feste Latenz oder Antwortzeit garantiert werden: Da vor einem Zugriff i.A. zuerst eine fremde Zeile deaktiviert werden muss und zusätzlich eventuell Auffrischzyklen durchgeführt werden müssen, kann nicht garantiert werden, wann ein spezieller Zugriff wirklich an das Speichermodul weitergeleitet wird, bzw. wann die Daten von Lesezugriffen verfügbar sind.

Durch das zweistufige Adressierungsschema und die Notwendigkeit, in regelmäßigen Abständen Auffrischzyklen durchzuführen, ist die Ansteuerung von SDRAM deutlich komplexer als die Ansteuerung von SRAM.

Double Data Rate (DDR) SDRAM überträgt auf der Schnittstelle die Daten mit doppelter Datenrate. Intern wird in jedem Taktzyklus ein einzelner 128 Bit breiter Zugriff durchgeführt, auf der physikalischen Schnittstelle werden allerdings die Daten in zwei 64 Bit breiten Teilen pro Taktzyklus übertragen, d.h. die Daten wechseln sowohl mit der steigenden als auch mit der fallenden Taktflanke.

1.3.5 Linux-Kernel

Die Aufgabe eines Betriebssystems, speziell des Linux-Kernels, ist einerseits die Verwaltung der Hardwareressourcen, andererseits die Bereitstellung einer Umgebung, in der die Anwendungen und Programme der Benutzer ausgeführt werden. Dabei können die Zuständigkeiten des Linux-Kernels in verschiedene Bereiche unterteilt werden, Abb. 1.14 zeigt eine vereinfachte Übersicht: Das *Prozess Management* regelt die Verteilung unterschiedlicher Prozesse auf den physikalisch vorhandenen Prozessor, während das *Speicher*

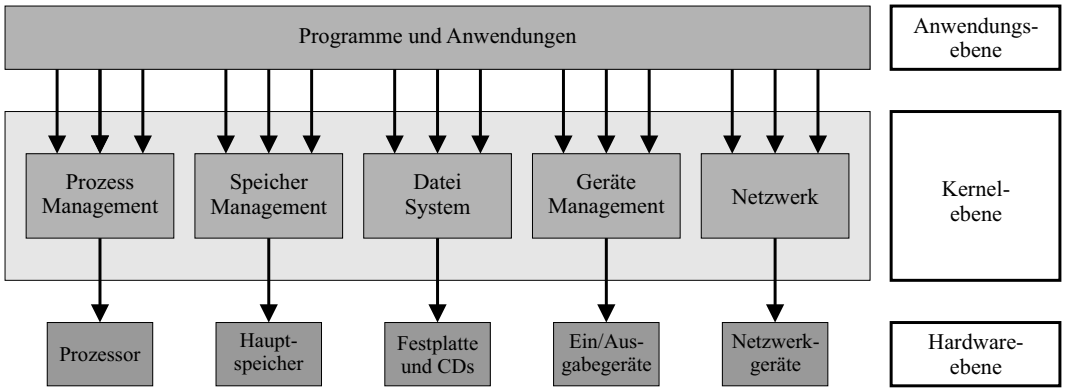

Abbildung 1.14: Vereinfachte Darstellung der Aufgaben des Linux-Kernels

Management den vorhandenen Hauptspeicher verwaltet. Das *Dateisystem* ist auch in einem eingebetteten System von großer Bedeutung, obwohl dort physikalisch keine optischen oder magnetischen Speichermedien vorhanden sind, da Linux Zugriffe auf externe Hardware wie Dateizugriffe verwaltet. Gerade für den eingebetteten PowerPC interessant ist die Behandlung von *Netzwerkschnittstellen* und der Zugriff auf externe *Geräte*.

Wollen Benutzerprogramme eine Hardwareressource verwenden, so können sie das nur über Funktionen, die der Kernel bereitstellt. Um diese Aufteilung zu unterstützen, erlauben moderne Prozessoren zwei unterschiedlich Betriebsmodi: Im *Privilegierten Modus* gelten keinerlei Einschränkungen, es können daher alle Funktionen des Prozessors verwendet werden. Im *Nicht-Privilegierten Modus* sind dagegen Funktionen, die Auswirkungen auf andere Benutzer haben und I/O-Funktionen nicht ausführbar. Das Betriebssystem Linux kennt daher ebenfalls zwei Modi: Der *Kernel Modus* entspricht dem privilegierten Modus, die Programme der Benutzer werden dagegen im nicht-Privilegierten, d.h. Benutzer-Modus ausgeführt.

Für den eingebetteten PowerPC von Bedeutung ist der Zugriff auf I/O-Funktionen. Die Zugriffe auf den DCR und auf selbst gewählte physikalische Adressen im Adressraum des PLB wie auch die Bearbeitung von Interrupts ist nur im privilegierten Modus des PowerPC, d.h. im Kernel-Modus möglich. Um also die volle I/O-Funktionalität des PowerPC ausnutzen zu können, müssen Kernel-Treiber entwickelt werden, die die Zugriffe im Kernel-Modus ausführen, und für Benutzerprogramme eine Schnittstelle im Benutzermodus bereitstellen.

1.4 Hardwaresysteme

Um den HAGEN Chip und auch zukünftige neuronale Netzwerkchips zu verwenden und anzusteuern, wurden von der Electronic Vision(s) Group am Kirchhoff-Institut für Physik zwei Hardwaresysteme entwickelt: Die erste Generation, das *Darkwing-System*, verwendet eine PCI-Karte mit FPGA und ist auf einen Standard-PC als Host-Computer angewiesen. Die zweite Generation, das *Nathan-System*, verwendet dagegen einen FPGA mit integriertem Mikroprozessor. Bis zu 16 Nathan Netzwerkmodule und damit bis zu 16 Netzwerkchips

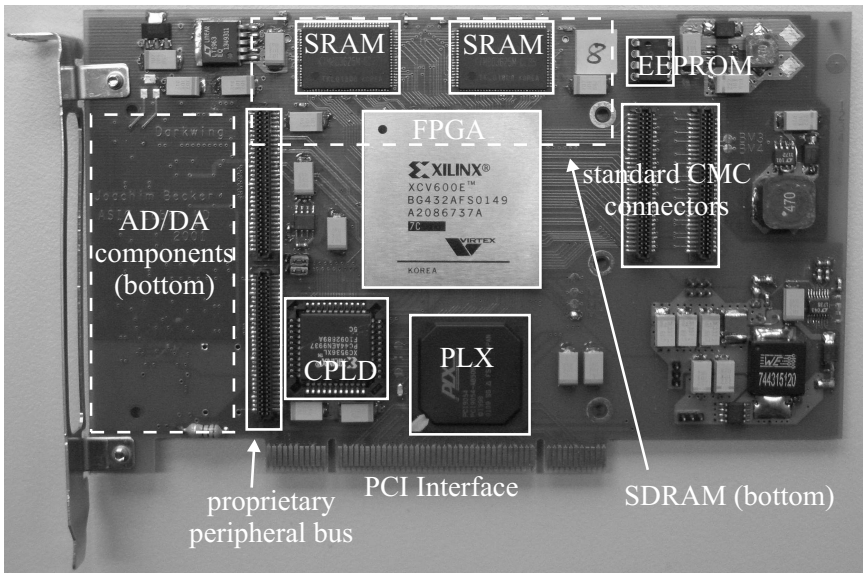

Abbildung 1.15: Fotografie der PCI-Karte Darkwing [69]

können verteilt und parallel in einer Backplane betrieben werden. In diesem Abschnitt wird die Hardware, die den beiden Systemen zugrunde liegt und jeweils im Rahmen von Diplomarbeiten [5] [25] entstanden ist, beschrieben.

1.4.1 Erste Generation: Darkwing-System

Abb. 1.15 zeigt eine Fotografie der Darkwing-Karte, die über den PCI-Bus in einem Standard-PC betrieben wird. Neben dem zentralen FPGA enthält die Darkwing-Karte Bausteine, um die Kommunikation über den PCI-Bus mit dem Host-PC abzuwickeln, Speicher in Form von SRAM und SDRAM und analoge Bauteile wie DAC und ADC (*Analog-Digital-Converter*). Über eine proprietäre periphere Schnittstelle können Tochterplatinen eingesetzt werden und so der Netzwerkchip HAGEN oder auch andere Chips angesteuert werden. Ausführliche Dokumentation zur Hardware des Darkwing-System findet sich in [5] und [69].

FPGA und PCI-Schnittstelle

Auf der Darkwing-Karte können verschiedene FPGA-Bausteine der VirtexE-Serie [81] der Firma Xilinx eingesetzt werden (XCV300E, XCV400E oder XCV600E). Dabei bietet der XCV600E mit 15552 FFs und 15552 LUTs die größten Logikressourcen. Die Schnittstelle zum PCI-Bus wird durch den PLX PCI 9054 [33] Baustein versorgt. Um höchste Flexibilität zu gewähren, wird die Konfiguration des FPGAs, d.h. die Programmierung der internen Logik, durch den Host-PC vorgenommen. Vor und während dieser Konfiguration kontrolliert auf der Darkwing-Karte ein nicht-flüchtiger *Complex Programmable Logic Device* (CPLD) [87] die Schnittstelle zum PLX Baustein. Nach erfolgter Konfiguration

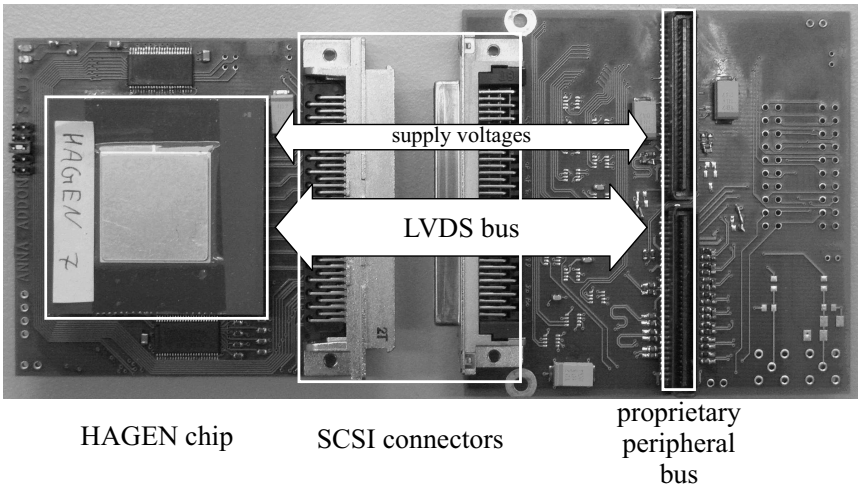

Abbildung 1.16: Fotografie der Tochterplatinen, die zum Betrieb des HAGEN Chip mit der PCI-Karte Darkwing notwendig sind [69]

übernimmt der FPGA alle Steuerungsaufgaben: Zum PLX und damit zum Host-PC, zum Speicher, zu den peripheren Schnittstellen und zu den analogen Bauteilen.

Speicher

Es stehen zwei externe Speicherbausteine zur Verfügung, die sich jedoch Daten- und Adressleitungen zum FPGA teilen und daher nicht gleichzeitig verwendet werden können. Zwei SRAM Bausteine mit jeweils 1 MByte Kapazität sind fest auf der Leiterplatte verlötet, auf der Rückseite der Platine können SDRAM-Module mit der maximalen Kapazität von 256 MByte über einen SO-DIMM-Sockel betrieben werden. Das SRAM bietet deutlich weniger Speicherkapazität, ermöglicht aber Zugriffe mit garantierter Latenz. Dieser Speicher wird in einem typischen evolutionären Experiment verwendet, um die Individuen, d.h. die Konfigurationen des HAGEN Chips, die Eingabedaten sowie die Ergebnisdaten jedes Individuums zwischenzuspeichern.

Analoge Bausteine

Das Darkwing-System wird nicht nur zum Training von neuronalen Netzwerkchips verwendet, sondern es wird auch als Testsystem für andere Chipentwicklungen eingesetzt [10] [37]. Dafür stehen zwei Arten von DACs mit einer Umwandlungszeit von 12 μs bzw. 25 ns und ein ADC mit eine Abtastrate von 40 MHz zur Verfügung. Im HAGEN Chip ist die nötige DAC-Funktionalität integriert, daher ist die Schnittstelle zwischen FPGA und HAGEN Chip bis auf die Bereitstellung von Referenzspannungen rein digital.

Periphere Schnittstelle

Um den Netzwerkchip HAGEN oder auch andere *Application Specific Integrated Circuits* (ASICs) im Darkwing-System zu betreiben, wird die periphere Schnittstelle verwendet.

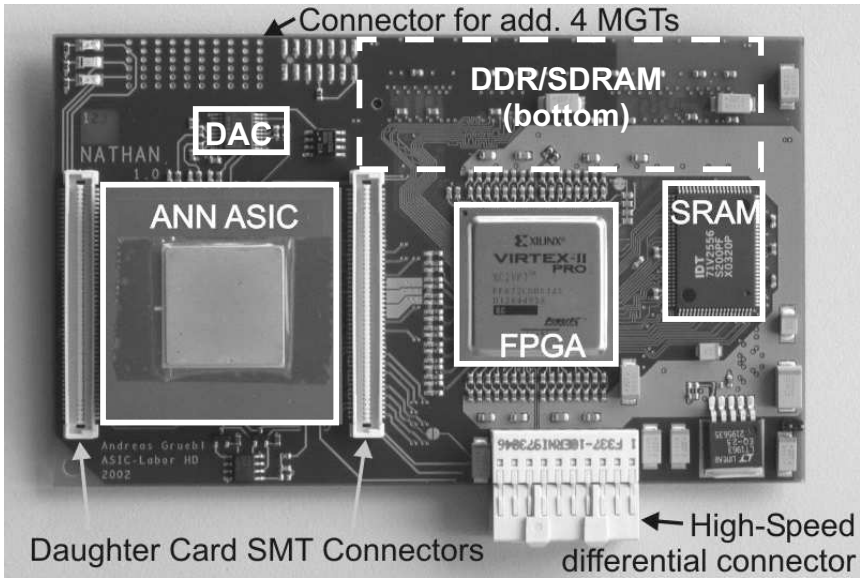

Abbildung 1.17: Fotografie der Oberseite eines Nathan Netzwerkmodules [69]

Abb. 1.16 zeigt die nötigen Leiterplatten, um den HAGEN Chip mittels einer Tochterplatine an den FPGA und damit an die Darkwing-Karte anzuschließen.

1.4.2 Zweite Generation: Nathan-System

Als Weiterentwicklung des Darkwing-Systems wurde am Kirchhoff-Institut für Physik das parallele, verteilte Nathan-System entwickelt. Kernstück ist eine Leiterplatte [25] [69] in den Abmessungen 13 x 8 cm, die die gesamte Funktionalität eines vollständigen Darkwing-System einschließlich des Host-PCs bietet. Möglich ist dies durch die Verwendung des Xilinx Virtex-II Pro, der neben der programmierbaren Logik einen eingebetteten PowerPC zur Verfügung stellt. Auf diesen 32-Bit Prozessor wurde Linux portiert [55], so dass die im Darkwing-System entwickelte Software lokal ausgeführt werden kann. Abb. 1.17 zeigt eine Fotografie der Oberseite der Nathan-Platine, der schematische Aufbau ist in Abb. 1.18 dargestellt. Im Folgenden wird Aufbau, Vernetzung und die Ansteuerung über einen Kontroll-PC dieser Nathan Netzwerkmodule beschrieben.

FPGA

Der zentrale Baustein im verteilten System ist ein Xilinx FPGA der Virtex-II Pro-Serie. Es können die Modelle XC2VP4 und XC2VP7 bestückt werden. Die verwendete Experimentalplattform nutzt den größeren XC2VP7, der 11088 FFs und 11088 LUTs, einen PowerPC Prozessorkern, 8 MGTs und 4 DCMs bietet. Damit stellt der Virtex-II Pro etwa 71% der programmierbaren Logikressourcen im Vergleich zum VirtexE-FPGA des Darkwing-System zur Verfügung.

Speicher

Wie im Darkwing-System sind SRAM-Bausteine fest mit der Nathan-Platine verbunden während SDRAM-Module über einen Sockel eingesteckt werden können. Im Gegensatz zum Darkwing-System können aber sowohl SRAM als auch SDRAM über eigene Adress- und Datenleitungen parallel betrieben werden. Es sind 2 SRAM Bausteine der Firma IDT [35] mit je 512 kByte Kapazität und ein SO-DIMM-Sockel zur Aufnahme von DDR-SDRAM Modulen [49] mit bis zu 1 GB Kapazität auf jeder Nathan-Platine vorhanden.

Analoge Bauteile

Die analogen Referenzspannungen für den HAGEN Chip und neuronale Netzwerkchips der nächsten Generation erzeugt ein Vier-Kanal-DAC MAX5235 der Firma Maxim [43]. Zusätzlich lassen sich die Betriebstemperaturen sowohl des FPGAs als auch des Netzwerkchips mit Hilfe eines Temperatursensors [44] überwachen.

Periphere Schnittstelle

Für den Betrieb des HAGEN Chips ist direkt ein Sockel auf der Nathan-Platine integriert. Die Datenleitungen zu diesem Sockel sowie überzählige Anschlüsse des FPGAs werden aber zusätzlich mit *Surface Mount Technology* (SMT) Steckern verbunden. Netzwerkchips der nächsten Generation können über Tochterplatinen an diesen Steckern mit dem FPGA verbunden werden.

Backplane

Die Backplane bietet Platz für bis zu 16 Nathan Netzwerkmodule und erfüllt neben der mechanischen Halterung folgende Funktionen:

- Über die Backplane sind alle Nathan Netzwerkmodule in einer Kette verbunden. Als weiteres Glied der Kette wird eine Darkwing-PCI-Karte verwendet. Sämtliche Kommunikation zwischen dem Kontroll-PC und den Nathan Netzwerkmodulen, d.h. sowohl die Konfiguration der FPGAs als auch sämtlicher Datentransfer während des Betriebes, wird über diese Verkettung abgewickelt. Dabei erfolgt die Übertragung seriell und ist bei Frequenzen zwischen 60 und 80 MHz stabil. Im Folgenden wird dieser Kommunikationskanal *Slow-Control* genannt.
- Jeder FPGA stellt 8 Multi-Gigabit-Transceiver mit einer maximalen Transferleistung von jeweils 3,125 Gbit/s in beide Richtungen (*duplex*) zur Verfügung. 4 dieser Transceiver sind über die Backplane fest verdrahtet, so dass die 16 Nathan Netzwerkmodule einen 2 dimensionalen Torus bilden.
- Die Backplane stellt die elektrische Infrastruktur für die Nathan Netzwerkmodule bereit, d.h. auf der Backplane werden die Versorgungsspannungen für die FPGAs und Netzwerkchips, sowie die Taktsignale zur Versorgung der FPGA und der Taktung der Multi-Gigabit-Transceiver erzeugt.

Abb. 1.19 zeigt eine Fotografie einer mit 16 Nathan Netzwerkmodulen bestückten Backplane

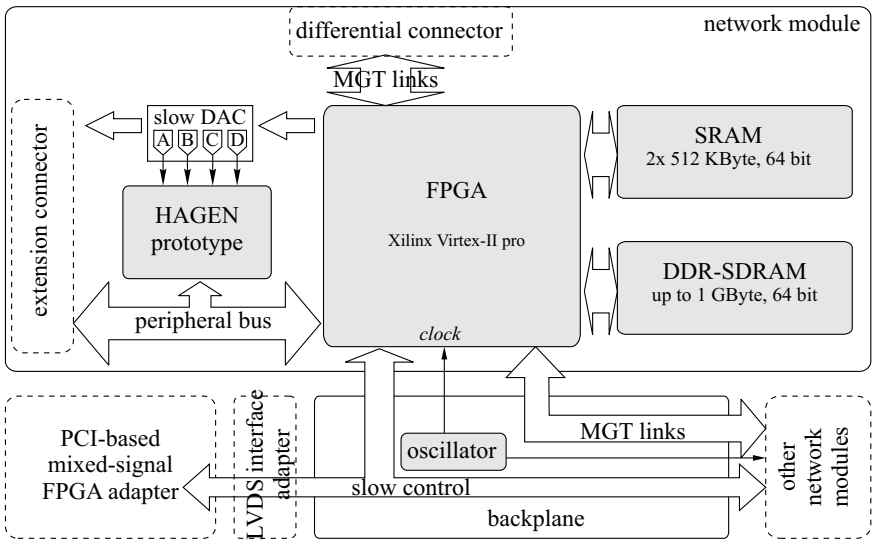

Abbildung 1.18: Schematische Übersicht über ein Nathan Netzwerkmodul. Die Backplane vernetzt über die MGTs die Nathan-Module untereinander, zusätzlich hat der Kontroll-PC Zugriff auf jedes Nathan-Modul über die Slow-Control. Für diesen Zugang wird bereits beschriebene Darkwing PCI-Karte verwendet [69].

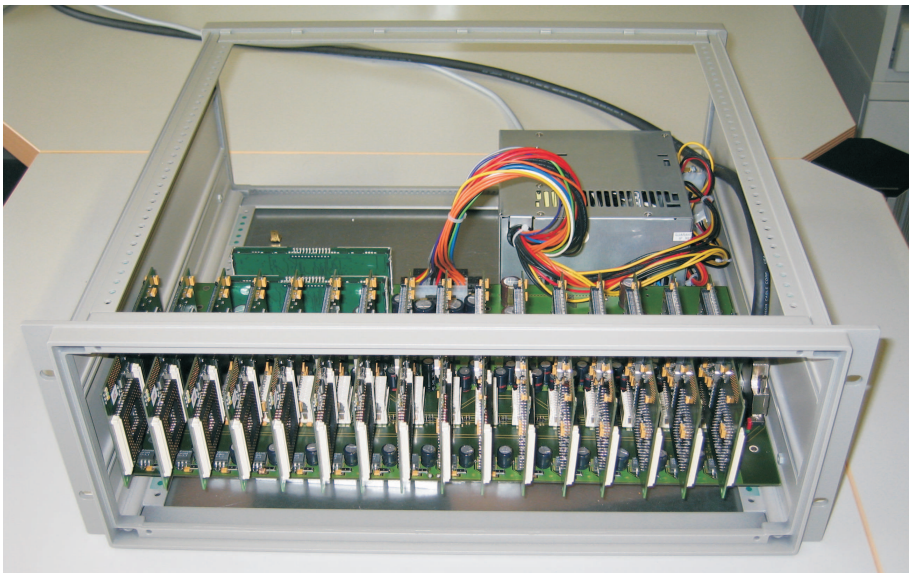

Abbildung 1.19: Fotografie einer mit 16 Nathan Netzwerkmodulen bestückten Backplane. Die Stromversorgung wird durch ein ATX-PC-Netzteil gewährleistet [69].

Kapitel 2

Konzeption

Sowohl die erste Hardwaregeneration, das Darkwing-System, als auch die zweite Generation, das Nathan-System, sind Gemeinschaftsprojekte der Electronic Vision(s) Gruppe. Die zugrunde liegende Hardware und das Leiterplattenlayout ist in [5] [25] [69], die Software [20] in [29] [18] und wichtige Teile der Kommunikation zwischen den verteilten Modulen des Nathan-Systems sind in [55] dokumentiert. Diese Arbeit konzentriert sich auf die System-Integration, die hardwarenahe Software und die programmierbare Logik des Darkwing- und des Nathan-Systems.

In diesem Kapitel werden die grundlegenden Gedanken, die beiden Experimentalplattformen zugrunde liegen, beschrieben. In Kapitel 3 folgt dann eine detaillierte Beschreibung der einzelnen Komponenten der programmierbaren Logik sowie der hardwarenahen Ansteuerungssoftware.

2.1 Anforderungen

Ein sehr wichtiger Aspekt des analog-digitalen Netzwerkchips HAGEN ist seine Geschwindigkeit, sowohl während der Konfigurationsphase, d.h. dem Schreiben der Synapsengewichte, als auch in der darauf folgenden Betriebsphase, in der die neuronale Netzwerkoperation ausgeführt wird (vgl. Abschnitt 1.1.3). Um diese Geschwindigkeit aber praktisch nutzen zu können, muss der HAGEN Chip in ein Hardwaresystem eingebettet werden, das ihn mit Daten versorgt und die vom Chip berechneten Daten sichert bzw. analysiert.

Da neuronale Netzwerke auf jede neue Problemstellung neu trainiert werden, ist das umgebende Hardwaresystem auch für die Implementation der Trainingsalgorithmen zuständig. Da aber die Trainingsalgorithmen selbst Gegenstand aktueller Forschung sind, müssen sie möglichst variabel und ohne Hardware-Expertenwissen, idealerweise in Software, beschrieben und benutzt werden können. Durch den technischen Fortschritt ist zudem zu erwarten, dass es in Zukunft zu Weiter- oder Neuentwicklungen sowohl des Netzwerkchips als auch des Hardwaresystems kommt. Daher sollte die Struktur des Hardwaresystems möglichst modular sein, um bei einer Weiterentwicklung einerseits möglichst viel wiederverwerten zu können und andererseits Verbesserungen gezielt und einfach einfügen zu können. Eine dieser Weiterentwicklungen, der Übergang vom Darkwing-System zum leistungsfähigeren Nathan-System ist im Rahmen dieser Arbeit vollzogen und betreut worden.

Folgende Liste fasst die Anforderungen zusammen:

- Die Ansteuerung der Netzwerkchips
  muss das zeitkritische Protokoll der Schnittstelle sowie die hohen Datenraten des verwendeten Chips unterstützen.
- Die Implementation der Trainingsalgorithmen
  sollte eine ähnlich hohe Ausführungsgeschwindigkeit wie die Netzwerkberechnung erreichen, um das volle Potential des Netzwerkchips nutzen zu können. Um variabel verschiedene Algorithmen einsetzen zu können, ist eine Steuerung des Training in Software essentiell.
- Die Erweiterbarkeit
  wird durch eine hohe Modularität erreicht, dadurch ist ein Wechsel der Hardwareplattform einfach möglich, zudem können verschiedene Trainingsalgorithmen verwendet werden.

Der HAGEN Chip ist besonders für hoch iterative *Chip-in-the-loop*- Algorithmen (vgl. Abschnitt 1.2) geeignet, da sowohl die Netzwerkoperation als auch die Rekonfiguration der Synapsengewichte mit hoher Geschwindigkeit möglich ist. Diese Geschwindigkeit ist aber nur praktisch nutzbar, wenn die Trainingsalgorithmen auch entsprechend schnell implementiert werden können. Ideal geeignet sind daher evolutionäre Algorithmen (vgl. Abschnitt 1.2), die Standardoperatoren *Mutation* und *Rekombination* lassen sich schnell und parallelisiert ausführen.

Diese Arbeit konzentriert sich auf den Aufbau der beiden Experimentalplattformen Darkwing und Nathan zum Training mit evolutionären Algorithmen. Aber auch andere Möglichkeiten des Trainings werden unterstützt und wurden erfolgreich angewandt: Eine Adaption von *Liquid Computing* ([41], [6]) an die Möglichkeiten des HAGEN Chips ist in [11], [68] und [69] dokumentiert, ein anderer Ansatz setzt den HAGEN Chip im Nathan-System zur Mustererkennung ein ([19], [18]).

2.2 Technologien

Betrachtet man die Substrat- oder Technologieebene, so vereinen die beiden vorgestellten Hardwaresysteme Darkwing und Nathan die Vorteile von gemischt analog-digitalem ASIC Design, programmierbarer Logik und Software. Sie bestehen jeweils aus dem HAGEN Chip[1], einem FPGA, einem Mikroprozessor sowie Massenspeicher zur Sicherung von Netzwerkkonfigurationen, Netzwerkdaten sowie Betriebssystem und Programmen für den Mikroprozessor. Eine Übersicht dieser Struktur gibt Abb. 2.1.

Durch die Variabilität der programmierbaren Logik kann das zugrunde gelegte Substrat jedes dieser Module leicht ausgetauscht werden, wie es bei der Weiterentwicklung vom Darkwing-System zum leistungsfähigeren Nathan-System bereits geschehen ist. Der veraltete FPGA Virtex-E wurde durch den Virtex-II Pro ersetzt und statt eines x86-Mikroprozessors im PCI-basierten Darkwing-System wird der eingebettete PowerPC des Virtex-II Pro verwendet.

[1] Das Nathan-System ist darauf ausgelegt, auch Weiter- bzw Neuentwicklungen von Netzwerkchips zu unterstützen.

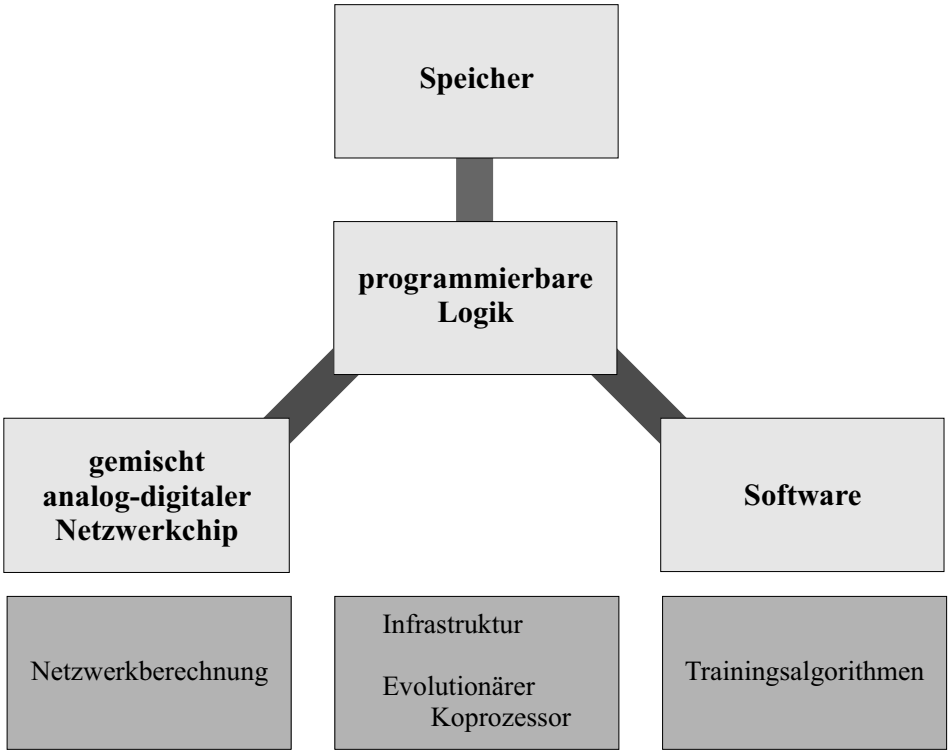

Abbildung 2.1: Struktur der vorgestellten Experimentalplattform: Auf der Substrat-Ebene wird unterschieden zwischen den Technologien gemischt analog-digitale neuronale Hardware, programmierbare Logik in Form eines FPGAs und Software, die auf einem Standard-Prozessor ausgeführt wird. Die Netzwerkberechnung erfolgt in analoger Hardware, die gesamte Kommunikation sowie rechenintensive Teile der Trainingsalgorithmen werden durch den evolutionären Koprozessor im FPGA implementiert, die Trainingsalgorithmen dagegen werden in Software beschrieben.

Neben der beschriebenen Modularität auf der Substrat- bzw. Technologieebene wird auch auf der Funktionsebene ein modularer Aufbau verwendet: Die Arbeitsaufteilung orientiert sich an den Stärken und Schwächen von gemischt analog-digitalem ASIC Design, programmierbarer Logik und Software:
Die *Netzwerkberechnung* erfolgt in den analogen Blöcken des HAGEN Chips (vgl. Abschnitt 1.1.3), die Kommunikation zum und vom HAGEN Chip – in diesem Fall mit dem FPGA – ist aus Gründen der Signalintegrität und Skalierbarkeit rein digital. Die programmierbare Logik sorgt für die *Infrastruktur* innerhalb des FPGAs: Sie stellt die Kommunikation zwischen den einzelnen Bauteilen HAGEN Chip, externen Speicherbausteinen und Mikroprozessor her. Darüber hinaus werden daten- und rechenintensive Teile der Trainingsalgorithmen innerhalb der programmierbaren Logik ausgeführt. Der *Evolutionäre Koprozessor* führt die Berechnungen parallel in einer dedizierten Datenpipeline aus. Die Infrastruktur verwendet einheitliche Schnittstellen, dadurch können die einzelnen Komponenten der programmierbaren Logik einfach ausgetauscht, erweitert oder verbessert werden. Die Trainingsalgorithmen schließlich werden in Software beschrieben, sie sind dadurch einfach und variabel veränderbar.

Die Aufteilung der verschiedenen Arbeitsschritte (vgl. Abschnitt 1.2) des evolutionären Algorithmus sieht folgendermaßen aus:
Die *Evaluation* eines Individuums teilt sich auf in die *Netzwerkberechnung*, die durch den HAGEN Chip ausgeführt wird, und die *Bewertung* der Ergebnisse dieser Berechnung. Der letzte Teil wird im Folgenden *Fitnessberechnung* genannt, i.A. müssen hierbei nur die Ergebnisdaten des neuronalen Netzwerkes mit den Solldaten verglichen werden. Diese Fitnessberechnung ist stark abhängig von der gewählten Problemstellung. Um vollständige Freiheiten in der Wahl der Fitnessfunktion zu gewähren, erfolgt die Fitnessberechnung rein in Software. Die *Selektion* erfordert keinen nennenswerten Rechenaufwand und wird daher auch in Software implementiert. Die *Erzeugung der neuen Generation* dagegen ist sehr daten- und rechenintensiv und die genetischen Operatoren Mutation und Crossover sind unabhängig von der gewählten Problemstellung. Speziell für diesen Arbeitsschritt wurde der evolutionärer Koprozessor entwickelt, der die neue Generation in dedizierter Hardware erzeugt, dabei aber vollständig von der Trainingssoftware gesteuert wird. Durch diesen Koprozessoransatz ist es möglich, die Geschwindigkeit der Hardwareimplementation unter voller Kontrolle der Software zu nutzen.

Im Folgenden wird zuerst die allgemeine Struktur der programmierbaren Logik beschrieben, dann folgt eine Abwägung der Eigenschaften des PCI-basierten Darkwing-Systems und des leistungsfähigeren verteilten Nathan-Systems.

2.3 Struktur der programmierbaren Logik

Ein FPGA ist der zentrale Baustein sowohl der Darkwing- als auch der Nathan-Plattform. Die programmierbare Logik implementiert die Schnittstellen zu der Software des Hauptprozessors, zum externen Speicher und zum HAGEN Chip und sie ermöglicht die Kommunikation zwischen diesen Bauteilen bzw. Komponenten. Zusätzlich werden Teile der Trainingsalgorithmen durch den evolutionären Koprozessor im FPGA ausgeführt.

Bezüglich des Prozessors und des Speichers muss allerdings zwischen den beiden Plattformen Darkwing und Nathan unterschieden werden:

Auf dem Darkwing-System wird der Mikroprozessor des PC-Systems verwendet, der auch für Betriebssystem und Programme den Speicher des PC-Systems verwendet. Um Netzwerkdaten mit der programmierbaren Logik auszutauschen, werden diese über den PCI-Bus in den externen Speicher auf der Darkwing-Platine übertragen. Auf dem Nathan-System wird sowohl für Betriebssystem, Programme und Netzwerkdaten der externe Speicher auf der Nathan-Platine verwendet, auf den sowohl der eingebettet PowerPC als auch die programmierbare Logik direkt zugreifen können.

Die programmierbare Logik ist modular aufgebaut, die wichtigsten Komponenten innerhalb des FPGAs zeigt die Abb. 2.2. Die Komponenten lassen sich in zwei Gruppen unterteilen: Die physikalischen Schnittstellen einschließlich der Datenpfade zum externen Speicherbaustein, zum Netzwerkchip und zum Hauptprozessor sind hellgrau unterlegt. Sie kapseln die unterschiedlichen Eigenschaften der beiden Experimentalplattformen, wie etwa die physikalische Schnittstelle zwischen dem FPGA und dem HAGEN Chip. Diese Komponenten sind plattformabhängig und wurden jeweils speziell auf jede Plattform angepasst. Die dunkelgrau dargestellten Komponenten können durch diese Kapselung plattformunabhängig beschrieben werden.

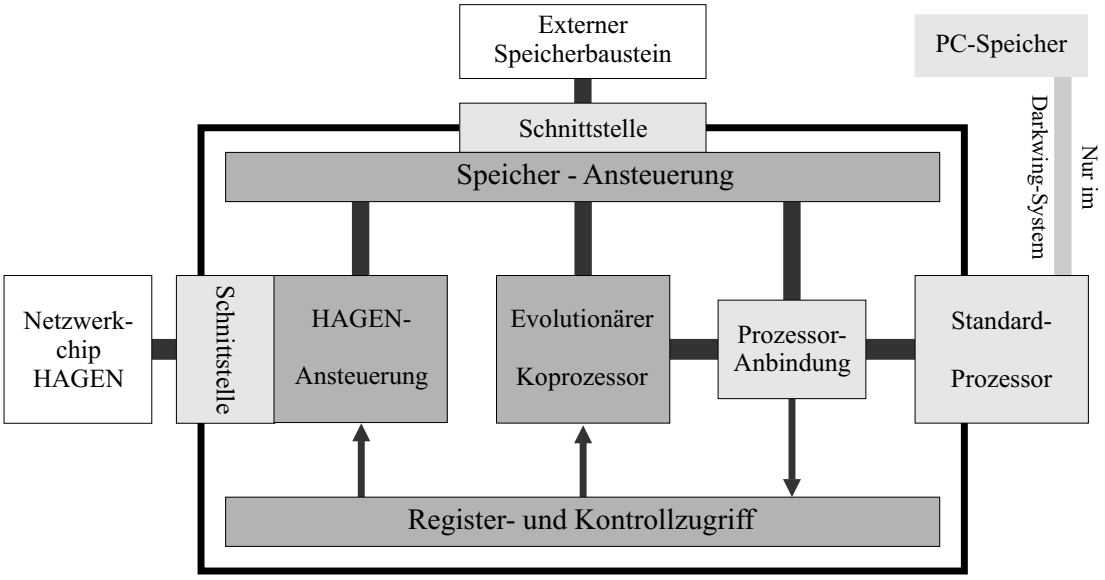

Abbildung 2.2: Struktur der programmierbaren Logik: Die plattformspezifischen Schnittstellen und die Anbindung des Standard-Prozessors sind hellgrau, die auf beiden Plattformen verwendeten Komponenten dunkelgrau dargestellt. Der Standard-Prozessor ist im Nathan-System Teil des FPGAs, im Darkwing-System wird der Prozessor des PCs und zusätzlich der Speicher innerhalb des PCs verwendet.

Der evolutionäre Koprozessor verwendet beispielsweise die standardisierten Schnittstellen zur Speicheransteuerung und wird über den Register- und Kontrollzugriff überwacht. Diese Modularität ist Voraussetzung, um ein derart komplexes System erweiterbar und wartbar zu halten. Im Folgenden wird ein kurzer Überblick über die Aufgaben der einzelnen Komponenten gegeben, in Kap. 3 folgt die detaillierte Beschreibung.

- Wie die Abb. 2.2 zeigt, greifen verschiedene Komponenten auf den externen Speicherbaustein zu. Die *Speicheransteuerung* kapselt die physikalische Schnittstelle zum externen Speicherbaustein und stellt eine beliebige Anzahl von internen, logischen Speicherschnittstellen, im Folgenden *Ramklienten* genannt, für die restlichen Komponenten innerhalb des FPGAs zur Verfügung.

- Der *Register- und Kontrollzugriff*, im Folgenden *Slow-Control* genannt, ermöglicht es der Software, mit begrenzter Datenrate Kontroll- und Steuerungsdaten der einzelnen Komponenten zu schreiben und zu lesen. Die Slow-Control stellt dabei für jede angeschlossene Komponente eine eigene logische Schnittstelle, im Folgenden *Slow-Control-Klient* genannt, zur Verfügung.
  Sowohl Speicheransteuerung als auch Slow-Control erlauben eine beliebige Anzahl an Ramklienten bzw. Slow-Control-Klienten. Die einzelnen Klienten sind dabei voneinander unabhängig. Nur durch diese Unabhängigkeit ist es möglich, das System wartbar, übersichtlich und erweiterbar zu halten, neue Klienten können beispielsweise ohne Änderung der bestehenden Komponenten hinzugefügt werden.

- Die *HAGEN-Ansteuerung* bedient die physikalische Schnittstelle zum Netzwerkchip HAGEN und steuert den Ablauf der Gewichtskonfiguration und Netzwerkoperation. Durch die Chip-interne Organisation der Datenpfade innerhalb des HAGEN Chips müssen alle Daten vorverarbeitet werden, bevor sie an die physikalische Schnittstelle weitergegeben werden. Diese Verarbeitung wird innerhalb der HAGEN-Ansteuerung vorgenommen. Netzwerkgewichte, Eingabe- und Ausgabedaten für den HAGEN Chip werden über drei Ramklienten aus dem externen Speicher gelesen.

- Der *evolutionäre Koprozessor* berechnet die daten- und rechenintensiven Teile des evolutionären Algorithmus, er wird mittels eines speziellen Instruktionssatzes durch die Trainingssoftware gesteuert. Das genetische Material, d.h. die einzelnen Netzwerkkonfigurationen werden dabei über Ramklienten aus dem externen Speicher gelesen, bzw. dorthin zurück geschrieben.

- Der Anschluss des Hauptcomputers ist in beiden Systemen sehr unterschiedlich. Im Darkwing-System wird der Standard-Prozessor des PC verwendet, die Kommunikation verläuft über den PCI-Bus. Im Nathan-System dagegen werden die Trainingsalgorithmen durch den eingebetteten PowerPC ausgeführt, der unterschiedliche Kommunikationskanäle zur programmierbaren Logik bietet (vgl. Anschnitt 1.3.2).

2.4 Darkwing-System

In Abschnitt 1.4.1 wurde die Hardware der Darkwing PCI-Karte vorgestellt. Als Hauptcomputer wird ein handelsüblicher PC verwendet. Die Trainingssoftware wird durch den Mikroprozessor des Hauptcomputers ausgeführt. Obwohl bereits eine Reihe von Experimenten auf dem Darkwing-System durchgeführt wurden und werden [11] [70] [31] [30] [68] [52] [69] [19] [18], hat dieses System einige Einschränkungen, vor allem in Bezug auf die zu erreichenden Trainingsgeschwindigkeiten, die Skalierbarkeit und die Erweiterungsmöglichkeiten:

- Die Trainingsalgorithmen erfordern einen schnellen Datenaustausch zwischen dem neuronalen Netzwerk und der Trainingssoftware. Dieser Datenaustausch muss im Darkwing-System über den PCI-Bus abgewickelt werden. Dieser hat eine theoretische[2] maximale Bandbreite von 132 MB/s, was die Datentransferrate zwischen Software und neuronalem Netzwerk und damit die Trainingsgeschwindigkeit stark begrenzt.

- Jeder eingesetzte neuronale Netzwerkchip hat ein feste Größe, d.h. eine feste Anzahl an Neuronen und Synapsen. Wenn größere Netzwerke betrachtet werden sollen, können aufgrund der digitalen Schnittstelle mehrere Netzwerkchips parallel betrieben werden. Die Kommunikation zwischen zwei oder mehr dieser Netzwerkchips müsste aber wiederum über den PCI-Bus laufen. Durch die geringe Bandbreite und die nicht definierte Antwortzeit bzw. Latenz des PCI-Protokolls ist diese parallele Nutzung im Darkwing-System nur sehr eingeschränkt möglich.

[2] In der Praxis erreicht der PCI-Bus etwa eine Rate von 80-90 MB/s.

- Auch die parallele Nutzung von mehreren HAGEN Chips unabhängig voneinander ist aufwendig, weil jeder HAGEN Chip ein eigenes PC-System benötigt. Prinzipiell lassen sich natürlich mehrere Darkwing PCI-Karten in einem PC betreiben, allerdings wird damit auch die vorhandenen Bandbreite des PCI-Busses auf die verschiedenen Darkwing Karten aufgeteilt.

- Aufgrund der Eigenschaften des HAGEN Chips (vgl. Abschnitt 3.3) und des im Darkwing-System verwendeten FPGA ist es nicht möglich, die Schnittstelle zwischen beiden Bauteilen mit höheren Frequenzen als 84 MHz zuverlässig zu betreiben[3] Dies schränkt die maximal erreichbare Geschwindigkeit der Konfiguration und Netzwerkberechnung stark ein.

Dabei ist zu beachten, dass die aufgezählten Einschränkungen nicht nur für den Betrieb des HAGEN gelten, sondern auch eventuelle Weiterentwicklungen des HAGEN Chips oder neuartige Netzwerkchips betreffen. Durch die i.A. größeren Datenraten und höheren Geschwindigkeiten der Schnittstellen dieser Neuentwicklungen würden sich die genannten Einschränkungen zudem deutlich stärker auswirken.

2.5 Nathan-System

Um den HAGEN Chip effektiv und schnell trainieren können und als Test- und Trainingssystem für künftige Generationen von Netzwerkchips wurde eine neue Experimentalplattform entwickelt, das verteilte Nathan-System, dessen Hardware in Abschnitt 1.4.2 beschrieben wurde. Jedes Nathan Netzwerkmodul, von denen bis zu 16 in einer Backplane zusammengefasst werden können, ersetzt ein vollständiges Darkwing-System einschließlich des PCs. Dabei stellt der verwendete FPGA *Virtex-II Pro* [86] sowohl die programmierbare Logik, einen PowerPC Mikroprozessor als auch spezielle Hochgeschwindigkeitslinks zur Kommunikation mit anderen Nathan Netzwerkmodulen zur Verfügung.

Die Einschränkungen des Darkwing-Systems werden dadurch überwunden: Die Trainingssoftware wird lokal auf dem eingebetteten PowerPC ausgeführt und hat damit über verschiedene Kanäle[4] direkten und unmittelbaren Zugriff auf die programmierbare Logik. Der Datenaustausch zwischen Software und dem HAGEN Chip ist dadurch mit großer Geschwindigkeit und kleiner Latenz möglich. Auch die unabhängige parallele Nutzung oder Vernetzung von mehreren Netzwerkchips stellt kein Problem dar: In einer Backplane können bis zu 16 Nathan Netzwerkmodule und damit auch 16 Netzwerkchips betrieben werden, die seriellen Multi-Gigabit-Transceiver (vgl. Abschnitt 1.3.2) erlauben zudem eine schnelle Kommunikation zwischen den einzelnen Nathan Modulen ([85] [55]). Durch spezielle Bausteine zur Taktgenerierung und eine Verbesserung der I/O Zellen innerhalb des FPGAs ist die Ansteuerung des HAGEN Chips mit höheren Taktraten machbar, zudem profitieren auch künftige neuronale Netzwerkchips von diesen verbesserten Eigenschaften der FPGA-Schnittstelle.

[3] Der verwendete FPGA Virtex-E bietet keine Möglichkeiten, die Phase des Schnittstellentaktes gegenüber der Phase des Datentaktes zu verschieben, ebensowenig sind dedizierte DDR-I/O-Zellen zur Ein- und Ausgabe vorhanden.

[4] Die Möglichkeiten des On-Chip-Memory (OCM), Device Control Register (DCR) und Processor Local Bus (PLB) wurden in Abschnitt 1.3.2 beschrieben.

2.6 Software

Parallel zur Entwicklung der Hardwaresysteme wurde ein umfangreiches Softwarepaket entwickelt, das *Heidelberg Analog Neural Network Evolution Environment* (HANNEE) [20] [29], [18], um hardwarebasierte neuronale Netzwerke und insbesondere den HAGEN Chip zu testen und zu trainieren. Der Zugriff der Trainingsalgorithmen auf die verwendete Hardware erfolgt nicht direkt, sondern über die *Hardware-Abstraktionsebene* (HAE). Dadurch können die Trainingsalgorithmen und Anwendungen innerhalb des HANNEE Paketes weitgehend unabhängig von der verwendeten neuronalen Hardware und der aktuellen Experimentalplattform beschrieben werden. In dieser Arbeit wurde die Klassenstruktur des HAE entwickelt, die die Unterschiede der beiden Hardwaresysteme Darkwing und Nathan für die oberen Hierarchieebenen kapselt. Dieser Teil der HANNEE Software wird in Abschnitt 3.6.1 erläutert, eine Beschreibung der Algorithmen und Anwendung findet sich in [29] [18] und [69].

Kapitel 3

Implementation

Nachdem in Kapitel 2 die Ideen und Konzepte der beiden Experimentalplattformen Darkwing und Nathan vorgestellt wurden, beschreibt dieses Kapitel die Implementation und Funktionsweise der Komponenten innerhalb der programmierbaren Logik sowie die hardwarenahe Software, die zum Betrieb beider Plattformen entwickelt wurde. Zuerst werden die Komponenten beschrieben, die die Infrastruktur innerhalb des FPGAs bilden: Die *Speicheransteuerung* (Abschnitt 3.1) ermöglicht den Zugriff auf den externen Speicher mit hoher Geschwindigkeit und die *Slow-Control* (Abschnitt 3.2) erlaubt es, Register- und Kontrollzugriffe auszuführen. Dann folgen die Komponenten, die den HAGEN Chip anbinden und Teile der Trainingsalgorithmen in programmierbarer Logik ausführen: Die *HAGEN-Ansteuerung* wird in Abschnitt 3.3 und der *Evolutionäre Koprozessor* in Abschnitt 3.4 beschrieben. Da das gesamte System von der Trainingssoftware gesteuert wird, ist eine schnelle Interaktion zwischen Software und der vorgestellten programmierbaren Hardware sehr wichtig, die dafür entwickelten Komponenten innerhalb des FPGAs, die nötigen Software-Treiber und die hardwarenahe Software selbst werden in Abschnitt 3.6 beschrieben.

Da die zweite Hardware-Generation, das Nathan-System, die aktuellere und leistungsfähigere Plattform ist, wird bei plattformspezifischen Komponenten besonders auf die Implementation im Nathan-System eingegangen, besonders die physikalischen Schnittstellen zwischen FPGA und HAGEN Chip, bzw. zwischen FPGA und DDR-SDRAM im Nathan-System werden genauer beschrieben.

Für die Synchronisation mit externen Bausteinen und auch für die interne Taktung innerhalb des Nathan FPGAs von großer Bedeutung sind die DCMs (vgl. Abschnitt 1.3.2). Der im Nathan-System verwendete FPGA stellt 4 DCMs zur Verfügung, die alle genutzt werden, bzw. für künftige Erweiterungen reserviert sind: Je ein DCM wird von der HAGEN-Ansteuerung zur Synchronisation mit dem HAGEN Chip und von der Speicheransteuerung zur Synchronisation mit dem externen Speicherbaustein verwendetet. Der dritte DCM ist für zukünftige Erweiterungen für die Synchronisation mit dem externen SRAM-Bausteinen reserviert, während der vierte und letzte DCM zur Generierung der internen Taktgeber verwendet wird.

3.1 Speicheransteuerung

In Kap. 2 wurde gezeigt, dass mehrere Komponenten der programmierbaren Logik den externen Speicher verwenden. Diese werden im Folgenden *Ramnutzer* genannt. Im Darkwing- wie auch im Nathan-System sind sowohl SRAM als auch SDRAM Bausteine vorhanden[1]. Die *Speicheransteuerung* stellt die Schnittstelle zwischen diesen externen Speicherbausteinen und den verschiedenen Ramnutzern innerhalb der programmierbaren Logik dar. Sie besteht aus drei Teilen, die als eigenständige Objekte implementiert sind, den Aufbau zeigt Abb. 3.1. Die einzelnen Teile und ihre Aufgaben sind:

- Die *Ramklienten* stellen jedem Ramnutzer eine logische Speicherschnittstelle mit einheitlichem Protokoll zur Verfügung. Sie können mehrere Zugriffsanforderungen der Ramnutzer zwischenspeichern, diese werden an den Manager weitergeleitet.
- Der *Manager* bekommt die Zugriffsanforderungen der Ramklienten, entscheidet anhand einer vorgegebenen Priorität in jedem Takt, welcher Ramklient einen Zugriff ausführen darf und leitet diese Zugriffe an die Ramkontrolleinheit weiter.
- Die *Ramkontrolleinheit* bedient und kapselt weitestgehend die physikalische Schnittstelle zum externen Speicherbaustein. Die Zugriffe, die der Manager stellt, werden in Befehlssequenzen für die jeweils angeschlossenen Speicher übersetzt. Wird SDRAM verwendet, so müssen durch die interne Struktur der SDRAM-Bausteine Auffrischzyklen eingefügt werden, ebenso wird die Organisation der Zeilenwechsel der einzelnen Speicherbänke durch die Ramkontrolleinheit erledigt (vgl. Abschnitt 1.3.2).

Da sowohl die HAGEN-Ansteuerung als auch der Koprozessor Datenströme mit hoher Geschwindigkeit verarbeiten, ist die Speicheransteuerung darauf ausgelegt, um den parallelen Zugriff mehrere Ramnutzer effektiv zu organisieren. Die Speicheransteuerung verfolgt zwei weitere Ziele: Zum einen soll ein Ramnutzer unabhängig von anderen, eventuell gleichzeitig aktiven Ramnutzern sein. Diese Unabhängigkeit wird nur eingeschränkt durch die Bandbreite des externen Speicherbausteins, die nach einer fest vorgegebenen Priorität verteilt wird. Zum anderen soll die Entwicklung und Kodierung der Ramnutzer unabhängig vom tatsächlich vorhandenen externen Speicherbaustein sein, sei es nun SRAM oder SDRAM auf dem Darkwing- oder Nathan-System.

Vor allem in der Antwortzeit sind die Eigenschaften der Speicherbausteine aber sehr unterschiedlich: Im Gegensatz zu SRAM Speicher kann SDRAM-Speicher wegen Auffrischzyklen oder Zeilenwechseln keine feste Antwortzeit garantieren (vgl. Abschnitt 1.3.2). Dieser Unterschied ist aber in einem System mit mehreren Ramnutzern ohne Bedeutung: Der einzelne Ramnutzer kann ohnehin nicht mit einer festen Antwortzeit rechnen, da eventuell ein anderer Ramnutzer höhere Priorität hat und vorgezogen wird.

Das vorgestellte Konzept garantiert deshalb keine festen Antwortzeiten. Der Ramnutzer gibt seine Zugriffsanforderungen an seinen Ramklienten. Ein Schreibzugriff ist damit ohne Schreib-Bestätigung abgeschlossen. Im Falle eines Lesezugriffs kann der Ramnutzer eine unbestimmte Zeit später die gelesenen Daten wiederum von seinem Ramklienten übernehmen. Weiterhin werden keine festen Blockgrößen beim Transfer vorgeschrieben.

[1] Im Darkwing-System stehen gegenwärtig 2 MB SRAM und 64 MB SDRAM (max. 512 MB), im Nathan-System 1 MB SRAM und 256 MB SDRAM (max. 1 GB) zur Verfügung.

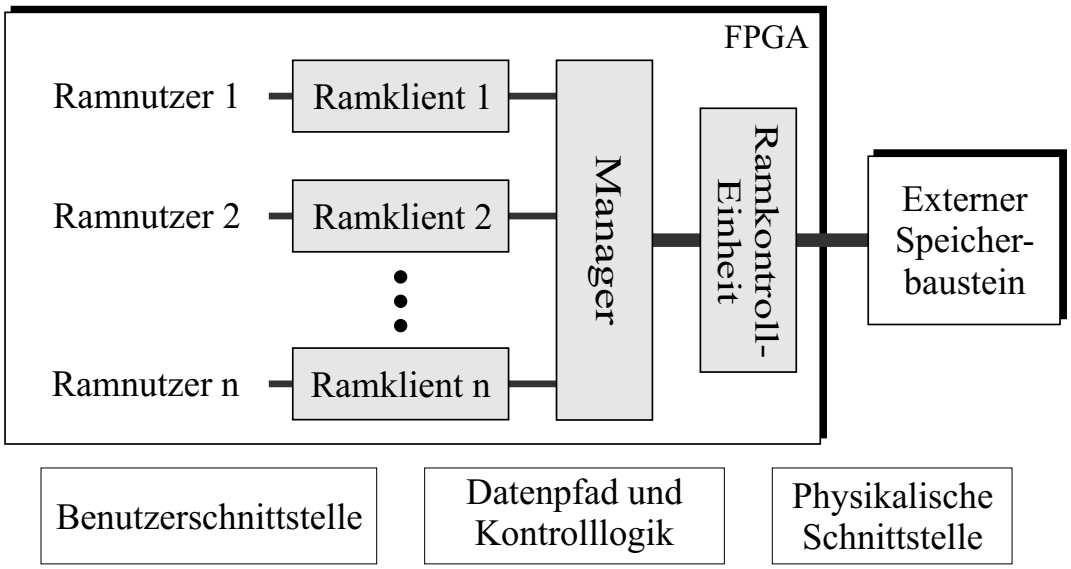

Abbildung 3.1: Aufbau der Speicheransteuerung: Die Benutzerschnittstelle besteht aus je einem *Ramklienten* für jeden *Ramnutzer*. Durch Zwischenspeicherung der Zugriffsanforderungen sind die Ramklienten unabhängig voneinander. Der *Manager* entscheidet, wann welcher Ramklient einen Zugriff auf den *externen Speicherbaustein* ausführen darf und implementiert die Datenpfade zur Ramkontrolleinheit, die die physikalische Schnittstelle zum externen Speicherbaustein außerhalb des FPGAs bedient.

Durch die Kapselung der schnittstellenspezifischen Details in der Ramkontrolleinheit können sowohl für die SRAM Bausteine auf Darkwing [64] und Nathan [35] als auch das SDRAM Modul auf Darkwing [47] nahezu die gleichen Manager und Ramklienten verwendet werden. Dem DDR-SDRAM Modul im Nathan-System [49] [48] fällt eine Sonderrolle zu, weil dieses DDR-Technologie verwendet, aus diesem Grund musste ein spezieller Manager sowie ein spezieller Ramklient entwickelt werden.

Im Folgenden wird zuerst die Organisation der Datenpfade von den Ramnutzern bis zum externen Speicherbaustein erklärt. Danach folgt eine Beschreibung der einzelnen Komponenten Ramklient, Manager und Ramkontrolleinheit am Beispiel der DDR-Schnittstelle zum SDRAM im Nathan-System.

3.1.1 Datenbreiten und Taktung

Das externe DDR-SDRAM Modul im Nathan-System bearbeitet pro Takt einen Zugriff, dabei werden sowohl mit der steigenden als auch der fallenden Taktflanke je 64 Bit Daten übertragen. Die Speicheransteuerung im FPGA muss daher für Schreibzugriffe in jedem Takt 128 Bit an Daten bereitstellen, bzw. bei Lesezugriffen 128 Bit an Daten akzeptieren können.

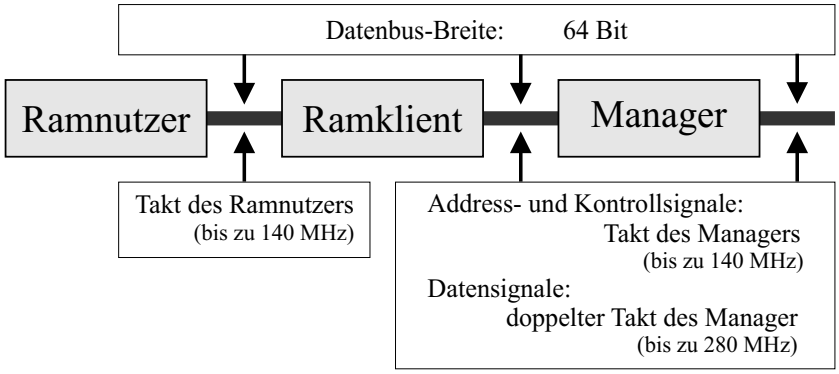

Abbildung 3.2: Datenpfad von den Ramnutzern über Ramklienten und Manager zum externen Speicherbaustein. Die Schnittstelle zum externen Speicher wird mit doppelter Datenrate betrieben, die Schnittstelle zwischen Ramklient und Ramnutzer allerdings nur mit einfacher Datenrate. Da die jeweiligen Datenbusse die gleiche Breite von 64 Bit haben, müssen die Daten in den Ramklienten zwischengespeichert werden.

Die Schnittstelle zu den Ramnutzern wurde allerdings aus folgenden Gründen mit einer Breite von 64 Bit implementiert:

- Das SDRAM-Modul und der SRAM-Baustein im Darkwing-System und auch der SRAM-Baustein im Nathan-System haben eine physikalische Datenbreite von 64 Bit. Um mit diesen Bausteinen kompatibel zu bleiben, d.h. um den Ramnutzern eine einheitliche Schnittstelle unabhängig vom aktuell vorhandenen System oder Speicher anbieten zu können, muss die Schnittstelle zu den Ramnutzern 64 Bit breit sein. Da jeder einzelne Ramnutzer nur 64 Bit Daten pro Takt verarbeiten kann, würden 128 Bit breite Datenleitungen keine Geschwindigkeitsvorteile bringen.
- Die Ramklienten müssen vollständige Zugriffsanforderungen sowie die Ergebnisse von Lesezugriffen für den jeweils angeschlossenen Ramnutzer zwischenspeichern. Eine Ausführung der Datenleitungen in 128 Bit Breite ist wegen der begrenzten Routing-Ressourcen im verwendeten FPGA nicht praktikabel. Da die Zwischenspeicherung der Daten in Blockrams (siehe Abschnitt 3.1.2) erfolgt, diese aber eine eine maximale Datenbreite von 32 Bit erlauben, verdoppelt eine Datenbreite von 128 Bit zusätzlich die Anzahl der belegten Blockrams.

Die Datenpfade der Speicheransteuerung sind folgendermaßen organisiert (vgl. Abb. 3.2): Die Schnittstelle zwischen Ramnutzer und Ramklient hat einen 64 Bit breiten Datenbus und wird mit dem Takt des Ramnutzers betrieben, der bis zu einer Betriebsfrequenz von 140 MHz getestet wurde (siehe Abschnitt 3.1.5). Die Schnittstellen vom Ramklienten über Manager, Ramkontrolleinheit zum externen Speicherbaustein verwenden zwei synchrone Takte: Die Adressen und die Kontrolllogik werden mit dem Takt des Managers weitergegeben, die Daten werden in beiden Richtungen allerdings mit dem doppelten Takt des Managers aktualisiert, und entsprechen damit der physikalischen DDR-Schnittstelle. Der Ramklient stellt folglich eine Taktgrenze zwischen dem Takt eines Ramnutzers und dem Takt des Managers dar.

Die Beschränkung auf eine einheitliche Datenbreite von 64 Bit von den Ramnutzern über Manager bis zum externen Speicherbaustein verringert den Ressourcenverbrauch, allerdings müssen dadurch die Daten mit der doppelten Frequenz getaktet werden. Im Folgenden werden 3 unterschiedliche Taktfrequenzen unterschieden (vgl. auch Abb. 3.2): Der *Takt des Managers* und der *Doppelte Takt des Managers* werden durch die Speicheransteuerung bestimmt, davon unabhängig und auch asynchron kann der *Takt des Ramnutzers* durch den Ramnutzer gewählt werden. Wird allerdings der Takt des Ramnutzers und der Takt des Managers identisch gewählt, so kann ein einzelner Ramnutzer maximal die Hälfte der verfügbaren Bandbreite des externen Speicherbausteins beanspruchen. Um die Schnittstelle zum externen Speicherbaustein voll auszulasten, sind folglich mindestens zwei kontinuierlich aktive Ramnutzer nötig.

3.1.2 Ramklient

Der Ramklient ist die Schnittstelle zum Ramnutzer, er speichert die Zugriffsanforderungen des Ramnutzers, bis der Manager sie bearbeitet, bzw. an die Ramkontrolleinheit weitergibt. Auch aus dem externen Speicher gelesene Daten werden im Ramklienten zwischengespeichert, bis der Ramnutzer sie übernehmen kann. Für diese Zwischenspeicherung werden im Ramklient drei FIFO-Speicher zusammen mit der zugehörigen Logik zur Steuerung der Schnittstellenprotokolle zum Ramnutzer und zum Manager verwendet. Wenn sich der Takt des Ramnutzers vom Takt des Manager in Frequenz oder Phase unterscheidet, so können asynchrone FIFO-Speicher instantiiert werden, sind die beiden Takte gleich, so werden synchrone FIFO-Speicher verwendet, um Ressourcen zu sparen.

Jeder Ramklient kann vollständige Zugriffsanforderungen sowie die Ergebnisse von Lesezugriffen zwischenspeichern. Dadurch ist es möglich, die Daten für den schnellen Takt des Managers zu akkumulieren und die logischen Zugriffe des Ramnutzers von den physikalischen Zugriffen auf den externen Speicherbaustein zeitlich zu entkoppeln. Die drei FIFO-Speicher enthalten die Adressen und die Daten, die zum Speicherbaustein geschrieben werden, bzw. die von dort gelesenen wurden.

Die Daten-FIFOs verwenden jeweils zwei der 32 Bit breiten *Blockrams* (vgl. Abschnitt 1.3.2). Um Blockrams zu sparen, wird der Adressen-FIFO in *Distributed Ram*, d.h. in den Logik-Elementen implementiert, er verfügt über eine Speichertiefe von 16 Einträgen und kann sowohl Schreib- als auch Leseadressen speichern. Durch größere Tiefen der FIFO-Speicher könnten mehr Zugriffsanforderungen vorgehalten und daher größere Blöcke übertragen werden. Dadurch sind weniger Schreib-Lesewechsel und auch weniger Zeilenwechsel nötig, der Datendurchsatz steigt. Allerdings steigt auch der Ressourcenverbrauch der FIFO-Kontrolllogik und der Speicherbedarf des Adressen-FIFOs, die gewählten Speichertiefen stellen daher einen Kompromiss zwischen Ressourcenverbrauch und erzieltem Datendurchsatz dar. Werden für spezielle Anwendungen deutlich größere Speichertiefen benötigt, so kann auch der Adressen-FIFO in einem Blockram realisiert werden.

Die Schnittstelle zum Ramnutzer ist derart konzipiert, dass der Ramklient sich leicht als Datenquelle oder Datensenke in eine Pipeline einbauen lässt. Die genaue Schnittstellenbeschreibung und Beispiele des Quelltextes zur Ansteuerung finden sich in Anhang A.

3.1.3 Manager

Der Manager stellt die Verbindung zwischen den Ramklienten und der Ramkontrolleinheit dar. Er ist dafür zuständig, die Zugriffsanforderungen der vorhandenen Ramklienten zu arbitrieren, d.h. er entscheidet, welcher der Ramklienten wann auf den externen Speicher zugreifen darf. Zusätzlich fasst der Manager die Daten- und Adressleitungen der Ramklienten zusammen und gibt pro Takt maximal eine gültige Adresse bzw. pro Taktflanke ein gültiges 64 Bit Datenwort an die Ramkontrolleinheit weiter.

Arbitrierung Es wird eine erweiterte prioritätsbasierte Aufteilung der Zugriffe verwendet. Die Ramklienten haben eine zur Synthesezeit festgelegte eindeutige Priorität, jede Priorität wird dabei nur einmal vergeben. Der Manager setzt in jedem Takt maximal einen Ramklienten aktiv, der genau einen Zugriff durchführen darf. Vorrang hat derjenige Ramklient, der als letzter aktiv war. Sollte dieser im aktuellen Takt keinen Zugriff anfordern, so wird anhand der Priorität entschieden. Diese Vorgehensweise sorgt dafür, dass jeder Ramklient seine gesamten akkumulierten Zugriffe abarbeiten kann, sobald er einmal auf aktiv gesetzt wird. Damit wird sichergestellt, dass die übertragenen Blöcke groß sind und wenig Bandbreite durch Schreib-Lesewechsel oder Zeilenwechsel verloren geht. Im vorliegenden Testsystem ist die maximale Anzahl an Zugriffen, die ein Ramklient mit dieser Methode durchführen kann, durch die Tiefe der FIFO-Speicher begrenzt. Da die Ramnutzer mit einfachem Takt, der Manager aber mit doppeltem Takt Daten verarbeiten, läuft der Adressen-FIFO mit 16 Einträgen Speichertiefe nach maximal 16 Ramnutzer-Takten[2], d.h. 32 Zugriffen von 64 Bit breite, leer.

Datenweiche - Schreibpfad Jeder Ramklient schließt mit seinen Adressen- und Datenleitungen an den Manager an. Der Manager gibt pro Takt maximal einen Zugriff, d.h. eine Adresse und im Falle eines Schreibzugriffs die Daten an die Ramkontrolleinheit weiter. Der Manager muss folglich die Adressen- und Datenleitungen aller Ramklienten zusammenfassen und an die Ramkontrolleinheit weitergegeben. Dabei ist vor allem der Datenpfad in Richtung zur Ramkontrolleinheit problematisch. Die Daten werden mit doppeltem Takt und 64 Bit breit aus den Blockrams der FIFO-Speicher gelesen. Dann werden die Daten des aktiven Ramklienten über zwei Zwischenregister an den externen Speicher weitergeleitet. Es wurden zwei Implementationen dieser Datenweiche getestet: Die Verwendung eines Tristate-Busses wie in Abb. 3.3a dargestellt und eine Multiplexer-Baumstruktur wie in Abb. 3.3b. Da die Tristate-Treiber im FPGA vom restlichen Design kaum genutzt werden und somit reichlich zur Verfügung stehen, ist die Implementation mit Tristates sehr ressourcenschonend. Die Verwendung der schnelleren Multiplexer zusammen mit der Struktur des Binärbaums, der pro Stufe nur zwei Eingänge zusammenfasst und dadurch das Routing erleichtert, erreicht dagegen höhere Taktfrequenzen. Durch die Eigenschaften des Binärbaumes ist aber dabei die maximale Anzahl an Schreibklienten auf vier begrenzt und es werden wegen der doppelt ausgeführten Registerstufe zwischen Ramklient und I/O-Zelle mehr Ressourcen verbraucht (vgl. Abb. 3.3 Teil b).

[2] In 16 Takten kann der Ramnutzer maximal 16 neue Zugriffsanforderungen schreiben, der Manager aber 32 Anforderungen bearbeiten.

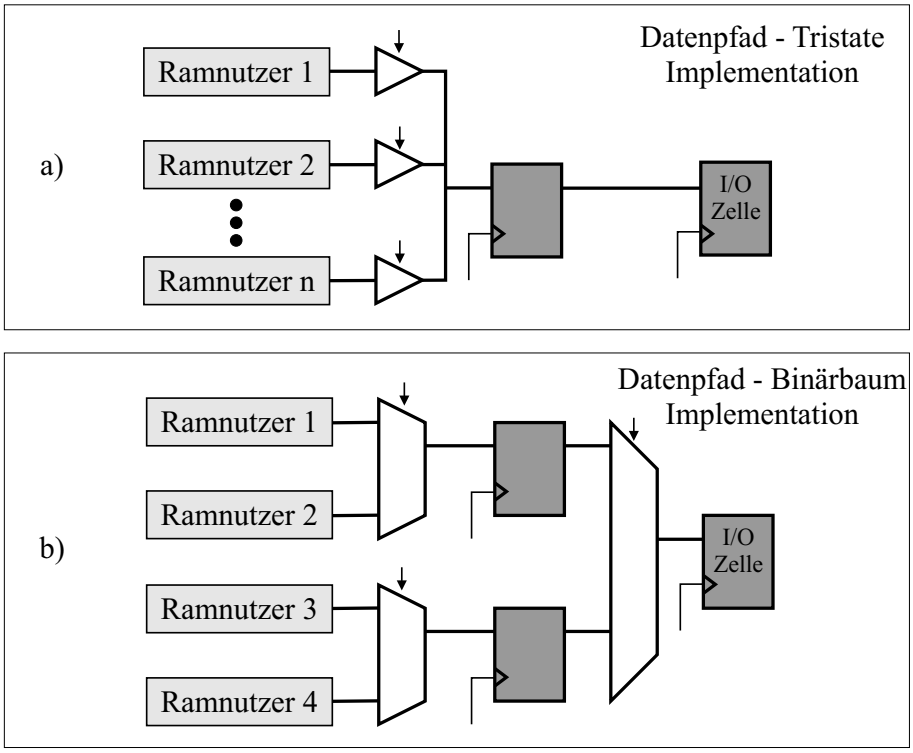

Abbildung 3.3: Datenpfad von den Ramklienten bis in die I/O-Zellen des FPGAs
a) Realisierung mit langsamen, aber ressourcensparenden Tristate-Treibern
b) Implementation mittels Multiplexern in einem Binärbaum
Die Beschaltung der Tristate-Treiber, bzw. der Multiplexer wird durch den Manager vorgenommen, er entscheidet, welcher Ramklient seine Daten in die I/O-Zelle schreibt, wie durch die senkrechten Pfeile angedeutet ist.

Datenweiche - Lesepfad Im Gegensatz zum Schreibdatenpfad müssen im Lesedatenpfad keine Signale zusammengefasst werden, die gelesenen Daten werden gleichzeitig an alle Ramklienten geführt. Zusätzlich erhält jeder Ramklient ein Signal, das anzeigt, ob die gelesenen Daten für ihn bestimmt sind. Ist dies der Fall, so werden die Daten in den Daten-FIFO dieses Ramklienten geschrieben, wo sie zwischengespeichert werden, bis der zugehörige Ramnutzer sie abholt.

Datenweiche - Adresspfad Der Adresspfad ist zeitlich weniger anspruchsvoll als die Datenpfade, weil die Adressen nur mit einfachem Takt aktualisiert werden, daher wird eine Implementation mit Tristate-Treibern verwendet (vgl. Abb. 3.4).

3.1.4 Ramkontrolleinheit

Die Ramkontrolleinheit kapselt die physikalische Schnittstelle und das komplexe Zugriffsschema (vgl. Abschnitt 1.3.2) zum externen Speicherbaustein und stellt dem Manager eine vereinfachte interne Schnittstelle zur Verfügung: Der Manager kann synchron das Bereitschaftssignal abfragen und bei aktiver Bereitschaft genau einen Speicherzugriff an die

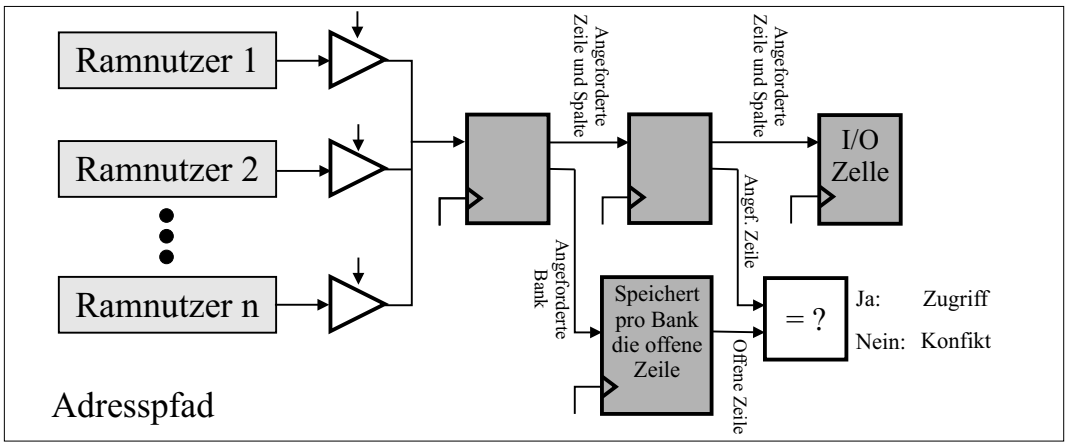

Abbildung 3.4: Adresspfad von den Ramklienten in die I/O-Zellen des FPGA. Da die Adressen nur mit einfacher Datenrate geschrieben werden, ist eine Implementation mit Tristate-Treibern ausreichend schnell. Ein Teil der Logik zur Abfrage eines Zeilenkonflikts ist ebenfalls skizziert (siehe Abschnitt 3.1.4): Ein Konflikt tritt dann auf, wenn die angeforderte Zeile und die aktuell geöffnete Zeile ungleich sind.

Ramkontrolleinheit übergeben. Falls keine Schreib-Lese- oder Zeilenkonflikte auftreten, wird dieser Speicherzugriff direkt ausgeführt, das Bereitschaftssignal bleibt aktiv und der nächste Zugriff kann direkt im nächsten Takt übergeben werden.

Bei Konflikten sichert die Ramkontrolleinheit die Zugriffsanforderungen, löscht das Bereitschaftssignal, löst die Konflikte, führt den Zugriff aus und setzt das Bereitschaftssignal wieder auf aktiv. Tab. 3.1 gibt die Art der möglichen Konflikte, die nötigen Schritte zur Beseitigung und die Anzahl der dafür benötigten Takte an.

Als Beispiel ist in Abb. 3.4 die Logik zur Abfrage eines Zeilenkonflikts skizziert: Pro Bank wird die in dieser Bank geöffnete Zeile in der Ramkontrolleinheit gespeichert. Stimmen die angeforderte Zeile und die aktuell geöffnete Zeile überein, so kann der Zugriff direkt ausgeführt werden. Wird allerdings eine Zeile angefordert, die nicht offen ist, so tritt ein Konflikt auf, es kann in diesem Takt kein Zugriff auf den externen Speicher stattfinden.

Die zeitliche Abfolge der Signale auf der Schnittstelle zwischen FPGA und dem externen Speicherbaustein ist durch den Hersteller der Speicherbausteine vorgegeben [48] [34]. Der Datentransfer erfolgt bei Schreibzugriffen einen Takt und bei Lesezugriffen 2,5 Takte nach dem eigentlichen Befehlstakt.

Physikalische Schnittstelle

Der externe DDR-SDRAM Baustein bezieht seinen Takt vom FPGA, dieser Takt wird im Folgenden *RAM-Versorgungstakt* genannt. Zur Untersuchung der Schnittstelle zwischen FPGA und dem DDR-SDRAM Baustein kann man drei Arten von Übertragungen unterscheiden:

Konflikt	Lösung	Benötigte Taktzyklen
Lesezugriff direkt nach Schreibzugriff	Wartezyklen einfügen	2
Schreibzugriff direkt nach Lesezugriff	Wartezyklen einfügen	3
Keine Zeile geöffnet	Richtige Zeile öffnen	6
Falsche Zeile nach Lesezugriff offen	Schließen, richtige Zeile öffnen	7
Falsche Zeile nach Schreibzugriff offen	Schließen, richtige Zeile öffnen	10
Auffrisch-Zyklus fällig	Auffrischen, richtige Zeile öffnen	23

Tabelle 3.1: Mögliche Konflikte beim Zugriff auf das DDR-SDRAM, Maßnahmen zur Behebung und Anzahl der dafür notwendigen Takte, in denen kein Datentransfer stattfinden kann

1. Das Schreiben der Adressen und Befehle zum DDR-SDRAM erfolgt mit dem einfachen Takt. Im SDRAM werden diese Signale mit dem RAM-Versorgungstakt registriert. Um eine stabile Übertragung zu gewährleisten, müssen die Signale der Adressen und Befehle zur steigenden Taktflanke des RAM-Versorgungstaktes stabil sein.

2. Das Schreiben von Daten zum DDR-SDRAM geschieht mit doppelter Datenrate (DDR), d.h. es werden sowohl mit der steigenden, als auch mit der fallenden Taktflanke Daten übertragen. Das DDR-SDRAM Protokoll legt fest, dass für jeweils 8 Datenleitungen ein zusätzliches Taktsignal (DQS) übermittelt wird. Die DQS-Signale werden verwendet, um die Schreibdaten im Speicherbaustein 8-bitweise zu registrieren. Dabei müssen beim Schreiben zum DDR-SDRAM die Datensignale und das DQS-Signal eine Phasenverschiebung von 90° aufweisen. Das macht den Entwurf der Leiterplatte einfacher, es müssen nur jeweils 8 Datenleitungen und die zugehörige DQS-Leitung die gleiche Signallaufzeit zwischen FPGA und SDRAM aufweisen. Da die DQS-Signale zum Registrieren der Daten verwendet werden, ist die Phasenlage des RAM-Versorgungstaktes in einem Bereich von 180° relativ zu den DQS-Signalen bzw. den Datensignalen beliebig.

3. Die Lesedaten, die der DDR-SDRAM Baustein an den FPGA sendet, werden ebenfalls mit doppelter Datenrate getaktet, sie müssen mit einem geeigneten Takt im FPGA registriert werden. Auch das DDR-SDRAM stellt – entsprechend der Leserichtung – DQS-Signale zur Verfügung. Allerdings können diese DQS-Signale im FPGA nicht verwendet werden, da einerseits zu wenige globale Taktnetze zur Verfügung stehen und andererseits die lokalen Taktnetze aufgrund des vorliegen Leiterplattenlayouts nicht verwendet werden können[3].

[3]Grundsätzlich ermöglicht der Virtex-2 Pro eine lokale Taktung der I/O-Zellen, allerdings muss dieses

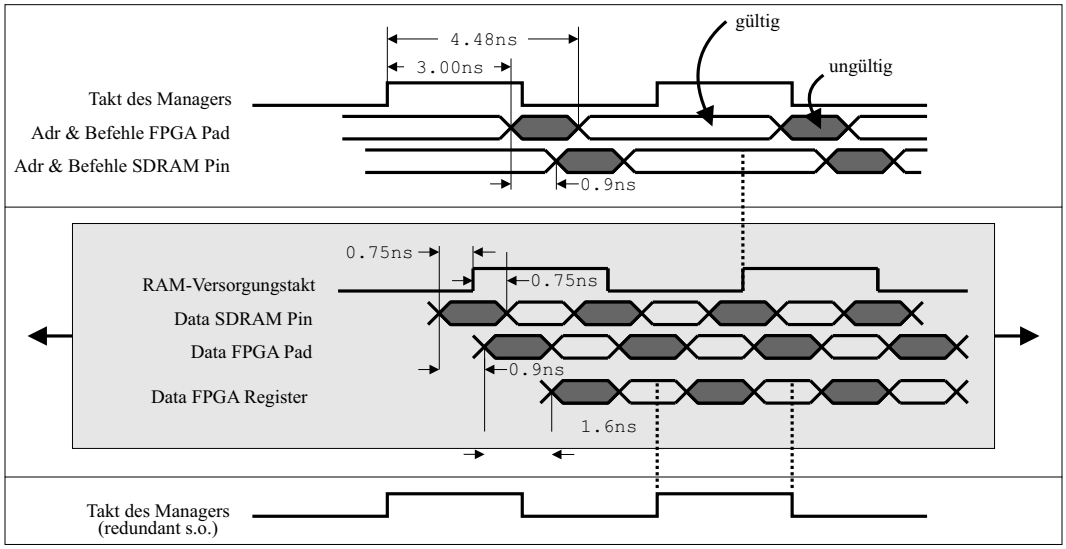

Abbildung 3.5: Zur Veranschaulichung der Synchronisation der physikalischen Schnittstelle zwischen FPGA und DDR-SDRAM: Im oberen Kasten sind die Adressen und Befehle dargestellt, die einerseits durch FPGA-interne Signallaufzeiten und die Laufzeiten auf der Leiterplatte verzögert werden. Im grau unterlegten Kasten ist der Weg der Lesedaten von den Pins des DDR-SDRAM-Bausteins bis zu den ersten Registern im FPGA gezeigt. Durch eine Verschiebung der Phase des RAM-Versorgungstakt kann der gesamte grau unterlegte Kasten verschoben werden. Zuverlässiger Datentransfer kommt zustande, wenn sowohl die Adressen und Befehle im SDRAM durch den RAM-Versorgungstakt als auch die Daten im FPGA durch den Takt des Managers registriert werden können. Dies ist durch die senkrechten gepunkteten Linien angedeutet. Die angegebenen Signallaufzeiten entstammen den Synthese-Werkzeugen des FPGA-Herstellers, den Datenblättern des Speicherherstellers [48] [49], bzw. Simulationen der Laufzeiten auf der Leiterplatte [25].

Idealerweise werden zwei DCMs zur Synchronisation der physikalischen Schnittstelle verwendet: Ein DCM verschiebt den RAM-Versorgungstaktes gegenüber den Adressen und Befehlen, dadurch kann die unter (1) beschriebene Übertragung synchronisiert werden. Danach wird mit einem weiteren DCM der Takt, mit dem die Lesedaten im FPGA registriert werden, verschoben. Damit kann die unter (3) beschriebene Übertragung synchronisiert werden. Da die Schreibdaten im DDR-SDRAM durch die zusätzlichen DQS-Signale getaktet werden, benötigt die unter (2) beschriebene Übertragung keine weiteren Taktsignale. Für den Betrieb des DDR-SDRAMs ist im Nathan Netzwerkmodul aber nur ein DCM vorgesehen, es musste also eine andere Lösung gefunden werden.

Mit dem einen verfügbaren DCM wird der RAM-Versorgungstakt verschoben, die gesamte Logik innerhalb des FPGAs wird mit dem internen Takt des Managers betrieben. Dieses Verhalten ist in Abb. 3.5 dargestellt. Durch die Verschiebung des RAM-Versorgungstaktes wird die gesamte Taktung der DDR-SDRAM-Bausteine verschoben,

Taktsignal über spezielle I/O-Pins in den FPGA eingespeist werden. Die Leitungen der DQS-Signale der vorliegenden Nathan Leiterplatte sind nicht an diese speziellen I/O-Pins angeschlossen.

also auch die Taktung der Lesedaten, die an den FPGA gesendet werden. In Abb. 3.5 kann daher der gesamte grau eingefärbte Bereich verschoben werden, bis die Lesedaten, die durch Signallaufzeiten auf der Leiterplatte (0,9 ns) und im FPGA (1,6 ns) verzögert sind, im FPGA mit dem Takt des Managers zuverlässig registriert werden können. Diese Beziehung ist durch gepunktete senkrechte Linien angedeutet.

Gleichzeitig beeinflusst die Verschiebung des RAM-Versorgungstaktes aber auch das Registrieren der Adressen und Befehle im DDR-SDRAM. Aus der Abb. 3.5 wird aber deutlich, dass die Adressen und Befehle am SDRAM Pin über einen großen Bereich gültig und daher flexibel gegenüber der Verschiebung des RAM-Versorgungstaktes sind, auch diese Beziehung ist durch eine gepunktete senkrechte Linien angedeutet.

Da also die Befehle und Adressen unabhängig von der Verschiebung des RAM-Versorgungstaktes sind, kann diese Verschiebung so gewählt werden, dass die Lesedaten im FPGA zur Taktflanke des internen Takts des Managers gültig sind.

Um die Korrektheit dieser Überlegungen und Abschätzungen und damit die Zuverlässigkeit der Schnittstelle zu testen und zu überprüfen, wurde ein Datenvolumen von 128 GByte bei verschiedenen Frequenzen und unterschiedlichen Verschiebungen des RAM-Versorgungstaktes übertragen. Dieser Test wurde bei Taktfrequenzen des Managers von 128 MHz, 136 MHz, 144 MHz und 152 MHz[4] auf den ersten 8 Nathan Netzwerkmodulen einer Backplane parallel ausgeführt. Die Daten werden dabei mit doppelter Datenrate, also 256 Mbit/s bis 304 Mbit/s pro Signalleitung getaktet. Das verwendete Design enthält 4 Ramklienten, die jeweils einen Block zufälliger Größe erst schreiben, und im Anschluss daran lesen und verifizieren. Jeder Ramklient verwendet dafür jeweils 2 der insgesamt 8 Speicherbänke des externen SDRAM-Bausteins. Die Ergebnisse sind in Abb. 3.6 dargestellt, wobei die schwarzen Kästen einen fehlerfreien Datentransfer anzeigen, im Falle der hellen Kästen traten Fehler auf. Da jeweils 128 GByte übertragen wurden, steht ein schwarzer Kasten für eine Byte-Fehlerrate unter $7,5 \cdot 10^{-12}$. Wie zu erwarten ist, gibt es leichte, bauteilbedingte Schwankungen zwischen den einzelnen Nathan Netzwerkmodulen, zudem wird der Bereich, in dem fehlerloser Datentransfer möglich ist, mit steigender Frequenz kleiner.

3.1.5 Verifikation und Leistungsdaten

Zur Verifikation und zur Bestimmung der Leistungsfähigkeit der Speicheransteuerung wird ein Aufbau verwendet, der den Anforderungen der Trainingsalgorithmen ähnelt (vgl. 4). Es werden bis zu 4 Ramklienten, die sowohl Schreiben als auch Lesen können und bis zu 4 reine Lese-Ramklienten parallel verwendet. Dabei werden die Datenpfade mit Multiplexern in einer Binärbaumstruktur implementiert, um maximale Geschwindigkeit zu erreichen.

Ein vorgegebener Speicherbereich wird gleichmäßig auf die vorhandenen Schreib-Lese-Ramklienten aufgeteilt, danach beginnt jeder dieser Ramklienten damit, einen Block zufälliger Größe in diesen reservierten Speicher zu schreiben und wieder zu lesen. Sobald ein Schreib-Lese-Ramklient diesen Vorgang abgeschlossen hat, liest ein reiner Lese-Ramklient (soweit vorhanden) diesen Block erneut. Traten bei beiden Lesevorgängen keine

[4]Die Synthesewerkzeuge des FPGA-Herstellers geben eine maximale Frequenz von 140 MHz bei worst-case Bedingungen an. Die FPGA-interne Logik und die physikalische Schnittstelle konnten aber auch bei höheren Frequenzen (144 bzw. 152 MHz) zuverlässig betrieben werden.

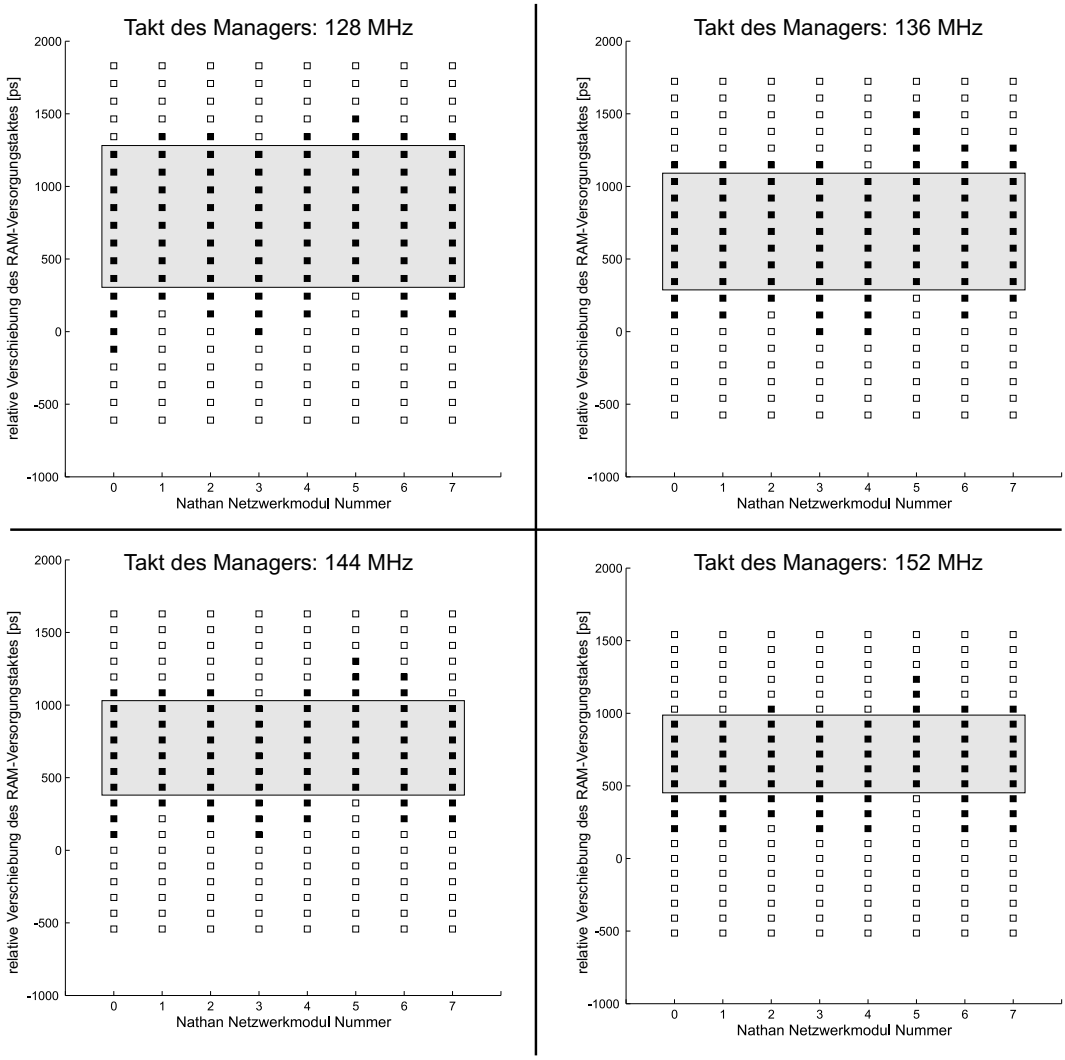

Abbildung 3.6: Untersuchung der physikalischen Schnittstelle zwischen FPGA und DDR-SDRAM auf den ersten 8 Nathan Netzwerkmodulen einer Backplane. Bei dem jeweils angegebenen Takt des Managers wurde ein Datenvolumen von 128 GByte bei jeweils unterschiedlicher Verschiebung des RAM-Versorgungstaktes geschrieben und gelesen. Die schwarzen Kästen zeigen eine fehlerfreie Übertragung, im Falle der weißen Kästen traten Fehler auf. Der fehlerfreie Bereich wird mit steigender Frequenz kleiner, zudem zeigen sich bauteilbedingte Schwankungen zwischen den einzelnen Nathan Netzwerkmodulen.

Ramnutzer Konfiguration	maximale Frequenz [MHz]	exklusive Bank Auslastung [%]	exklusive Bank Rate [GB/s]	geteilte Bank Auslastung [%]	geteilte Bank Rate [GB/s]
4 Schreib-Leseklienten	140	$(86,13 \pm 0,01)$	1,797	$(63,25 \pm 0,01)$	1,319
2 Schreib-Leseklienten	140	$(83,58 \pm 0,66)$	1,744	$(60,86 \pm 0,62)$	1,270
2 Schreib-Leseklienten	140	$(83,58 \pm 0,66)$	1,744	$(60,86 \pm 0,62)$	1,270
4 Schreib-Leseklienten und 4 Leseklienten	120	$(86,63 \pm 1,96)$	1,549	$(59,69 \pm 2,45)$	1,067

Tabelle 3.2: Maximale Frequenz und dabei erreichte Auslastung und Datenrate in GByte/s bei verschiedenen Ramnutzer-Konfigurationen.

Fehler auf, so wiederholt sich der Vorgang mit einem neuen Block zufälliger Größe. Der vorgegebene Speicherbereich wird dabei variiert: Die erste Testkonfiguration vergibt den gesamten vorhandenen Speicher, in diesem Falle benutzt jeder Ramklient eine oder mehrere Speicherbänke des SDRAM exklusiv. Die zweite Konfiguration vergibt insgesamt nur eine Speicherbank, diese teilen sich die vorhandenen Ramklienten. Da es dadurch vermehrt zu Zeilenkonflikten kommt, sinkt die erreichte Datenrate.
Die erreichbare Übertragungsrate der Speicheransteuerung hängt von zwei Faktoren ab:

- Der erste Faktor ist die maximale Taktrate, mit der die Logik und die Datensignale von Ramnutzer über Ramklienten, Manager und Ramkontrolleinheit zum externen SDRAM getaktet werden können. Diese maximal erreichbare Frequenz, gibt einen Wert für die maximale Datenrate. Diese Datenrate würde erreicht werden, wenn in jedem Takt Daten übertragen werden könnten.

- Durch die Eigenschaften des externen SDRAM können allerdings nicht in jedem Takt Daten übertragen werden. Je nach Zugriffsmuster geht ein Teil der Bandbreite verloren, weil Zeilen gewechselt oder Auffrischzyklen eingefügt werden müssen. Die Art der möglichen Konflikte wurde bereits in Tab. 3.1 dargestellt. Die Auslastung, d.h. der Prozentsatz der Takte, in denen effektiv Daten übertragen werden, ist also abhängig vom Zugriffsmuster.

Die maximal erreichbare Übertragungsrate unter verschiedenen Bedingungen ist in Tabelle 3.2 dargestellt. Die maximal erreichbare Taktfrequenz für den Takt des Managers ist 140 MHz, wenn 2 oder 4 Ramklienten verwendet werden und sie sinkt durch den erhöhten Aufwand der Datenweiche und der Arbitrierung im Manager bei Benutzung von 8 Ramklienten auf 120 MHz. Die Daten werden dabei mit doppelter Datenrate geschrieben und gelesen, also mit 280 MHz bzw. 240 MHz getaktet. Die erreichte Auslastung liegt bei 86%, wenn jeder Ramklient nur exklusiv auf eigene Speicherbänke zugreift und bei etwa 63%, wenn sich alle Ramklienten eine Speicherbank teilen. Im ersten Fall geht hauptsächlich durch Schreib-Lese-Wechsel und Auffrischzyklen Bandbreite verloren. Wenn sich aber alle

Ramklient	FFs	LUTs	Blockrams
Schreiben und Lesen - synchron	127	151	4
Schreiben und Lesen - asynchron	161	193	4
Reiner Leseklient - synchron	86	115	2
Reiner Schreibklient - synchron	88	115	2
Manager	**FF**	**LUT**	**Tristates**
4 Schreib-Leseklienten Baumstruktur	357	414	108
4 Schreib-Leseklienten Tristate-Bus	280	198	376
6 Schreib-Leseklienten Tristate-Bus	337	346	546
Ramkontrolleinheit	**FF**	**LUT**	
	417	400	

Tabelle 3.3: Ressourcenverbrauch der einzelnen Objekte der Speicheransteuerung

Ramklienten eine Speicherbank teilen, so muss zusätzlich häufig die geöffnete Zeile gewechselt werden, die Auslastung sinkt[5].
Für jede Messung der Tabelle 3.2 wurde ein Datenvolumen von 256 GByte auf den ersten 8 Nathan Netzwerkmodulen geschrieben und gelesen, im Falle der reinen Leseklienten noch einmal erneut gelesen. Insgesamt liegt die Byte-Fehlerrate folglich unter $5 \cdot 10^{-13}$.

3.1.6 Ressourcenverbrauch

Der Ressourcenverbrauch der einzelnen Module ist in Tab. 3.3 angegeben. Um den Ressourcenverbrauch der einzelnen Objekte zu ermitteln, wurden sie einzeln synthetisiert, die Zahlen wurden mit der Xilinx Software nach Synthese und Mapping ermittelt (vgl. Abschnitt 1.3.3).
Eine Speicheransteuerung, die vier synchrone Ramklienten verwendet, die sowohl Lesen als auch Schreiben können, zusammen mit dem geschwindigkeitsoptimierten Manager mit Baumstruktur und der Ramkontrolleinheit benötigt demnach 1282 FFs und 1418 LUTs. Im verwendeten FPGA sind das etwa 11,6 % rsp. 12,8% der verfügbaren Ressourcen.

3.1.7 Portabilität

Im Rahmen dieser Arbeit wurden insgesamt vier Schnittstellen zu den unterschiedlichen externen Speicherbausteinen SRAM und SDRAM auf dem Darkwing- bzw. Nathan-System entwickelt. Die hiervon leistungsfähigste und anspruchsvollste Schnittstelle zum DDR-SDRAMs im Nathan-System wurde als Beispiel im Detail beschrieben. Idealerweise kapselt die Ramkontrolleinheit die Eigenschaften der externen Speicherschnittstelle, so dass Manager und Ramklient für alle externen Speicherbausteine ohne Änderungen verwendet

[5] Die Auslastung im Falle von 2 Ramklienten ist niedriger als bei der Verwendung von 4 Ramklienten, zudem ist der Wert mit einem größeren Fehler behaftet. Dieser Umstand ist durch den Testaufbau erklärbar: Nach Beendigung eines Transfers wird der nächste Transfer nicht sofort erneut gestartet, zwischen dem Ende des einen und dem Start des nächsten Transfers ist der entsprechende Ramnutzer eine Zeitlang inaktiv. Da zur vollständigen Auslastung aber 2 kontinuierlich aktive Ramnutzer nötig sind, sinkt die gemessene Datentransferrate im Falle von nur 2 Ramklienten durch die kurze Inaktivität eines Ramklienten.

werden können. Da allerdings das SDRAM im Nathan-System mit doppelter Datenrate arbeitet, hier also bei jedem Zugriff nicht 64 Bit, sondern 128 Bit übertragen werden, mussten die Datenpfade des Managers und der Ramklienten darauf angepasst werden.

Wiederum aufgrund der doppelten Datenrate des DDR-SDRAM ist die Schnittstelle zwischen Ramklient und Ramnutzer nicht vollständig unabhängig vom verwendeten Speicherbaustein. Da das Nathan SDRAM pro Zugriff immer 128 Bit überträgt, müssen hier die 64 Bit breiten Zugriffe zum Ramnutzer immer in Paaren auftreten. Die genaue Definition des Zugriffsprotokolls findet sich in Anhang A.

3.1.8 Alternativen

Der FPGA-Hersteller Xilinx stellt mit dem *Synthesizable 400Mb/s DDR SDRAM Controller* [83] eine DDR-SDRAM Schnittstelle zur Verfügung. Der Controller von Xilinx versorgt die physikalische externe Schnittstelle zum DDR-SDRAM und bietet eine vereinfachte FPGA-interne Schnittstelle an. Dabei wird eine Taktfrequenz von 200 MHz anvisiert, die mit doppelter Datenrate übertragenen Daten würden damit 400 Mbit/s erreichen. Da dieser Controller keine Möglichkeit bietet, simultan mehrere logische Speicherschnittstellen zu nutzen, ist die Funktionalität mit der in Abschnitt 3.1.4 vorgestellte Ramkontrolleinheit vergleichbar.

Um diese hohe Geschwindigkeit zu erreichen, werden die Daten in den DDR-Registern der I/O-Zellen von 400 Mbit/s auf einen internen Takt von 200 MHz umgewandelt, dabei wird die Busbreite verdoppelt. Der weitere Datenpfad wird daraufhin mit der Frequenz von 200 MHz getaktet. Die Umwandlung auf die doppelte Busbreite bei einer Halbierung des Taktes vereinfacht das Routing und ermöglicht die hohe Frequenz auf der Schnittstelle von 400 Mb/s, verbraucht aber viele Ressourcen: Der DDR-SDRAM Controller von Xilinx benötigt für eine externe Datenbreite von 32 Bit 1359 FFs und 936 LUTs[6].

Im Nathan-System können weder die DDR-Register in den I/O-Zellen verwendet werden, noch stehen die Logikressourcen für eine interne Verdoppelung der Busbreite zur Verfügung (vgl. Abschnitt 3.1.1). Trotzdem kann die Speicheransteuerung bei 8 Ramnutzern mit einer Taktfrequenz von 120 MHz, bei 4 Ramnutzern mit einem Taktfrequenz von 140 MHz betrieben werden. Die Daten werden dabei von den FIFO-Speichern der Ramklienten bis in die IO-Zellen mit 240 MHz bzw. 280 MHz innerhalb des FPGAs getaktet. Der Ressourcenverbrauch ist deutlich niedriger: Obwohl die Ramkontrolleinheit eine externe Datenbreite von 64 Bit verarbeitet, benötigt sie nur 417 FFs und 400 LUTs. Für die gesamte Speicheransteuerung mit 4 synchronen Schreib-Lese-Ramklienten, d.h. 8 unabhängigen, logischen Speicherschnittstellen, werden 1282 FFs, 1418 LUTs und 16 Blockrams benötigt.

[6] Um diese Zahlen zu erreichen wurde der DDR-SDRAM Controller mit den vorhandenen Synthesewerkzeugen untersucht, Grundlage war die von Xilinx für die Synthese bereitgestellte Netzliste im EDIF-Format.

3.2 Register- und Kontrollzugriff: Slow-Control

Nachdem in Abschnitt 3.1 die Speicheransteuerung als wichtigster Teil Infrastruktur innerhalb des FPGAs beschrieben wurde, wird in diesem Abschnitt der Register- und Kontrollzugriff, im Folgenden *Slow-Control* genannt, vorgestellt. Die Slow-Control ist ähnlich aufgebaut wie die Speicheransteuerung: An einen zentralen Manager können beliebig viele *Slow-Control-Klienten* angeschlossen werden. Da die Slow-Control entwickelt wurde, um einzelne Statusregister zu lesen bzw. zu schreiben, ist die Datenrate und Geschwindigkeit im Gegensatz zur Speicheransteuerung nicht wichtig. An dieser Stelle wird nur eine kurze Übersicht des Aufbaus und der Eigenschaften der Slow-Control gegeben, für eine detaillierte Beschreibung sei auf [55] verwiesen.

3.2.1 Aufgaben und Organisation der Slow-Control

Über die Slow-Control wird einerseits ein langsamer und daher ressourcenschonender Zugriff auf jede im FPGA realisierte Komponente zur Steuerung und Diagnose ermöglicht, andererseits wird die Slow-Control aber auch verwendet, um die einzelnen Nathan Netzwerkmodule vom Kontroll-PC aus anzusprechen.

Abb. 3.7 zeigt den Aufbau im gesamten Nathan-System: Die Slow-Control auf jedem Nathan Netzwerkmodul verbindet eine beliebige Anzahl an Klienten, maximal einen Master und eine Schnittstelle zur Backplane. Auf der Backplane ist ein Token Ring Netzwerk realisiert, darüber kann der Kontroll-PC auf die Slow-Control jedes Nathan Netzwerkmodules, und damit auf jeden Slow-Control-Klienten zugreifen. Zusätzlich kann pro Nathan Netzwerkmodul maximal ein Master an die Slow-Control angeschlossen werden, dieser Master kann auf alle lokalen Klienten dieses FPGAs zugreifen. Im vorgestellten Design ist der eingebettete PowerPC als Slow-Control-Master angeschlossen und kann damit sämtliche lokalen Klienten ansprechen.

Im Darkwing-System wird die Slow-Control über den PCI-Bus angesprochen. Damit hat der Kontroll-PC Zugriff auf alle Slow-Control-Klienten auf der Darkwing-Karte (vgl. Abschnitt 3.5).

Anzahl Klienten	FFs	LUTs	Tristates
1	660	805	65
2	699	815	98
3	738	826	131
4	777	834	164
5	819	843	197
6	855	851	230
7	894	859	263
8	933	867	296

Tabelle 3.4: Ressourcenverbrauch der Slow-Control-Ansteuerung

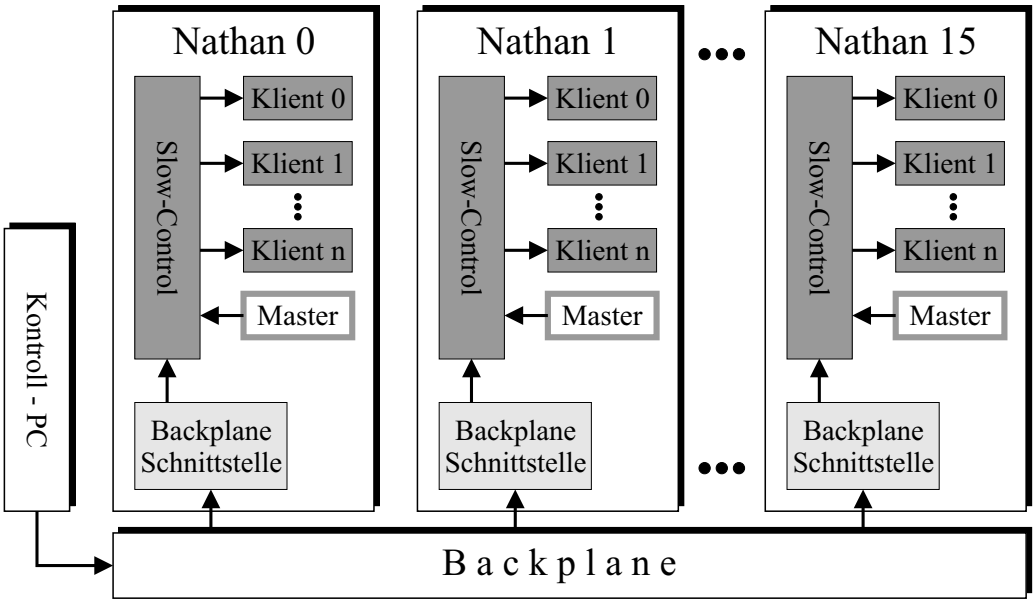

Abbildung 3.7: Aufbau der Slow-Control im Nathan-System. Der Kontroll-PC kann über ein Token Ring Netzwerk über die Backplane jeden Slow-Control-Klienten auf jedem Nathan Netzwerkmodul ansprechen. Der Slow-Control-Master auf jedem Nathan Netzwerkmodul hat dagegen nur Zugriff auf die lokalen Klienten. Die Pfeile deuten jeweils an, in welche Richtung ein Zugriff initiiert werden kann.

3.2.2 Implementation

Jeder Slow-Control-Klient verfügt über einen 32 Bit breiten Adressraum, der Datenbus ist ebenfalls 32 Bit breit ausgelegt. Die Kommunikation zum Kontroll-PC verläuft über eine serielle Verbindung, die im Frequenzbreich von 40-80 MHz betrieben werden kann. Die Slow-Control-Ansteuerung ist möglichst einfach gehalten, damit wenig Ressourcen im FPGA verbraucht werden, daher werden nur Einzelzugriffe mit jeweils 32 Bit Daten unterstützt. Die erreichte Datenrate beim Zugriff vom Kontroll-PC auf ein Nathan Netzwerkmodul bei einer Frequenz des Token Ring Netzwerkes von 78 MHz ist 1,38 MByte/s bei Schreibzugriffen und 0,83 MByte/s bei Lesezugriffen. Den Ressourcenverbrauch der Slow-Control-Ansteuerung gibt Tab. 3.4 wieder.

Das vollständige Design verwendet in jedem Nathan fünf Slow-Control-Klienten[7] und belegt demnach 819 FFs und 843 LUTs, also etwa 7,6 % der gesamten Logikressourcen des verwendeten FPGAs. Dabei ist allerdings zu beachten, dass die Slow-Control gegenwärtig überarbeitet wird, um den – verglichen mit der Funktion – hohen Ressourcenverbrauch zu senken. Für die Beschreibung der endgültigen Version sei erneut auf [55] verwiesen.

[7] Jeweils ein Slow-Control-Klient regelt den Zugriff auf das DDR-SDRAM, die Analog-Einheit, den evolutionären Koprozessor, die HAGEN-Ansteuerung und den PowerPC.

Komponente	FFs	LUTs
`sctrl_ramklient`	99	50
`sctrl_dac_tempsense`	61	198

Tabelle 3.5: Ressourcenverbrauch einzelner Slow-Control-Module.

3.2.3 Slow-Control Module

An dieser Stelle sollen kurz zwei Slow-Control Module beschrieben werden, die für die Infrastruktur innerhalb des FPGAs wichtig sind, den Ressourcenverbrauch dieser Module gibt Tab. 3.5 an. Um den externen Speicher an die Slow-Control anzuschließen, wird ein Slow-Control-Klient mit einem Ramklienten verbunden. Dadurch kann der lokale Slow-Control Master und auch der Kontroll-PC auf den externen Speicherbaustein des Nathan Netzwerkmodules zugreifen. Dieser Pfad wird z.B. genutzt, um während der Initialisierung Daten und Betriebssystem vom Kontroll-PC zu jedem Nathan Netzwerkmodule zu übertragen.

Auf jeder Nathan-Platine befindet sich auch ein DAC-Baustein [43], der verwendet wird, um den Netzwerkchip mit Referenzspannungen zu versorgen. Ebenso steht ein Temperatursensor [44] zur Verfügung, mit dem einerseits die Betriebstemperatur des FPGAs und auch die des Netzwerkchips überwacht werden kann. Sowohl der DAC als auch der Temperatursensor werden über eine gemeinsame Schnittstelle und über einen Slow-Control-Klienten gesteuert. Dadurch hat sowohl der Kontroll-PC als auch der lokale Slow-Control Master Zugriff auf den DAC und die Betriebstemperaturen.

3.3 Anbindung der Netzwerk Chips: HAGEN-Ansteuerung

Sowohl das Darkwing- als auch das Nathan-System wurden mit dem Ziel entwickelt, analoge neuronale Netzwerkchips anzusprechen und zu trainieren. In diesem Abschnitt wird die Schnittstelle zum Netzwerkchip HAGEN beschrieben und deren Leistungsfähigkeit untersucht. Da neben der physikalischen externen Schnittstelle zum HAGEN Chip nur die bestehende Infrastruktur verwendet wird, lässt sich die HAGEN-Ansteuerung nahtlos in ein bestehendes Design einbinden.

3.3.1 Überblick

Der HAGEN Chip wird in zwei Phasen angesteuert. In der *Konfigurationsphase* werden die Gewichte des Netzwerkes geschrieben, d.h. eine Ladung proportional zum gewünschten Gewicht auf die Speicherkondensatoren der Synapsen aufgebracht (vgl. Abschnitt 1.1.3). Da die dazu nötigen DACs innerhalb des HAGEN Chip integriert sind, müssen nur digitale Werte an den HAGEN Chip gesendet werden. Nach der Konfigurationsphase schließt sich die *Netzwerk-Betriebsphase* an: Ein Eingabevektor umfasst die Werte der 512 binären Eingabeneuronen und wird digital an den HAGEN Chip gesendet. Daraufhin werden ein oder mehrere analoge Netzwerkzyklen durchgeführt und schließlich die Ergebnisse dieser Berechnungen, die Werte der 256 binären Ausgabeneuronen, ausgelesen. Dieser Vorgang wird in der Regel für weitere Eingabe- bzw. Ausgabevektoren wiederholt. Die HAGEN-Ansteuerung steuert sowohl die Konfigurationsphase als auch die Betriebsphase. Der Aufbau ist in Abb. 3.8 dargestellt. Es müssen drei Datenströme verwaltet werden: Während der Konfiguration die *Gewichtsdaten* und während des Netzwerkbetriebs die *Eingabe-* und die *Ergebnisdaten*. Da die einzelnen Daten für eine Speicherung im FPGA zu umfangreich sind[8], werden sie im externen Speicherbaustein abgelegt und mit Hilfe von drei Ramklienten von der Speicheransteuerung gelesen, bzw. dorthin geschrieben.

Aufgrund des internen Aufbaus des HAGEN Chips können diese drei Datenströme aber nicht im Sinne der Genanordnung an die Schnittstelle und dann zum HAGEN Chip gesendet werden, sondern sie müssen umsortiert werden. Diese Umrechnung erfolgt in der programmierbaren Logik durch spezielle *Sortierer*, die direkt an die Ramklienten angeschlossen werden.

Die physikalische Schnittstelle zwischen FPGA und dem HAGEN Chip überträgt Daten mit steigender und fallender Taktflanke (DDR) und einer Datenbreite von 16 Bit, die Schnittstelle zu den Ramklienten erfordert aber gemäß Abschnitt 3.1 eine Datenbreite von 64 Bit. Der *Datenpfad* führt diese Umstrukturierung von den langsameren, breiten internen Datenleitungen auf die schmaleren Datenleitungen der physikalischen Schnittstelle aus.

Die *Ablaufsteuerung* besteht im wesentlichen aus einem Zustandsautomaten, der den Datenpfad steuert und Befehle, Testvektoren sowie die Daten an den HAGEN Chip sendet. Die Ablaufsteuerung wird durch einen *Slow-Control-Klienten* kontrolliert, über den die *Statusregister* geschrieben bzw. gelesen werden. Im Folgenden werden die einzelnen Teile der HAGEN-Ansteuerung und ihre Funktionsweise erläutert.

[8] Die Synpasengewichte zur Konfiguration des kompletten HAGEN Chips benötigen 64 kByte Speicherplatz, die Eingabedaten 64 Byte, sowie die Ausgabedaten 32 Byte pro Testvektor. Typische Anwendungen verwenden mindestens 1000 Testvektoren.

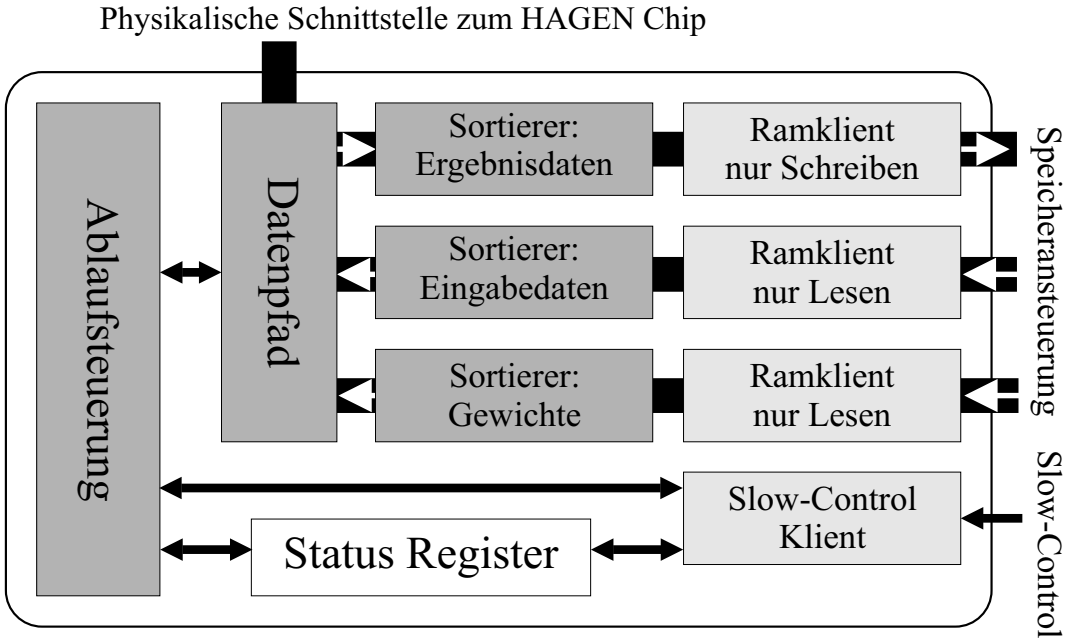

Abbildung 3.8: Aufbau der HAGEN-Ansteuerung. Neben der physikalischen Schnittstelle zum externen HAGEN Chip wird nur die bereits beschriebene Infrastruktur innerhalb des FPGAs benutzt: Die Daten werden über Ramklienten mit der Speicheransteuerung ausgetauscht, die Zugriff auf die Status Register und die Ablaufsteuerung wird über einen Slow-Control-Klienten realisiert.

3.3.2 Datenpfad und physikalische Schnittstelle

Die Datenpfade und die physikalische Schnittstelle von den Sortierern der Eingabe-, Ergebnis- und Gewichtsdaten bis zum HAGEN Chip sind in Abb. 3.9 dargestellt. Die Logik der Ablaufsteuerung und die Schnittstelle zu den Sortierern wird mit einer Taktrate von 78,125 MHz betrieben, die Datenbreite zu den Sortierern beträgt 64 Bit. Die physikalische Schnittstelle zwischen FPGA und HAGEN Chip wird allerdings mit der doppelten Frequenz, d.h. mit 156,25 MHz betrieben. Da die Daten mit doppelter Datenrate wechseln, wird so ein Datentransfer von 312,5 Mbit/s pro Datenleitung erreicht. Der *Datenpfad* übersetzt die Daten mit Hilfe von zwei Registerstufen von dem 64 Bit breiten Bus der Ramklienten auf den 16 Bit breiten Bus der physikalischen Schnittstelle. Da die Schnittstelle zu den Sortierern nur mit einem Viertel der Frequenz im Vergleich zur physikalischen Schnittstelle betrieben wird, sind die Datenraten trotz der unterschiedlich breiten Busse gleich.

Der HAGEN Chip besteht aus zwei Hälften, die als *obere* und *untere* Hälfte unterschieden werden. Jede Hälfte enthält zwei Netzwerkblöcke. Prinzipiell können die obere und die untere Hälfte getrennt betrieben werden. Jede der beiden Hälften verfügt über eine quellensynchrone[9] physikalische Schnittstelle mit den folgenden Signalleitungen (vgl.

[9]Eine quellensynchrone Taktung liegt vor, wenn die Schaltung, die die Daten erzeugt, gleichzeitig auch einen eigenen Takt bereitstellt, der mit den Daten transferiert wird. Dadurch werden Takt und Daten in gleicher Weise erzeugt und auch transportiert, haben also das gleiche Zeitverhalten.

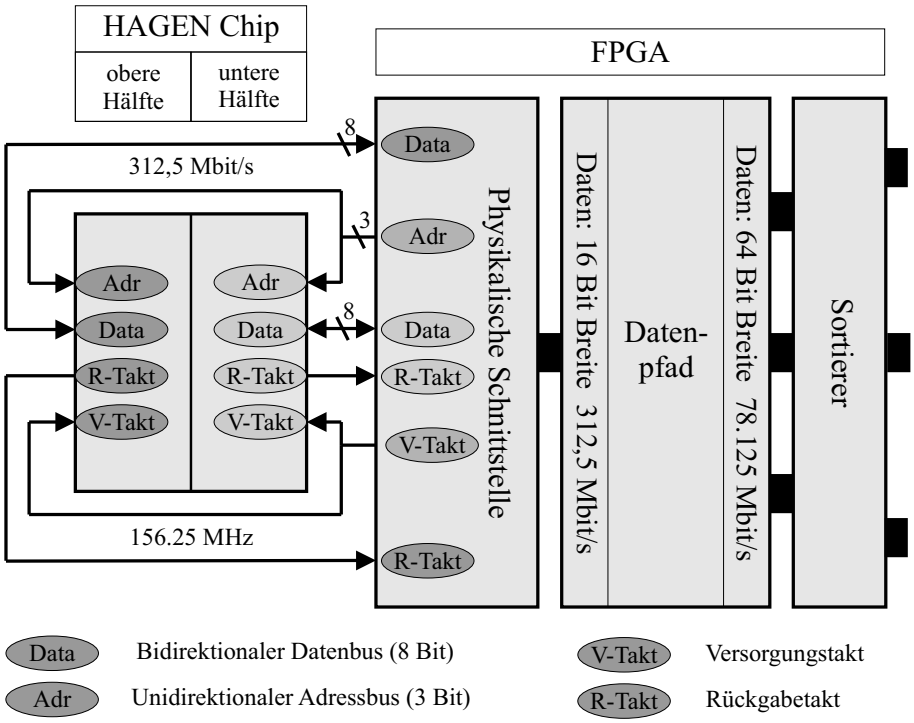

Abbildung 3.9: Schematische Darstellung der Datenpfade zwischen FPGA und HAGEN Chip. Die physikalische Schnittstelle besteht aus je 3 Adress-, 8 Daten- und 2 Taktleitungen pro Chiphälfte. Die *untere* Chiphälfte ist die Hälfte des HAGEN Chips, die dem FPGA zugewandt und mit kürzeren Signalleitungen auf der Platine angeschlossen ist. In der schematischen Darstellung befindet sich die *untere* Chiphälfte daher rechts. Die insgesamt 16 Datenleitungen werden durch den Datenpfad auf die Datenbreite und die Frequenz der Sortierer umgerechnet.

Abb. 3.9): 8 Datenleitungen pro Chiphälfte werden bidirektional verwendet. Eine Taktleitung, im Folgenden *Versorgungstakt* genannt und 3 Adressleitungen führen unidirektional zu jeder Hälfte des HAGEN Chips und je eine Taktleitung pro Chiphälfte, *Rückgabetakt* genannt, führt unidirektional vom HAGEN Chip zum FPGA (vgl. Abschnitt 1.1.3). Im Folgenden werden die beiden Richtungen des Datentransfers beschrieben, mit *Schreiben* ist die Richtung zum HAGEN Chip gemeint, das *Lesen* bezeichnet den Datentransfer in Richtung zum FPGA.

Schreiben zum HAGEN Chip Werden Adressen, Daten und Taktsignal auf gleiche Art in den I/O-Zellen des FPGAs erzeugt, so verlassen sie den FPGA synchron und mit gleicher Phasenlage. Die Signallaufzeiten zwischen FPGA und HAGEN Chip sind für die Daten-, Adress- und die Taktleitungen nahezu gleich [25], die Daten-, Adress- und das Taktsignale haben folglich auch am HAGEN Chip die gleiche Phasenlage. Die I/O-Register im HAGEN Chip registrieren die Eingangsdaten mit steigender und mit fallender Taktflanke. Aufgrund der gleichen Phasenlage zwischen Takt und Daten werden allerdings die Setup- und Hold-Zeiten der Eingangsregister verletzt, die Daten können nicht zuverlässig

im HAGEN Chip registriert werden. Um eine stabile Übertragung in Schreibrichtung zu erreichen, muss daher die Phase der Adressen und Daten gegenüber der Phase des Taktsignals verschoben werden. Nach der Betrachtung der Leserichtung werden Möglichkeiten beschrieben, wie diese Phasenverschiebung erreicht werden kann.

Lesen vom HAGEN Chip Bei einem Lesezugriff schreibt der HAGEN Chip die Datenleitungen und den Rückgabetakt synchron zum FPGA. Sowohl die obere als auch die untere Chiphälfte senden jeweils einen Rückgabetakt. Dabei ist der HAGEN Chip darauf ausgelegt, mehrere HAGEN Chips an einem gemeinsamen Bus zu betreiben. Daher sind die Daten und auch die beiden Rückgabetakte im HAGEN Chip entsprechend implementiert worden: Daten und auch der Rückgabetakt werden nur getrieben, wenn ein Lesebefehl dekodiert wurde, in der übrigen Zeit werden die Leitungen hochohmig ("tristate") geschaltet.

Dies führt zu folgendem Problem: Während eines Schreibzugriffs wird der Rückgabetakt weder vom FPGA noch vom HAGEN Chip getrieben, dadurch entladen sich die differentiellen Leitungen über den Terminierungswiderstand. Beim nächsten Lesezugriff treibt der HAGEN Chip gemäß Abb. 3.9 die Daten und auch den Rückgabetakt, allerdings kann die erste Flanke des Rückgabetaktes vom FPGA nicht zuverlässig erkannt werden. Aus diesem Grund kann der Rückgabetakt im FPGA nicht verwendet werden, um die vom HAGEN Chip gelieferten Daten zu lesen.

Synchronisation von Daten- und Taktsignalen In den vorherigen Abschnitten wurde dargestellt, dass sowohl bei Schreibzugriffen zum HAGEN Chip als auch bei Lesezugriffen die Synchronisation der ankommenden Daten zu dem jeweils zugehörigen Takt problematisch ist. Mit Synchronisation ist in diesem Falle gemeint, dass die Setup- und Hold-Zeiten der Eingangsregister – bei Schreibzugriffen im HAGEN Chip, bei Lesezugriffen im FPGA – eingehalten werden. Die Taktflanken sollten hierfür idealerweise gegenüber den Datenleitungen eine Phasenverschiebung von 90° aufweisen.

Zur Lösung dieses Problems bietet es sich an, die DCMs, die der FPGA zur Verfügung stellt (vgl. Abschnitt 1.3.2), zu verwenden. Jeder DCM kann die Phase seiner Ausgabetakte in Schritten von $\frac{1}{256}$ der Taktperiode verschieben, wobei es auch möglich ist, die Verschiebung im laufenden Betrieb vorzunehmen. Idealerweise werden für die korrekte Einstellung der physikalischen Schnittstelle zum HAGEN Chip drei DCMs benutzt:

- DCM 1 erzeugt den Versorgungstakt des HAGEN Chips und verschiebt die Phase relativ zum FPGA-internen Systemtakt. Somit lässt sich eine beliebige Phasenlage zwischen den Daten- bzw. Adressleitungen und dem Versorgungstakt am HAGEN Chip einstellen, so dass die Daten bei geeigneter Verschiebung zuverlässig im HAGEN Chip registriert werden können.

- DCM 2 verschiebt den internen Takt mit dem die Daten, die die obere Chiphälfte des HAGEN Chips an den FPGA sendet, registriert werden. Wiederum lässt sich dadurch eine optimale Phasenlage (90°) zwischen dem Takt und den ankommenden Daten erreichen.

- DCM 3 wird wie DCM 2 verwendet, allerdings für die Daten, die von der unteren HAGEN Chiphälfte geschrieben werden. Da die Signallaufzeiten vom HAGEN Chip

zum FPGA wegen des Leiterplatten-Layouts für die untere und obere Hälfte unterschiedlich sind, ist jeweils ein DCM für die untere und die obere Hälfte vonnöten.

Der verwendetet FPGA stellt insgesamt allerdings nur vier DCMs zur Verfügung. Wie zu Begin des Kapitels 3 erklärt wurde, sind drei dieser DCMs für die Synchronisation mit dem DDR-SDRAM, dem SRAM bzw. als System-DCM reserviert. Daher kann für die Schnittstelle zum HAGEN Chip nur ein einziger DCM verwendet werden. Aus diesem Grunde musste eine andere Methode zur Synchronisation der Schreib- wie der Lesedaten gefunden werden:

Um einerseits den Datentransfer zum HAGEN Chip, d.h. das Schreiben, sicherzustellen, wird der Versorgungstakt des HAGEN Chips mittels des einen verfügbaren DCMs verschoben. Um andererseits die Lesedaten im FPGA zuverlässig zu registrieren, muss die Phase des internen Taktes relativ zu der Phase der Lesedaten verschoben werden. Da keine weiteren DCMs zur Verfügung stehen, wird folgende Methode verwendet:
Die ersten Leseregister innerhalb des FPGAs werden nicht wie sonst üblich in den I/O-Zellen untergebracht, sondern in speziell gewähltem Abstand von den I/O-Zellen platziert[10]. Die Lesedaten erreichen diese Register mit einer Verzögerung, die der Signallaufzeit von den I/O-Zellen zu diesen Registern innerhalb des FPGAs entspricht. Durch diese Verzögerung verschiebt sich effektiv die Phase der Lesedaten relativ zu dem internen Systemtakt[11]. Es werden folglich die internen Signallaufzeiten im FPGA verwendet, um eine Phasenverschiebung der Lesedaten zu erreichen, und so die Synchronisation dieser Übertragungsrichtung sicherzustellen. Die Phasenverschiebung ist dabei abhängig vom Routing innerhalb des FPGAs und dem physikalischen Abstand zwischen platzierten Leseregistern und den zugehörigen I/O-Zellen.

Durch Verschiebung des Versorgungstaktes kann also der Schreibzugriff zum HAGEN Chip und durch die Platzierung der Leseregister kann der Lesezugriff vom HAGEN Chip synchronisiert werden. Zum Test und zur Verifikation dieser Überlegungen wurden drei verschiedene Platzierungen der Leseregister untersucht: Direkt neben den I/O-Zellen und im Abstand von 10 bzw. 30 Logikelementen von den I/O-Zellen. Dann wurde, ähnlich dem Aufbau zum Test der DDR-SDRAM-Schnittstelle, der HAGEN-Versorgungstakt verschoben und dabei gemessen, ob Transferfehler auftreten oder nicht. Diese Messungen wurden auf 8 Nathan Netzwerkmodulen einer Backplane ausgeführt, um bauteilbedingte Schwankungen der FPGAs oder der HAGEN Chips zu erkennen. Die verwendete Taktfrequenz ist 156,25 MHz für den HAGEN Versorgungstakt, die Daten wechseln mit doppelter Datenrate, also auf jeder Leitung mit 312,5 Mbit/s. Aus der Zusammenstellung der Ergebnisse in Abb. 3.10 ist folgendes abzulesen, schwarze Quadrate markieren einen fehlerlosen Datentransfer, bei weißen Quadraten traten Fehler auf.

- Durch manuelle Platzierung der Leseregister und durch den damit erzwungenen Abstand von den I/O-Zellen lassen sich effektiv Verzögerungen der Lesedaten erzwingen und somit die Phase der Lesedaten relativ zum internen Takt verändern. Diese

[10] Die Synthesewerkzeuge des FPGA-Herstellers erlauben es, jedes Register gezielt im FPGA zu platzieren, es muss nur der Ort und der Name des Registers zur Synthesezeit festgelegt werden.

[11] Da der interne Systemtakt über spezielle Taktnetze innerhalb des FPGAs verteilt wird, erreicht er alle Register mit nahezu gleicher Phase.

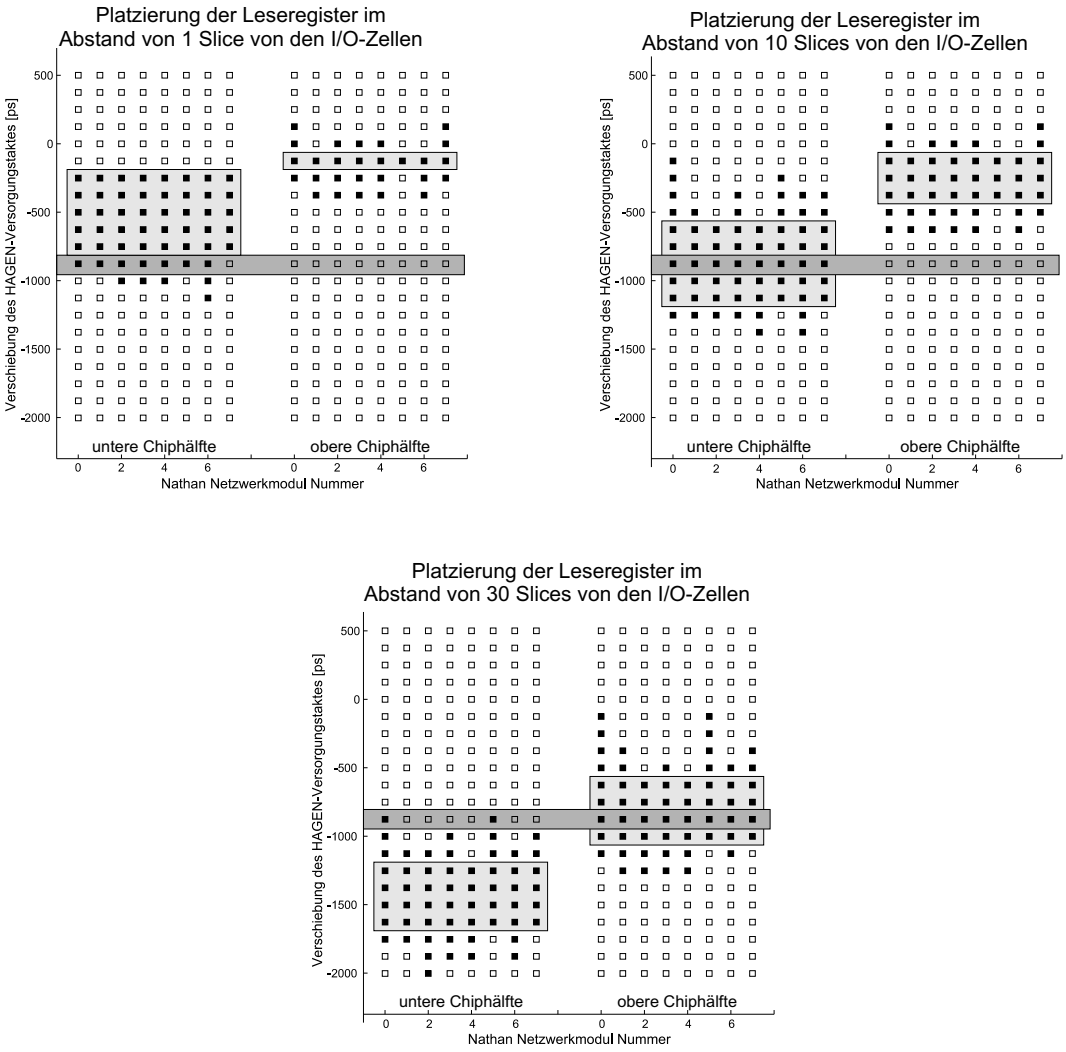

Abbildung 3.10: Untersuchung der physikalischen Schnittstelle zwischen FPGA und HAGEN Chip auf den ersten 8 Nathan Netzwerkmodulen einer Backplane. Für jedes Diagramm wurden die Leseregister in einem festen Abstand von einem, 10 bzw. 30 Logikelementen (*Slices*) von den I/O-Zellen platziert. Dadurch können die Laufzeiten der Datensignale innerhalb des FPGAs variiert werden. Ein schwarzes Quadrat zeigt an, bei welcher Verschiebung des HAGEN-Versorgungstaktes keine Fehler beim Datentransfer gemessen wurden. Ein zuverlässiger Datentransfer zu beiden Chiphälften wird erreicht, wenn der HAGEN-Versorgungstakt um -875 ps verschoben wird und die Leseregister der unteren Chiphälfte im Abstand von 10 Logikelementen, die Leseregister der oberen Chiphälfte im Abstand von 30 Logikelementen von den I/O-Zellen platziert werden.

Phasenverschiebung tritt bis auf geringe, bauteilbedingte Schwankungen bei allen getesteten Nathan Netzwerkmodulen in gleichem Maße auf.

- Die untere und die obere Chiphälfte unterscheiden sich deutlich in ihrem Verhalten, dieser Umstand ist auf unterschiedliche Signalführungen auf der Platine des Nathan Netzwerkmoduls zurückzuführen. Um eine stabile Übertragung sowohl zur oberen als auch zur unteren Hälfte sicherzustellen, müssen die Leseregister für die obere und untere Chiphälfte unterschiedlich platziert werden.

Für die Verifikation und den Betrieb der Schnittstelle zum HAGEN Chip wird der Versorgungstakt um $-875\ ps$ verschoben, diese Einstellung ist in Abb. 3.10 jeweils durch dunkelgraue Balken gekennzeichnet. Die Leseregister der unteren Chiphälfte werden dann im Abstand von 10 Logikelementen zu den den I/O-Zellen platziert, die Leseregister der oberen Chiphälfte dagegen im Abstand von 30 Logikelementen zu den I/O-Zellen platziert.

3.3.3 Umsortierung der Datenströme

Aufgrund der internen Organisation des HAGEN Chips müssen die Gewichtswerte von der zeilenweisen Darstellung im Sinne der Genwerte auf eine spaltenweise Darstellung für den HAGEN Chips umsortiert werden. Auch die Eingabe und Ergebnisdaten müssen umsortiert werden. Im Folgenden wird deshalb zwischen *Benutzerkoordinaten* und *Chipkoordinaten* unterschieden. Die Benutzerkoordinaten betrachten die vier Blöcke einzeln: Die Eingabedaten als Zeilenvektor der Länge 128, die Ausgabedaten als Spaltenvektor der Länge 64 und die Gewichte als Matrix. Diese Koordinaten sind in Abb. 3.11 aufgetragen.

Sortierer der Eingabe- bzw. Ausgabedaten Die Abbildung von Benutzerkoordinaten auf Chipkoordinaten für die Eingabe- und die Ausgabedaten sind in Anhang B zu finden. Pro Eingabevektor müssen 512 Bits byteweise umsortiert werden. Die Ausgabedaten umfassen pro Ausgabevektor 256 Bit, hier müssen jeweils Stücke von 2 Byte Länge umsortiert werden. Diese Umsortierung kann prinzipiell sowohl in Software als auch in Hardware innerhalb der programmierbaren Logik erfolgen. In der vorgestellten Experimentalplattform erfolgt diese Umrechnung in der programmierbaren Logik durch die Sortierer der Eingabe- bzw. Ausgabedaten (siehe Abb. 3.8). Die Trainingssoftware arbeitet vollständig in Benutzerkoordinaten und auch im Speicher liegen die Daten in Benutzerkoordinaten vor. Erst während des Einlesens aus dem externen Speicherbaustein, d.h. beim Schreiben an den HAGEN Chip werden die Eingabedaten durch den angeschlossenen Sortierer in Chipkoordinaten umgerechnet. Die Ausgabedaten werden entsprechend direkt beim Lesen von dem HAGEN Chip vor dem Schreiben in den Speicher in Benutzerkoordinaten umsortiert.

Werden die Sortierer in Hardware implementiert, so können sie zwischen Datenpfad und Ramklienten integriert werden, das Sortieren kann dadurch durchgeführt werden, ohne die Netzwerkberechnung nennenswert zu verzögern. Eine Implementation in Software dagegen müsste auf dem ohnehin stark beanspruchten PowerPC (siehe Abschnitt 5.3) erfolgen. Da dies die Trainingsgeschwindigkeiten reduzieren würde, wurde die Implementation in Hardware gewählt, obwohl dadurch zusätzliche Logikressourcen benötigt werden.

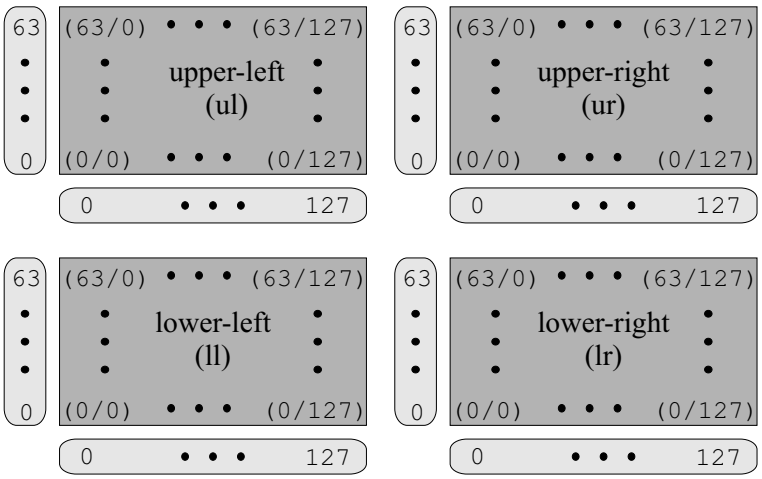

Abbildung 3.11: Definition der HAGEN Benutzerkoordinaten. Pro Block werden die 128 Eingabedaten als Zeilenvektor, die 64 Ausgabedaten als Spaltenvektor und die Gewichte als Matrix betrachtet.

Die Sortierer sind derart konzipiert worden, dass sie einfach in die Schnittstelle zwischen Datenpfad und Ramklient eingefügt werden können. Die Daten werden jeweils in *Distributed Ram* zwischengespeichert und in der jeweils richtigen Sortierung aus diesem Zwischenspeicher gelesen.

Sortierer der Gewichtsdaten Die Gewichtsdaten werden durch den evolutionären Koprozessor (vgl. Abschnitt 3.4) erzeugt und dann direkt von der HAGEN-Ansteuerung gelesen. Die Software liest oder bearbeitet die Gewichtsdaten nicht, kann sie also auch nicht umsortieren. Diese Umsortierung ist aber zwingend erforderlich, weil der Koprozessor voraussetzt, dass die einzelnen Synapsengewichte in Benutzerkoordinaten im externen Speicher abgelegt sind. Nur dann lassen sich die genetischen Operatoren, vor allem der Kreuzungsoperator, mit vertretbarem Aufwand implementieren.

Die Abbildung zwischen Benutzerkoordinaten und Chipkoordinaten der Synapsengewichte ist in Anhang B zu finden. Die Benutzerkoordinaten indizieren die Gewichtsmatrix zuerst block–, dann zeilen–, dann spaltenweise, d.h. die Gewichte einer Zeile innerhalb eines Blockes stehen konsekutiv im Sinne eines Zeilenvektors und damit optimal für den Koprozessor im Speicher.

Die Schnittstelle des HAGEN Chip ist aber so ausgelegt, dass die Gewichte spaltenweise übertragen werden müssen: Zuerst die erste Spalte der rechten Blöcke, darauf folgend die erste Spalte der linken Blöcke. Der Sortierer der Gewichte muss also die zeilenweise Darstellung der Benutzerkoordinaten in die spaltenweise Darstellung der Chipkoordinaten überführen. Dies geschieht folgendermaßen:

Als Zwischenspeicher werden vier Blockrams verwendet, es stehen also 8 kByte Speicherplatz zur Verfügung. Aus dem externen Speicherbaustein werden jeweils 8 Gewichte einer Zeile am Stück gelesen. Nach 256 dieser Zugriffe sind die ersten 8 Gewichte jeder Zeile von jedem der vier Netzwerkblöcke des HAGEN Chips im Zwischenspeicher abgelegt. Die ersten 8 Gewichte jeder Zeile entsprechen gerade den ersten 8 Spalten, d.h. die

Gewichte können nun spaltenweise aus dem Zwischenspeicher gelesen und an den HAGEN Chip übertragen werden. Während die Ablaufsteuerung diese 8 Spalten an den HAGEN Chip schreibt, liest der Sortierer bereits die Gewichte der nächsten 8 Spalten und schreibt diese Werte in die Blockrams.

Die Methode nutzt die Parallelität der programmierbaren Logik aus: Die Spalten $[k,\ k+7]$ werden an den HAGEN Chip übertragen, während die Spalten $[k+8,\ k+15]$ in den Zwischenspeicher gelesen werden. Dadurch bleibt die Anwesenheit und Funktionsweise des Sortierers fast vollständig verborgen, nur die gesamte Latenz wird um die Zeitspanne vergrößert, die der Sortierer benötigt, um die ersten 8 Spalten $[0,\ 7]$ jedes Blockes in den Zwischenspeicher zu schreiben.

Der Sortierer der Gewichte liest jeweils nur 8 Gewichte einer Zeile am Stück aus dem SDRAM, danach springt er in die nächste Zeile der Gewichtsmatrix. Durch diesen Sprung werden keine fortlaufend kontinuierlichen Zugriffe auf das SDRAM vorgenommen. Deshalb muss abgeschätzt werden, welches Zugriffsmuster und welche SDRAM Auslastung zu erwarten sind:

Eine Zeile der Gewichtsmatrix enthält 128 Gewichte mit einer Länge von jeweils 2 Byte, also 256 Byte. Eine Zeile des SDRAM umfasst 4096 Bytes, folglich werden jeweils $4096/256 = 16$ Gewichtszeilen in einer SDRAM-Zeile abgelegt. Da der Sortierer jeweils 8 Gewichte aus jeder Zeile liest, können $16 * 8 = 128$ Gewichte gelesen werden, bevor die SDRAM-Zeile gewechselt werden muss. Pro Takt überträgt das SDRAM 128 Bit, also 8 Gewichte, d.h. die SDRAM-Zeile muss alle 16 Takte gewechselt werden. Laut Tab. 3.1 benötigt ein Zeilenwechsel 7 Takte, so dass eine maximale Auslastung von $16/(16+7) = 69,6\%$ zu erwarten ist.

3.3.4 Ablaufsteuerung

Die Ablaufsteuerung kontrolliert die physikalische Schnittstelle zum HAGEN Chip und steuert den Datenpfad. Die Ablaufsteuerung ist als Zustandsautomat implementiert und ermöglicht drei verschiedene Abläufe: Die bereits im Überblick erwähnten Phasen der *Gewichtskonfiguration* und des *Netzwerkbetriebs* sowie zur Verifikation der Funktionalität der physikalischen Schnittstelle einen *Transfertest*.

Transfertest Der Transfertest schreibt 16 Bit Daten sowohl an die obere als auch an die untere Chiphälfte und liest sie daraufhin achtmal hintereinander aus. Da in der Netzwerkbetriebsphase jeweils 8 Datenwörter am Stück gelesen werden, wird dieses Muster auch für den Transfertest verwendet. Die Übertragung auf der physikalischen Schnittstelle erfolgt mit doppelter Datenrate, so dass je 8 Bit bei steigender und 8 Bit bei fallender Taktflanke übertragen werden, dadurch kann die DDR-Schnittstelle effektiv auf Bitfehler beim Schreiben und Lesen untersucht werden.

Gewichtskonfigurationsphase Die Synapsengewichte des HAGEN Chips werden in der Gewichtskonfigurationsphase geschrieben. Die Gewichte werden dabei aus dem externen Speicher über einen Ramklienten gelesen und durch den Sortierer der Gewichte in Chipkoordinaten umsortiert. Abb. 3.12 zeigt das Zustandsdiagramm der Ablaufsteuerung für die Gewichtskonfigurationsphase.

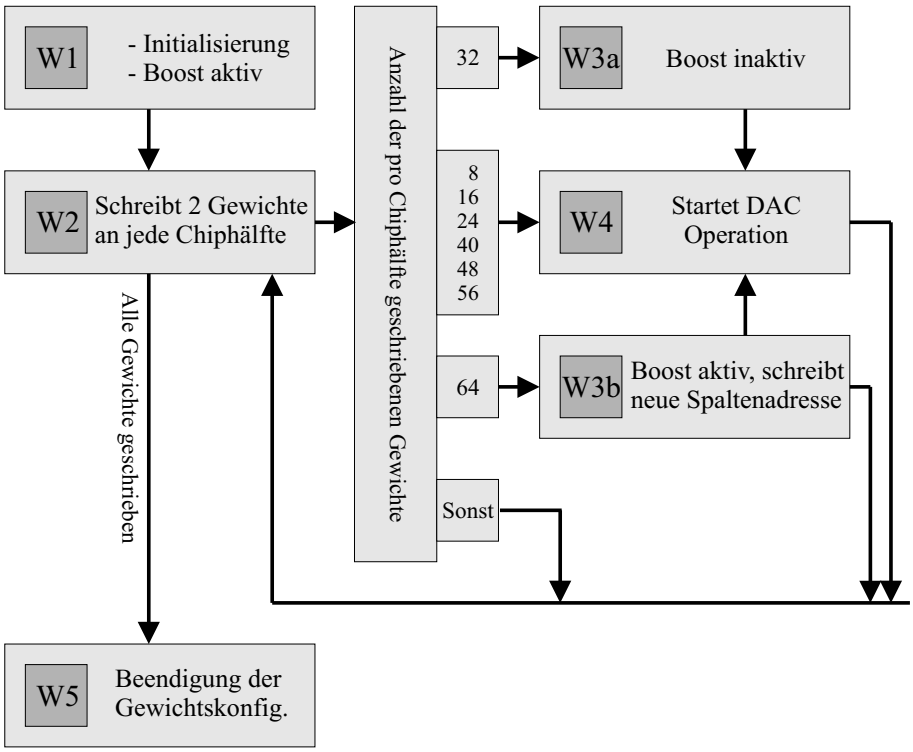

Abbildung 3.12: Zustandsdiagramm der Gewichtskonfigurationsphase. Nach der Initialisierung werden im Zustand `W2` jeweils 4 Gewichte geschrieben. Die Zustände `W3a`, `W3b` und `W4` setzten spezielle Schalter im HAGEN Chip bzw. aktualisieren die Zeilen- und Spaltenadressen.

Die Gewichtskonfiguration beginnt mit dem Zustand `W1`. In `W1` werden die Adresszähler initialisiert und das Boost-Signal[12] gesetzt.

Jede Chiphälfte enthält 8 DACs, pro DAC stehen zwei digitale Eingangsregister (`A` und `B`) zur Verfügung. Dadurch können die DACs die digitalen Werte aus einem Eingangsregister umwandeln, während über die Schnittstelle neue Werte in das jeweils andere Eingangsregister geschrieben werden. Die Ablaufsteuerung organisiert diese Parallelität:

Im Zustand `W2` werden immer 4 Gewichte in die digitalen Eingangsregister geschrieben, jeweils 2 Gewichte an die obere und 2 Gewichte an die untere Chiphälfte. Nach 4 Takten im Zustand `W2` wurden pro Chiphälfte 8 Gewichtswerte geschrieben, die Eingangsregister `A` aller DACs enthalten gültige Werte.

Der Zustand `W4` startet die Umwandlung in den DACs und schaltet gleichzeitig auf die Programmierung der Eingangsregister `B` um: Während die DACs die digitalen Werte der Eingangsregister `A` umwandeln, beschreiben die folgenden 4 Takte im Zustand `W2` die Eingangsregister `B`. Durch diese Parallelität ist eine kontinuierliche Benutzung der DACs möglich.

[12] Über den *Boost* lässt sich die Treiberstärke beim Schreiben der Gewichte auf die Gewichtskondensatoren im HAGEN Chip regeln. Um einen großen Dynamikbereich zu ermöglichen, wird anfangs mit hohem Strom geschrieben. Für eine bessere Genauigkeit wird gegen Ende des Schreibvorgangs mit geringerem Strom geschrieben. Das Boost-Signal steuert diese Treiberstärke.

Für das analoge Ergebnis der DAC-Umwandlung stehen insgesamt 128 analoge Speicher pro Chiphälfte zur Verfügung (vgl. Abb. 1.5). Nachdem die DACs 64 dieser Speicher mit analogen Gewichtswerten beschrieben haben, werden diese Gewichte in die Synapsenmatrix übertragen. Parallel dazu beschreiben die DACs die anderen 64 Speicher. Zur Steuerung dieser Operationen dienen die Zustände `W3a` und `W3b`. In `W3a` wird das Boost-Signal gelöscht, in `W3b` wird eine neue Spalte der Synapsenmatrix ausgewählt. Der Zustand `W5` wird erreicht, sobald alle Gewichte geschrieben wurden und markiert eine korrekte Beendigung der Gewichtskonfiguration.

Effektiv müssen für das Schreiben der je 64 Gewichte einer Spalte in die obere und untere Chiphälfte insgesamt 40 Takte im Zustand `W2`[13], 8 Takte im Zustand `W4` und je ein Takt im Zustand `W3a` und `W3b` durchlaufen werden. Insgesamt werden also zum Schreiben von 128 Gewichten 50 Takte benötigt. Die vollständige Konfiguration aller 32768 Gewichte ist folglich in 12800 Takten möglich.

Netzwerkbetriebsphase Während der Netzwerkbetriebsphase werden die Eingabedaten in digitaler Form an den HAGEN Chip gesendet, daraufhin wird die analoge Netzwerkoperation auf allen vier Blöcken ausgeführt und die wiederum digitalen Ergebnisdaten zurück gelesen. Die Initialisierung der Eingabe- und Ausgabeneuronen und die analoge Netzwerkoperation innerhalb des HAGEN Chips kann über die Schnittstelle mit speziellen Befehlen, im Folgenden *Taktmuster* genannt, programmiert werden. Die Definition und Bedeutung dieser Taktmuster findet sich in [69]. Anzahl und Art der Taktmuster wird in den Status Registern der HAGEN-Ansteuerung gespeichert und kann über den Slow-Control-Klienten geschrieben und gelesen werden. Das Zustandsdiagramm der Netzwerkbetriebsphase ist in Abb. 3.13 dargestellt, die einzelnen Zustände und ihre Funktion werden im Folgenden erläutert.

Die Netzwerkbetriebsphase beginnt mit dem Zustand `D0`. In `D0` werden durch Schreiben der entsprechenden Taktmusterfolge von 4 Takten Länge die Eingabe- und Ausgabeneuronen initialisiert. Im Zustand `D1` werden pro Takt 32 Bit sowohl an die obere als auch an die untere Chiphälfte geschrieben. Nach 4 Takten im Zustand `D1` sind je 128 Bit, d.h. die Eingabedaten aller 128 Eingabeneuronen der beiden rechten Blöcke geschrieben worden. Es wird in den Zustand `D2` gewechselt, um die Chipseite für das Schreiben der Eingabedaten zu wechseln: In den darauf folgenden 4 Takten `D1` werden die Eingabedaten für die linke Seite geschrieben.

Nach erfolgter Übertragung der Eingabedaten werden im Zustand `D3` die Taktmuster zur Ausführung der analogen Netzwerkoperation geschrieben. Die Anzahl und Art der Taktmuster ist über die Status Register der HAGEN-Ansteuerung festgelegt. Pro Netzwerkzyklus werden 6 Takte im Zustand `D3` benötigt. Der Zustand `D3a` beendet das Schreiben der Taktmuster und wechselt die Chipseite für das Schreiben der Eingabedaten zurück auf die rechte Seite. Im Zustand `D4` werden die Ergebnisdaten ausgelesen. Pro Takt werden 32 Bit von der oberen wie von der unteren Chiphälfte gelesen. Nach 4 Takten im Zustand `D4` ist das Lesen der 256 Ausgabeneuronen somit abgeschlossen. Nach der Leseoperation wird ein Takt im Zustand `D5` benötigt, um den bidirektionalen Datenbus

[13] In 32 dieser 40 Takte im Zustand `W2` werden jeweils 2 Gewichte geschrieben, die restlichen 8 Takten werden zur Synchronisation benötigt.

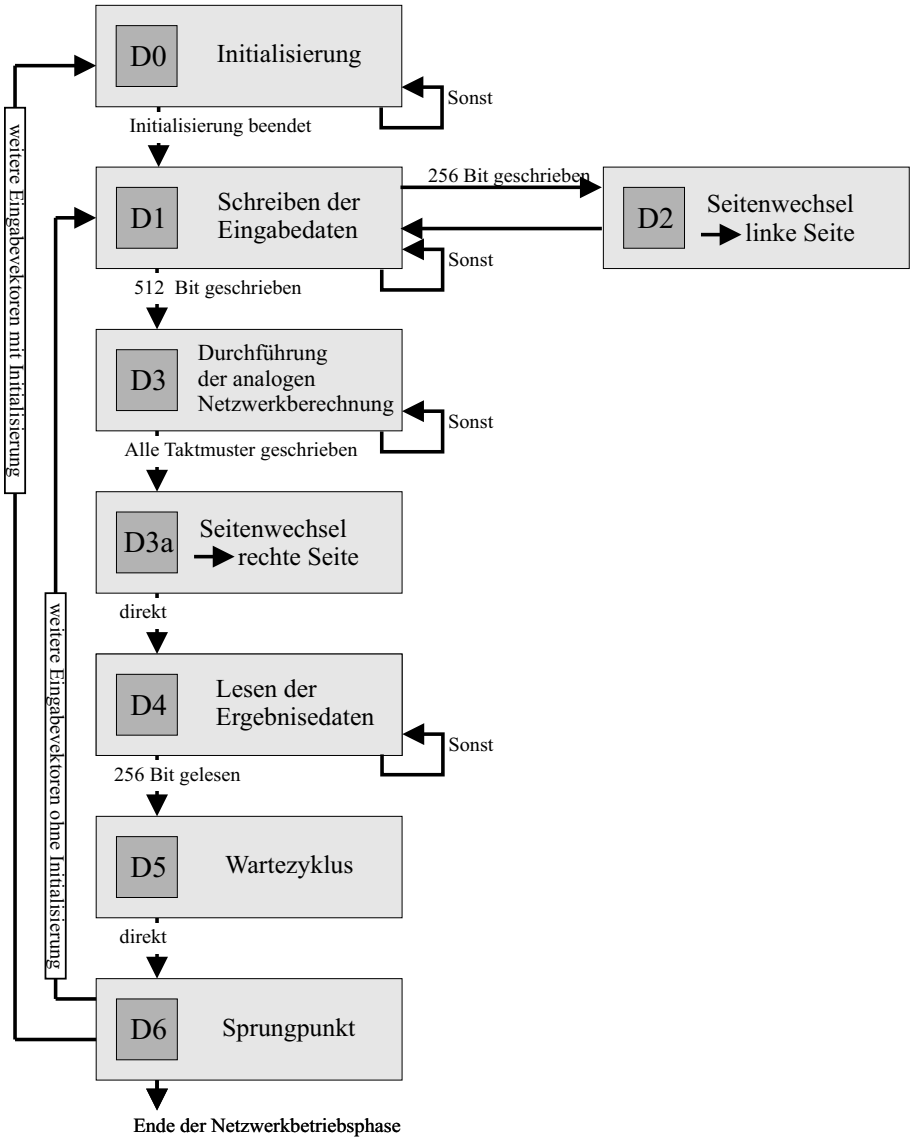

Abbildung 3.13: Zustandsdiagramm der Ablaufsteuerung während der Netzwerkbetriebsphase. Die wichtigsten Aufgaben sind das Schreiben der digitalen Eingabedaten im Zustand `D1`, die Durchführung der analogen Netzwerkberechnung im Zustand `D3` und das Zurücklesen der digitalen Ergebnisdaten im Zustand `D4`.

zwischen FPGA und HAGEN Chip von Lesezugriff auf Schreibzugriff umzustellen, d.h. die Treiber im HAGEN zu deaktivieren und im FPGA zum richtigen Zeitpunkt zu aktivieren.

Im Zustand `D6` wird entschieden, ob ein weiterer Eingabevektor bearbeitet werden soll. Die Anzahl der Eingabevektoren wird dabei den Status Registern der HAGEN-Ansteuerung entnommen. Über die Status Register wird auch festgelegt, ob in den Zustand `D0` oder `D1` gesprungen wird, d.h. ob eine neue Initialisierung der Eingabe- und Ausgabeneuronen durchgeführt wird. Wenn alle Eingabevektoren bearbeitet wurden, so wird die Netzwerkbetriebsphase beendet.

Die Ausführung eines zweischichtigen Netzwerkes erfordert zwei Netzwerkzyklen. Dafür werden 34 Takte benötigt: 4 Takte im Zustand `D0` zur Initialisierung, insgesamt 8 Takte im Zustand `D1` zum Schreiben der Eingabedaten, 12 Takte im Zustand `D3` zum Schreiben der Taktmuster, 4 Takte im Zustand `D4` zum Lesen der Ergebnisdaten und 6 Takte zur Synchronisation in den Zuständen `D1`, `D2`, `D3a`, `D5`.

3.3.5 Verifikation und Leistungsdaten

Zur Verifikation der physikalischen Schnittstelle wurde der Transfertest (siehe Abschnitt 3.3.4 auf den ersten 8 Nathan Netzwerkmodulen einer Backplane ausgeführt und dabei jeweils ein Datenvolumen von 3,05 GByte fehlerfrei gelesen. Dabei wurden die in Abschnitt 3.3.2 ermittelte optimale Platzierung der Leseregister und die Phasenverschiebung von $-875\ ps$ des Versorgungstaktes verwendet. Die Bitfehlerrate auf jedem Nathan Netzwerkmodul liegt damit unter $3,1 \cdot 10^{-10}$.

Die HAGEN-Ansteuerung wurde mit den nötigen Teilen Ablaufsteuerung, Datenpfad, Sortierer der Datenströme, angeschlossenen Ramklienten und Slow-Control-Klient mit einer Zielfrequenz von 100 MHz für den Takt des Zustandsautomaten, d.h. 200 MHz für den HAGEN-Versorgungstakt, synthetisiert. Da sowohl die Ramklienten als auch der Slow-Control-Klient asynchrone Schnittstellen anbieten, wird die gesamte HAGEN-Ansteuerung mit Ausnahme des Datenpfades mit einem Takt, dem Takt des Zustandsautomaten, gesteuert.

Die physikalische Schnittstelle zwischen HAGEN Chip und FPGA wurde bei einer Frequenz von 78,125 MHz für den Zustandsautomaten, bzw. 156,25 MHz für den HAGEN-Versorgungstakt optimiert[14]. Eine Taktperiode des Zustandsautomaten dauert folglich 12,8 ns. Gemäß Abschnitt 3.3.4 benötigt damit die Konfiguration aller Synapsengewichte $163,84\ \mu s$ und ein Netzwerkzyklus wird in 76,8 ns bearbeitet. Die Auswertung eines Eingabevektors und die Durchführung von zwei Netzwerkzyklen einschließlich des Datentransfers in beide Richtung ist in $435,2\ ns$ abgeschlossen.

Die Logikressourcen, die die HAGEN-Ansteuerung belegt, sind in Tab. 3.6 zusammengefasst. Einschließlich der Ramklienten belegt die HAGEN-Ansteuerung 997 FFs und 2635 LUTs, dies entspricht 9.0% rsp. 23,8% der verfügbaren Ressourcen des verwendeten FPGAs.

[14]Durch die feste Signallaufzeit innerhalb des FPGAs zeigt die Schnittstelle bei den Einstellungen ein frequenzabhängiges Verhalten. Um die Schnittstelle auch bei anderen Frequenzen betreiben zu können, müssten neue Messungen durchgeführt werden.

Komponente	FFs	LUTs	Block-Rams	DDR-I/O Register
Ablaufsteuerung und Statusregister	413	947	1	0
Datenpfad	195	120	2	80
Sortierer:				
Eingabedaten	36	408	0	0
Ergebnisdaten	22	334	0	0
Gewichte	97	444	4	0
HAGEN-Ansteuerung einschließlich 3 Ramklienten	997	2635	13	80
in Relation zu den vorhandenen Ressourcen	9,0%	23,8%	29,6%	10,1%

Tabelle 3.6: Ressourcenverbrauch der einzelnen Module der HAGEN-Ansteuerung

3.3.6 Portabilität

Die HAGEN-Ansteuerung verwendet einerseits die physikalische Schnittstelle zum HAGEN Chip, andererseits innerhalb des FPGAs einen Slow-Control-Klienten sowie drei Ramklienten. Die physikalische Schnittstelle wird durch den *Datenpfad* gekapselt[15].

Die FPGA-internen Schnittstellen der Ramklienten und des Slow-Control-Klienten sind für das Darkwing- und Nathan-System einheitlich, mit einer Ausnahmen: Der externe Speicher im Nathan-System arbeitet mit doppelter Datenrate, d.h. pro Zugriff werden statt 64 Bit wie im Darkwing-System 128 Bit zum externen SDRAM-Modul übertragen. Dieser Umstand muss bei der Ansteuerung der Ramklienten bedacht werden. Da die Datenbusse zwischen Ramnutzer und Ramklient 64 Bit breit ausgelegt sind, muss im Nathan-System folglich immer eine gerade Anzahl an Zugriffen, angefangen mit einer geraden Adresse erfolgen, um den Zugriff vollständig zu machen. Der gesamte Datentransfer zwischen HAGEN-Ansteuerung und den drei Ramklienten wird über die Sortierer abgewickelt, die ohnehin nur in Vielfachen von 128 Bit auf die Daten im externen Speicherbaustein zugreifen, damit also kompatibel zu sowohl einer Zugriffsgröße von 64 Bit als auch von 128 Bit sind.

Mit Ausnahme der physikalischen Schnittstelle und Teilen des Datenpfades ist daher die Implementation der HAGEN-Ansteuerung durch die Verwendung der einheitlichen Schnittstellen zu Slow-Control und Speicheransteuerung plattformunabhängig.

[15] Für das Darkwing System wurde eine eigene Komponente für den Datenpfad entwickelt. Da der im Darkwing System verwendete FPGA weder bidirektionale DDR-I/O-Zellen noch verschiebbare Takte ermöglicht, konnte eine Frequenz des HAGEN-Versorgungstaktes von maximal 84 MHz erreicht werden.

3.4 Evolutionärer Koprozessor

Prinzipiell lassen sich mit den bisher beschrieben Komponenten Experimente mit dem HAGEN Chip durchführen: Der Kontroll-PC hat gemäß Abb. 3.7 Zugriff auf den externen Speicher sowie auf alle Slow-Control-Klienten jedes Nathan Netzwerkmodules. Damit können vom Kontroll-PC Eingabe– und Gewichtsdaten in den Speicher geschrieben werden, es kann die HAGEN-Ansteuerung initialisiert und gestartet, sowie die Ergebnisdaten zurück gelesen werden. Die Trainingsalgorithmen werden dabei vollständig durch den Prozessor des Kontroll-PCs Nathan-Systems ausgeführt. Durch die geringe Geschwindigkeit und die hohen Latenzen der Slow-Control sind aber nur sehr einfache Probleme in vertretbarer Zeit lösbar.

Die weiteren Komponenten, die nötig sind, um das evolutionäre Training auf dem Nathan-System mit hoher Geschwindigkeit auszuführen, werden im Folgenden vorgestellt: Der evolutionäre Koprozessor berechnet die die datenintensiven Teile der Trainingsalgorithmen innerhalb der programmierbaren Logik. In Abschnitt 3.6 folgt dann eine Beschreibung des eingebetteten PowerPCs, der die Trainingssoftware lokal ausführt, sowie der nötigen Schnittstellen zwischen PowerPC und programmierbarer Logik.

3.4.1 Überblick

Um auf dem Nathan-System beim Training mit evolutionären Algorithmen hohe Geschwindigkeiten zu erreichen, wird die Bearbeitung des genetischen Materials, d.h. die Erzeugung der neuen Generationen, in dedizierter Hardware ausgeführt. Die nötige Variabilität wird durch einen Koprozessoransatz erreicht: Die Trainingssoftware wird durch den Hauptprozessor ausgeführt, sie legt die Parameter der genetischen Operatoren fest und bestimmt, welches genetische Material verarbeitet wird. Die Berechnungen, die Generierung der vom genetischen Algorithmus benötigten Zufallszahlen und ein großer Teil des Datentransfers wird aber durch einen Koprozessor unabhängig von der Software und dem Hauptprozessor abgewickelt. Dieser Evolutionäre Koprozessor kommt in beiden Systemen zum Einsatz, der Hauptprozessor im Darkwing-System ist der Prozessor des verwendeten PCs, im Nathan-System wird die Trainingssoftware durch den PowerPC ausgeführt [66] [30] [28]. Abb. 3.14 zeigt eine Übersicht des Koprozessors mit den wichtigsten Funktionsblöcken *Befehlsspeicher*, *Dekodierer*, *Pipelinesteuerung* und *Pipeline*.

Das genetische Material wird mit Hilfe von zwei Ramklienten aus dem externen Speicherbaustein gelesen und in mehreren Stufen innerhalb der *Pipeline* bearbeitet. Um einen hohen Datendurchsatz zu erreichen, ist die Pipeline 4fach parallel ausgelegt, d.h. es werden in jeder Stufe 4 Synapsengewichte parallel verarbeitet. Die Steuerung der Pipeline erfolgt teilweise direkt durch den *Dekodierer*, teilweise auch durch das Modul *Pipelinesteuerung*, wodurch größere Freiheiten bei der Programmierung der Pipeline möglich sind. Die *Software* kann einerseits über einen Slow-Control-Klienten, andererseits durch direktes Schreiben in den *Befehlsspeicher* auf den Koprozessor zugreifen. Dabei werden die Instruktionen und auch häufig verwendete Daten im Befehlsspeicher vorgehalten, der Slow-Control-Anschluss wird nur zum Start und zur Statusabfrage des Koprozessors verwendet. In der gegenwärtigen Fassung wird der Koprozessor verwendet, um anhand der

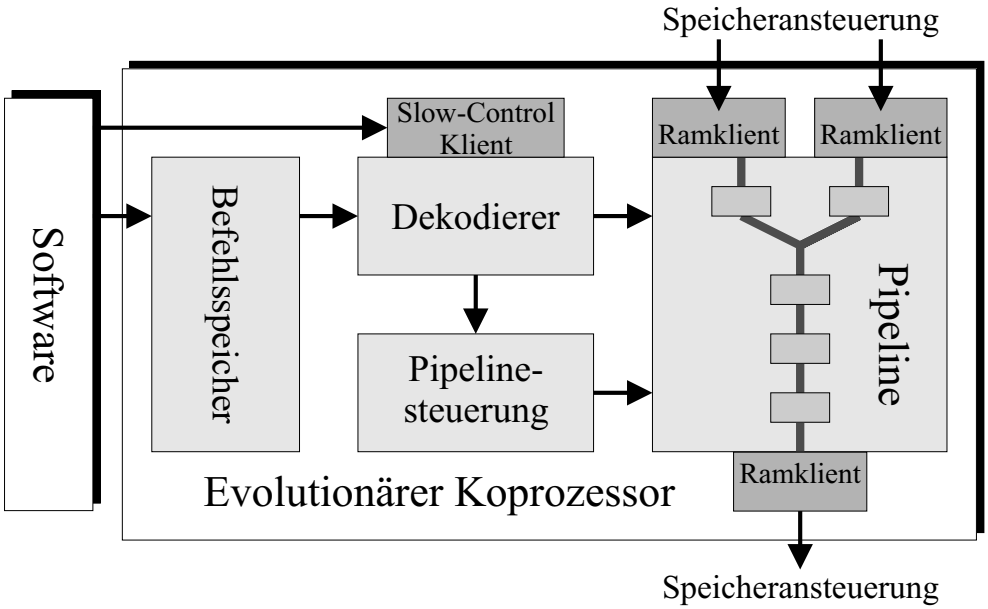

Abbildung 3.14: Aufbau des evolutionären Koprozessors. Der Koprozessor wird von der Software über Befehlsspeicher und einen Slow-Control-Klienten gesteuert. Der Dekodierer liest und dekodiert die Befehle und steuert die Pipeline einerseits direkt, andererseits über die Pipelinesteuerung. Die genetische Daten werden über Ramklienten und die Speicheransteuerung aus dem externen Speicher gelesen bzw. dorthin geschrieben.

Softwarebefehle das genetische Material für den HAGEN Chip zu berechnen. Da die Synapsengewichte des HAGEN Chips mit einer Genauigkeit von 11 Bit geschrieben werden, arbeitet auch der Koprozessor mit 11 Bit-Integerwerten. Es ist aber problemlos möglich, diese Größe an die Erfordernisse anderer Netzwerkchips anzupassen.

3.4.2 Pipeline

Abb. 3.15 zeigt eine schematische Übersicht der Pipeline. Der Zusammenhang zwischen den Funktionsblöcken *Dekodierer*, *Pipelinesteuerung* und *Pipeline* ist in Abb. 3.16 dargestellt. In der Pipeline wird das genetische Material durch Kreuzungsoperator, Mutation (vgl. Abschnitt 1.2) und einen auf den HAGEN Chip angepassten Spezialoperator zur Erzeugung von Integerneuronen [70] bearbeitet. Die Pipeline verarbeitet Gensequenzen von mindestens 8 und maximal 4096 Genen[16]. Speziell der Kreuzungsoperator verknüpft zwei Individuen miteinander, daher werden über zwei reine Lese-Ramklienten beide Individuen, im Folgenden *Eltern* genannt, simultan aus dem externen Speicher gelesen. Am Ende der Berechnungen wird ein Individuum, der *Nachkomme* über einen reinen Schreibklienten zurück in den externen Speicher geschrieben. Sollen Gensequenzen von nur einem Elternteil ohne Anwendung des Kreuzungsoperators mutiert werden, so arbeitet nur einer der Lese-Ramklienten.

[16]Die minimale Größe ist durch die kleinste Zugriffsgröße auf das DDR-SDRAM vorgegeben, die maximale Größe entspricht einer Zählerbreite von 12 Bit.

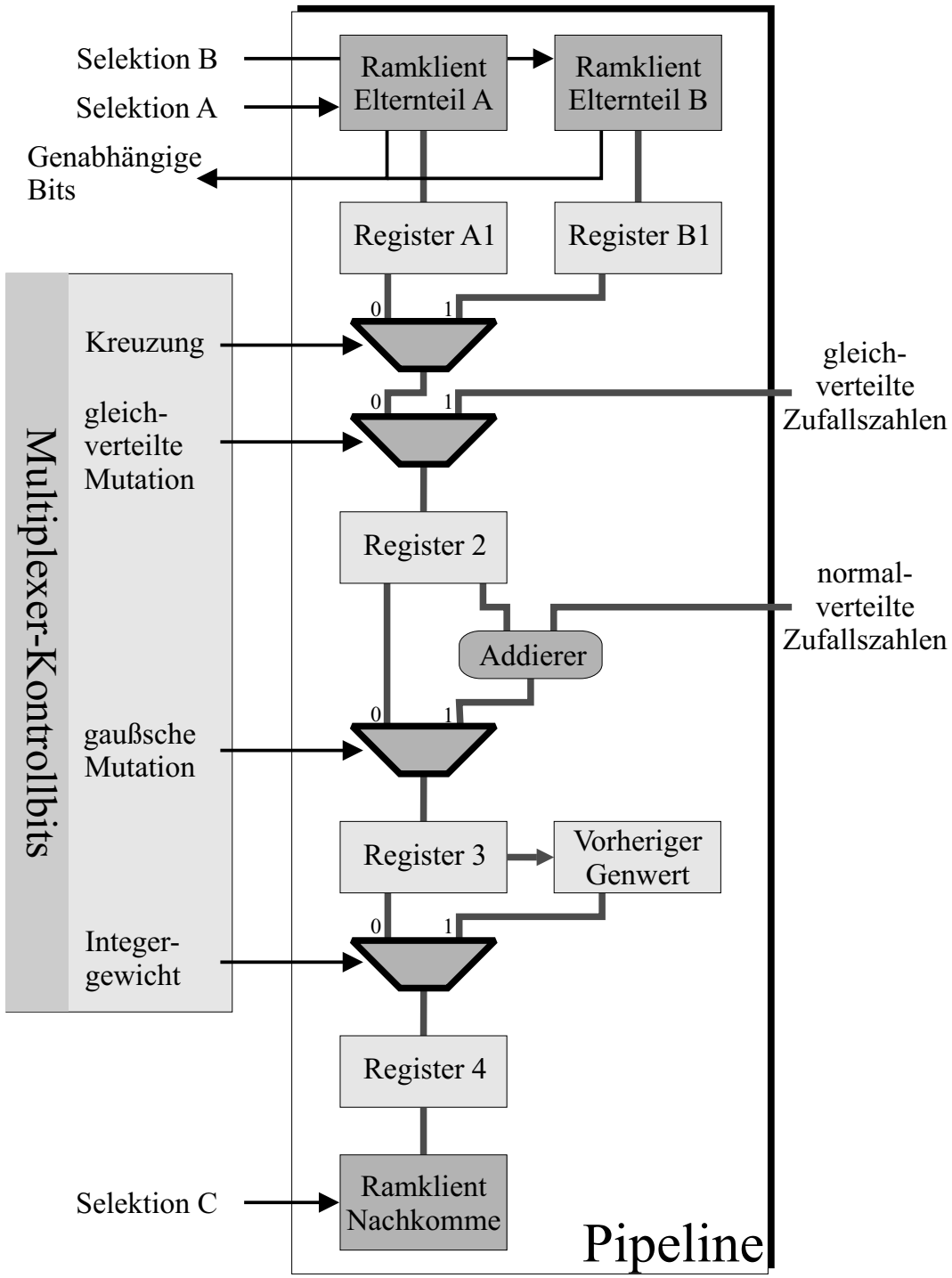

Abbildung 3.15: Schematische Übersicht der Pipeline des evolutionären Koprozessors. Als Datenquellen und -senke werden Ramklienten verwendet. Neben den Selektionskommandos und Multiplexer-Kontrollbits, die von außerhalb durch die Komponenten Dekodierer und Pipelinesteuerung erzeugt und bereitgestellt werden, benötigt die Pipeline sowohl gleichverteilte als auch gaußverteilte Zufallszahlen. In jeder Pipelinestufe wird jeweils ein Gen mit der Genauigkeit von 11 Bit verarbeitet.

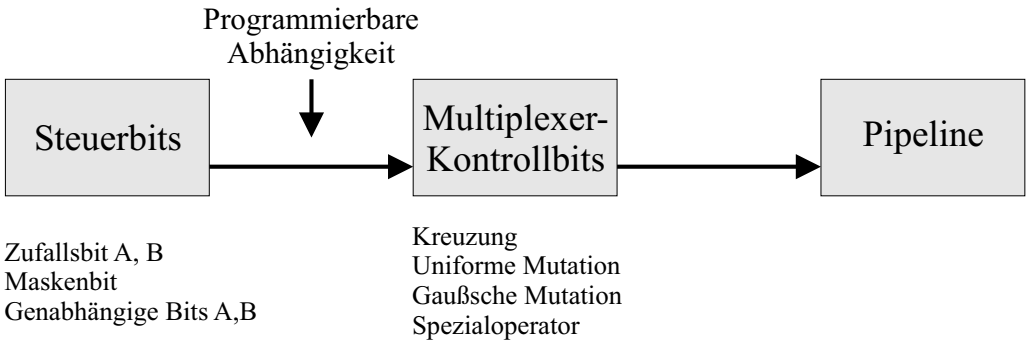

Abbildung 3.16: Die Multiplexer-Kontrollbits steuern die Pipeline. Um größere Freiheiten in der Formulierung des evolutionären Algorithmus zu ermöglichen, sind die Multiplexer-Kontrollbits von den Steuerbits abhängig, die genaue Abhängigkeit hingegen ist von der Software programmierbar.

Durch die Verwendung der drei Ramklienten muss sich die Pipeline nicht um die Arbitrierung der Zugriffe auf den externen Speicherbaustein kümmern, da diese Aufgabe von der Speicheransteuerung übernommen wird. Die Ramklienten bieten noch einen weiteren Vorteil: Es werden sowohl in den Leseklienten die von der Pipeline zu lesenden Daten als auch in dem Schreibklienten die geschriebenen Daten in FIFO-Speichern zwischengelagert. Dadurch werden die Ramklienten als Datenquellen, bzw. Datensenken der Pipeline verwendet.

Die Pipeline wird von außen gesteuert, die Steuersignale werden teilweise durch den Dekodierer und teilweise durch die Komponente Pipelinesteuerung generiert. Es können zwei Arten von Steuersignalen unterschieden werden:

- Die *Selektionskommandos* bestimmen, von welchen Adressen aus dem externen Speicherbaustein das genetische Material der beiden Eltern gelesen wird und wohin der daraus erzeugte Nachkomme geschrieben werden soll. Jeder der drei Datenströme erwartet Kommandos der Art `(Speicheradresse, Anzahl Gene)`. Als ein Chromosom wird im Folgenden eine Sequenz aufeinander folgender Gene bezeichnet. Ein Chromosom für den HAGEN Chip enthält 128 Gene (vgl. Abschnitt 1.2. Pro Chromosom ist i.A. für jeden Datenstrom ein Selektionskommando erforderlich. Durch die Größe des externen Speichers lassen sich ganze Populationen problemlos abspeichern.

- Die *Multiplexer-Kontrollbits* bestimmen, wie das genetische Material durch die genetischen Operatoren bearbeitet wird. Für jedes Gen muss z.B. festgelegt werden, von welchem Elternteil es übernommen wird, ob und wie es mutiert werden soll und ob es durch den Spezialoperator zur Erzeugung von Integerneuronen beeinflusst werden soll. Pro Gen werden mehrere Multiplexer-Kontrollbits benötigt.

Neben diesen beiden Arten von Steuersignalen benötigt die Pipeline zusätzlich Zufallszahlen zur Realisierung der Mutation. Es werden sowohl Zufallszahlen aus einer gleichverteilten Verteilung als auch aus einer gaußschen Verteilung bereitgestellt. Mittels rückgekoppelter Schieberegister ("Linear Feedback Shift Register") [72] lassen sich gleichverteilte Zufallszahlen in Hardware sehr effizient erzeugen, eine gaußsche Verteilung der Zufallszahlen kann durch Addition von gleichverteilten Zufallszahlen erzeugt werden [56].

Gemäß der obigen Aufteilung steuern die Selektionskommandos die beiden Quellen und die Senke der Pipeline, sie entscheiden also, welche Chromosomen bearbeitet werden. Die Multiplexer-Kontrollbits dagegen bestimmen, wie diese Chromosomen bearbeitet werden, d.h. wie die genetischen Operatoren arbeiten. Beide Arten von Steuersignalen werden nach Anweisungen der Software durch den Dekodierer bzw. die Pipelinesteuerung (vgl. Abb. 3.14) erzeugt und an die Pipeline weitergegeben. Auf die Art der Erzeugung wird in den Abschnitten 3.4.3 und 3.4.4 eingegangen.

Ein *Gen* bestimmt bei dem verwendeten HAGEN Chip ein Synapsengewicht und hat somit eine Genauigkeit von 11 Bit (vgl. Abschnitt 1.1.3). Im externen Speicher werden allerdings für jedes Gen 16 Bit Speicherplatz reserviert, dadurch stehen 5 Bits pro Gen an Speicherplatz zusätzlich zur Verfügung. Diese 5 Bits – im Folgenden *Genabhängige Bits* genannt – werden verwendet, um die genetischen Operatoren genweise zu steuern. Die Pipeline liest diese genabhängigen Bits zusammen mit den Genwerten aus dem externen Speicher, verarbeitet sie aber nicht selber, sondern leitet sie an die Komponente Pipelinesteuerung. Die Pipelinesteuerung erzeugt aus den genabhängigen Bits die Multiplexer-Kontrollbits (siehe Abschnitt 3.4.3).

Die Pipeline besteht aus vier Pipelinestufen:

- Stufe 1 - Verzögerung
  In der ersten Stufe werden die Gene beider Elten nicht verändert, sondern durch eine erneute Registerung um einen Takt verzögert. Diese Verzögerung ist notwendig, um die genabhängigen Bits, die zusammen mit den Genwerten im externen Speicher abgelegt sind, an die Komponente Pipelinesteuerung zu übertragen.

- Stufe 2 - Kreuzungsoperator und gleichverteilte Mutation
  In der zweiten Pipelinestufe wird der Kreuzungsoperator und die gleichverteilte Mutation nacheinander ausgeführt, beide benötigen jeweils nur einen 11 Bit breiten 2:1-Multiplexer. Die Steuerung dieser beiden Multiplexer erfolgt durch zwei Multiplexer-Kontrollbits, die in der Komponente Pipelinesteuerung erzeugt und der Pipeline bereitgestellt werden.

- Stufe 3 - Gaußsche Mutation
  Die dritte Pipelinestufe berechnet die gaußsche Mutation. Die Zufallszahl entstammt einer gaußschen Verteilung und wird auf den aktuellen Genwert aufaddiert. Überschreitet das Ergebnis den erlaubten Maximalwert (Maximalwert), so wird stattdessen der Maximalwert (Minimalwert) verwendet.

- Stufe 4 - Spezialoperator: Integerneuronen
  Mehrere der binären Neuronen des HAGEN Chips können zusammengefasst und als ein Integerneuron verwendet werden (siehe Abschnitt 1.1.3). Für ein maximales Gewicht der Stärke W_n müssen dann Einzelgewichte der Stärken $\frac{W_n}{2}$, $\frac{W_n}{4}$, etc.

erzeugt werden. In der vierten Pipelinestufe kann – wiederum gesteuert durch ein Multiplexer-Kontrollbit – die Hälfte des vorherigen Gewichtes verwendet werden. Damit ist es möglich, die Gewichte von Integerneuronen mit der Pipeline zu erzeugen.

Parallelität Eine einzelne Pipeline bearbeitet in jedem Arbeitstakt 16 Bit, 11 Bit stellen den Genwert bzw. das Gewicht einer Synapse für den HAGEN Chip dar und zusätzlich können bis zu 5 genabhängige Bits verwendet werden. Um einen höheren Datendurchsatz zu erreichen und um ohne zusätzliche Multiplexer und Kontrolllogik direkt an die 64 Bit breite Speicheransteuerung anzuschließen, wird die Pipeline 4fach parallel ausgelegt. In jedem Arbeitstakt können dadurch 64 Bit, d.h. 4 Gene parallel berechnet werden.

3.4.3 Steuerung der Pipeline

Im vorherigen Abschnitt wurde beschrieben, wie die Pipeline arbeitet und dass sie sämtliche Steuersignale von außen, d.h. von anderen Komponenten des evolutionären Koprozessors, erhält. Hier wird untersucht, wie und mit welcher Rate diese Steuersignale generiert werden müssen.

Die Pipeline wird durch Selektionskommandos und Multiplexer-Kontrollbits gesteuert. Jedes Selektionskommando weist die Pipeline an, ein Chromosom mit der typischen Größe von 128 Genen zu bearbeiten. Für beide Elternchromosomen und für den erzeugten Nachkommen ist je ein Selektionskommando erforderlich. Die 4fach parallele Pipeline benötigt dafür mindestens 32 Takte, bei drei Selektionskommandos pro Chromosom muss folglich etwa alle 11 Takte ein Kommando erzeugt werden. Diese kurze Abschätzung zeigt, dass die Erzeugung der Selektionskommandos nicht schnell sein muss. Daher werden die Selektionskommandos durch spezielle Befehle von der Komponente Dekodierer erzeugt (vgl. Abschnitt 3.4.4).

Die Multiplexer-Kontrollbits dagegen müssen in jedem Takt erzeugt werden, da sie nur jeweils für ein Gen gelten. In jedem Takt werden durch die 4fache Parallelität vier Multiplexer-Kontrollbits für jeden der in der Pipeline verwendeten Multiplexer benötigt. Da aber pro Takt maximal ein Befehl im Dekodierer ausgeführt werden kann, ist es nicht möglich, die Multiplexer-Kontrollbits direkt durch Befehle im Dekodierer zu erzeugen. Um die Multiplexer-Kontrollbits schnell und trotzdem variabel zu erzeugen, wird eine neue Gruppe von Steuersignalen eingeführt:
Die *Steuerbits*. In der aktuellen Implementation werden 5 verschiedene Steuerbits verwendet. Die verschiedenen Steuerbits sind:

- Zwei Zufallsbits A und B
  Die Zufallsbits werden mit Zufallsgeneratoren erzeugt, dabei kann die Wahrscheinlichkeit programmiert werden, mit denen jedes den Wert '1' bzw. '0' annimmt. Nachdem die Wahrscheinlichkeit programmiert wurde, erzeugen die Zufallsgeneratoren eine beliebige Anzahl an Zufallsbits, diese werden in den meisten Fällen verwendet, um zu entscheiden, ob eine Mutation auftritt oder nicht.

- Ein Maskenbit
  Das Maskenbit wird über eine Lauflängenkodierung erzeugt. Dabei wird jeweils die erwünschte Anzahl an Bits mit dem Wert '1' (bzw. '0') als Integerzahl `n` an den

Lauflängendekodierer gesendet, dieser erzeugt mit einem Zähler einen Bitstrom der Länge n mit jeweils dem Wert '1' (bzw. '0'). Pro Befehl an den Lauflängendekodierer werden also n Maskenbits erzeugt. Mit dem Maskenbit wird i.A. der Kreuzungsoperator betrieben, die Integerzahl n bezeichnet dann einen Kreuzungspunkt.

- Zwei genabhängige Bits A und B
  Jedes Gen hat eine Genauigkeit von 11 Bit, belegt im Speicher aber 2 Bytes. Von den freien 5 Bit werden 2 Bit als Steuerbits verwendet. Mit diesen Bits können die genetischen Operatoren spezifisch für einzelne Gene formuliert werden.

Die Zufallsbits A und B werden über die Programmierung der Wahrscheinlichkeit von der Software kontrolliert, ebenso werden die Maskenbits mit der Lauflängenkodierung über Softwarebefehle bestimmt. Durch die 4fach parallele Pipeline werden pro Takt je 4 Steuerbits, d.h. insgesamt 20 Steuerbits, benötigt. Obwohl der Dekodierer maximal einen Befehl pro Takt ausführen kann, ist es möglich, die Steuerbits mit der nötigen Rate zu erzeugen, da ein Befehl im Dekodierer beliebig viele Zufallsbits bzw. n (s.o.) Maskenbits erzeugt. Die genabhängigen Bit werden ohnehin den Elterngenen entnommen und sind ohne Interaktion des Dekodierers gültig.

Der Zusammenhang zwischen Steuerbits, Multiplexer-Kontrollbits und der Pipelinesteuerung ist in Abb. 3.16 dargestellt: Die Pipeline bearbeitet das genetische Material, die Multiplexer-Kontrollbits steuern die Pipeline, sie sind aber selber von den Steuerbits abhängig. Diese Abhängigkeit nun ist durch Befehle an den evolutionären Koprozessor auch während der laufenden Evolution programmierbar. Um dieses Konzept deutlich zu machen, werden zwei Beispiele angeführt:

Ein sehr einfacher Algorithmus, der nur die gleichverteilte Mutation verwendet, setzt die Multiplexer-Kontrollbits für Kreuzung, gaußsche Mutation und Integergewicht fest auf '0'. Damit wird in Abb. 3.15 jeweils der linke Eingang der Multiplexer durchgeschaltet. Das Multiplexer-Kontrollbit der gleichverteilten Mutation wird mit dem Zufallsbit A gesteuert. Die Abhängigkeiten fasst Tab. 3.7 zusammen.

Steuerbits	erzeugen	Multiplexer-Kontrollbits
'0'	=>	Kreuzung
Zufallsbit A	=>	gleichverteilte Mutation
'0'	=>	gaußsche Mutation
'0'	=>	Integergewicht

Tabelle 3.7: Diese Programmierung der Multiplexer-Kontrollbits erzeugt einen Ablauf, der weder Kreuzungsoperator noch gaußsche Mutation noch den Spezialoperator für Integerneuronen verwendet, die entsprechenden Multiplexer-Kontrollbits werden identisch '0' gesetzt. Die gleichverteilte Mutation wird über das Steuerbit "Zufallsbit A" gesteuert.

Eine komplexere Beziehung zwischen Steuerbits und Multiplexer-Kontrollbits ist in Tab. 3.8 dargestellt. Durch diese Einstellung bestimmt das Maskenbit den Kreuzungsoperator, eine gleichverteilte Mutation findet statt, wenn das Zufallsbit A den Wert '1' annimmt, die gaußsche Mutation wird durch das Zufallsbit B entschieden. Ist allerdings

Steuerbits	erzeugen	Multiplexer-Kontrollbits
(Maskenbit)	=>	Kreuzung
(Zufallsbit A) & ! (Genabhängiges Bit A)	=>	gleichverteilte Mutation
(Zufallsbit B) & ! (Genabhängiges Bit A)	=>	gaußsche Mutation
(Genabhängiges Bit B)	=>	Integergewicht

Tabelle 3.8: Programmierung der Abhängigkeit zwischen Steuerbits und Multiplexer-Kontrollbit für einen komplexeren Algorithmus.

das genabhängige Bit A gesetzt ist, so wird das entsprechende Gen nicht mutiert. Ein Integergewicht wird durch das genabhängige Bit B angezeigt und entsprechend von der Pipeline bearbeitet.

Eine Übersicht der Komponenten der Pipelinesteuerung in Zusammenarbeit mit Pipeline und Dekodierer ist in Abb 3.17 dargestellt. Neben den bereits beschriebenen Mechanismen der Pipelinesteuerung werden folgende Hilfskomponenten verwendet:

- Der *Lauflängenkodierer* erzeugt das Maskenbit. Als Eingabe erwartet er die Lauflänge `n` als Integerzahl und den Bitwert `b` und formt diese Daten in einen Bitstrom um, es werden `n` Bits mit dem Wert `b` an die Pipeline gesendet.
- Die Komponente *Zufallsbitgenerator* erzeugt die Zufallsbit A und B. Es werden zwei 16 Bit breite Zufallsgeneratoren verwendet, deren Ergebnis jeweils mit der Wahrscheinlichkeit A bzw. B verglichen wird[17]. Ein Zufallsbit nimmt den Wert '1' an, wenn die Zufallszahl kleiner als die angegebene Wahrscheinlichkeit ist. Die Wahrscheinlichkeiten werden durch den Dekodierer über einen Befehl vorgegeben.
- Die gleichverteilten und gaußschen *Zufallsgeneratoren* erzeugen die Zufallsgene für die gleichverteilte bzw. gaußsche Mutation.

Das vorgestellte Konzept ermöglicht eine schnelle genweise Steuerung der einzelnen genetischen Operatoren nach programmierbaren Parametern. Die gegenwärtige Wahl der Steuerbits wie auch der Anzahl und Funktionsweise der Pipelinestufen orientiert sich an den bisher implementierten evolutionären Algorithmen [28] [30] [29]. Durch die Modularität und strikte Aufgabenteilung – die Pipeline führt nur die Berechnungen aus, die Steuersignale werden außerhalb durch spezielle Module erzeugt – sind Erweiterungen einfach in die bisherige Struktur zu integrieren.

3.4.4 Dekodierer

In den vorherigen Abschnitten wurden die Komponenten Pipeline und Pipelinesteuerung erklärt. Die Pipeline führt genetischen Operatoren auf den Gewichtsdaten aus, die Hauptaufgabe der Pipelinesteuerung ist die Erzeugung der Multiplexer-Kontrollbits in jedem Takt. Die Parameter und Steuersignale zu diesen beiden Komponenten werden durch den Dekodierer, der nun beschrieben wird, vorgegeben.

[17] Dabei wird die Wahrscheinlichkeit nicht in Prozent, sondern bezüglich des Maximalwertes der Zufallsgeneratoren angegeben, also als 16 Bit vorzeichenloser Integerwert im Bereich [0, 65535].

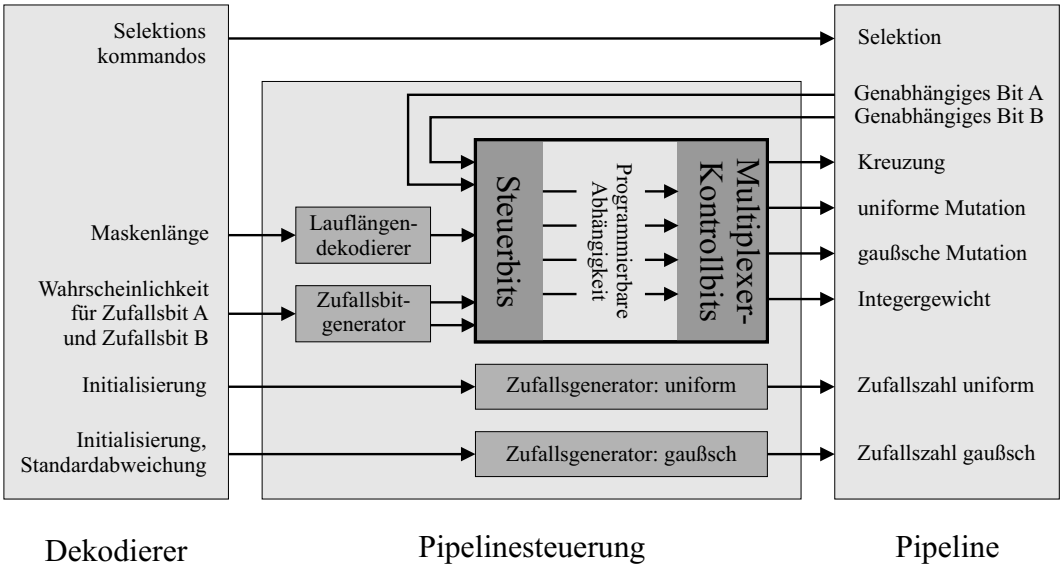

Abbildung 3.17: Schematische Übersicht der Interaktion zwischen Dekodierer, Pipelinesteuerung und Pipeline. Der Dekodierer sendet die Selektionskommandos direkt an die Pipeline. Die Multiplexer-Kontrollbits und auch die Zufallszahlen dagegen werden in der Komponente Pipelinesteuerung erzeugt.

Der Dekodierer verbindet den evolutionären Koprozessor mit dem Hauptprozessor und damit mit der Trainingssoftware. Es gibt zwei Kommunikationskanäle zum Hauptprozessor, die in Abb. 3.14 eingezeichnet sind: Einerseits werden Befehle und mehrfach benötigte Daten von der Software in den Befehlsspeicher geschrieben und können vom Dekodierer von dort gelesen werden. Andererseits kann die Software über den Slow-Control-Klient den Dekodierer starten und Statusinformationen abfragen. Dabei wird der größte Teil des Datentransfers zwischen Software und dem evolutionären Koprozessor über den Befehlsspeicher abgewickelt. Der Befehlssatz lässt sich in 3 Gruppen von Befehlen unterteilen:

- Selektionsbefehle
  Die Selektionsbefehle erzeugen ein Selektionskommando, das an die Pipeline geschickt wird. Unterschieden wird zwischen Befehlen für Eltern oder Nachkomme, und ob der Befehl in einer Schleife oder einzeln gegeben wird.

- Programmflussbefehle
  Die Programmflussbefehle steuern die Schleifenfunktionalität und beschreiben interne Register des Dekodierers.

- Befehle zum Erzeugen der Steuerbits
  Gemäß Abb. 3.17 schreibt der Dekodierer die Signale zur Erzeugung der Steuerbits, d.h. schreibt die Maskenlängen an den Lauflängendekodierer, legt die Kreuzungspunkte fest und setzt die Wahrscheinlichkeiten für die Zufallsbits.

Die Kodierung der Befehle und Operanden, eine genaue Beschreibung ihrer Funktionsweise und die Anzahl der nötigen Takte, die der Dekodierer zur Bearbeitung benötigt, ist in Anhang C.1 zu finden.

Befehl(Operand)	zur Dekodierung benötige Anzahl Takte
`mask 0(maskLengthA)`	1
`mask 1(maskLengthB)`	1
`parent1(iB_AdrP1)`	4
`parent2(iB_AdrP2)`	4
`offspring(iB_AdrO)`	4

Tabelle 3.9: Befehlssequenz zur Berechnung eines Chromosoms mit Kreuzungsoperator. Dabei wird vorausgesetzt, dass sowohl die Wahrscheinlichkeiten der Zufallsbits, als auch die Abhängigkeit zwischen Steuerbits und Multiplexer-Kontrollbits bereits programmiert sind.

Selektionsbefehle Für die Selektionsbefehle wird ein indirektes Adressierungsschema verwendet. Ein Selektionskommando, das der Dekodierer an die Pipeline schickt, hat das Format `(Speicheradresse, Anzahl Gene)`, dem Dekodierer muss also die Speicheradresse und die Anzahl der Gene mitgeteilt werden. Diese Werte sind aber nicht im Selektionsbefehl kodiert, stattdessen werden die Startadressen und Längen aller verwendeten Chromosomen im Datenteil des Befehlsspeichers abgelegt. Jeder Selektionsbefehl verweist nur auf die entsprechende Speicherstelle im Befehlsspeicher. Die indirekte Adressierung hat den Vorteil, dass die Trainingssoftware die Startadressen und Längen der Chromosomen nur einmal, und nicht mit jedem Selektionsbefehl in den Befehlsspeicher schreiben muss.

In einer Evolution werden meist viele Individuen, die jeweils die gleiche Chromosomenstruktur besitzen, nacheinander bearbeitet. Bei diesen Individuen ist die Reihenfolge, Anzahl und Länge aller Chromosomen gleich. Der Dekodierer kann ein gesamtes Individuum in einer Schleife in einem Programmdurchlauf bearbeiten. Dazu wird vor dem Start des Koprozessor die Chromosomenstruktur der Individuen im Datenteil des Befehlsspeichers abgelegt, der Dekodierer benötigt dann nur noch die Startadressen der beiden Eltern sowie die Zieladresse, an die der erzeugte Nachkomme geschrieben werden soll.

Aufbau des Dekodierers Der Dekodierer ist als Zustandsautomat implementiert. Das Zustandsdiagramm ist in Abb. 3.18 abgebildet. Der Dekodierer beginnt im Zustand `IDLE`. Gestartet wird der Dekodierer und damit der Koprozessor durch einen Slow-Control-Zugriff, der Zustandsautomat wechselt in den Zustand `READ` und liest den ersten Befehl aus dem Befehlsspeicher. Der Zustand `READ` wird nur direkt nach dem Start und nach einem Sprungbefehl angenommen. Im Zustand `DECODE` wird der aktuelle Befehl ausgeführt und gleichzeitig der nächste Befehl gelesen. Für diejenigen Befehle, die mehrere Takte zur Ausführung benötigen, schließen sich die Zustände `CROSS`, `W REG`, sowie die Zustände zur Generierung der Selektionsbefehle `SRC N` an. Diese zusätzlichen Zustände sind erforderlich, da z.B. für ein Selektionskommando wegen der indirekten Adressierung mehrfach auf den Befehlsspeicher zugegriffen werden muss.

Um ein Chromosom mit dem in Tab. 3.8 aufgeführten Einstellungen zu bearbeiten, muss der Dekodierer insgesamt drei Selektionskommandos für zwei Eltern und einen

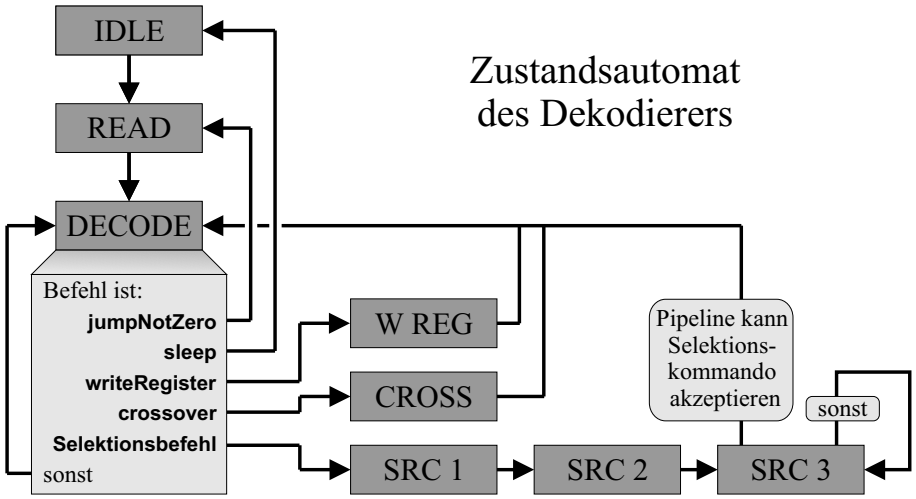

Abbildung 3.18: Zustandsautomat des Dekodierers. Im Zustand `DECODE` wird anhand des dekodierten Befehls entschieden, welcher Zustand angesprungen wird. In `SRC N` werden Selektionskommandos geschrieben, der Zustand `CROSS` schreibt Masken, `WREG` schreibt interne Register des Dekodierers. Die restlichen Befehle werden direkt im Zustand `DECODE` ausgeführt.

Nachkommen dekodieren und für den Kreuzungsoperator die Kreuzungspunkte über den Lauflängenkodierer setzen. Dies wird mit der in Tab. 3.9 angegeben Befehlssequenz erreicht. Für die Dekodierung dieser Befehlssequenz benötigt der Dekodierer insgesamt 14 Takte. Werden die Chromosomen im Schleifenmodus erzeugt, so müssen pro Chromosom zwei weitere Befehle gegeben werden, die interne Zähler inkrementieren bzw. den Sprungbefehl ausführen.

Geschwindigkeit des Dekodierers Der Dekodierer ist nicht darauf ausgelegt worden, mit hoher Geschwindigkeit Befehle zu bearbeiten. Da die Hauptarbeit, nämlich die Berechnung der genetischen Operatoren in der Pipeline und die genweise Steuerung dieser Berechnungen in der Pipelinesteuerung erfolgt, muss der Dekodierer nur schnell genug sein, um die Pipelinesteuerung mit den nötigen Daten und Signale zu versorgen. Zur Verdeutlichung wird die Berechnung eines Chromosoms für den HAGEN Chip betrachtet: Die 4fach-parallel Pipeline benötigt für ein Chromosom, das aus 128 Genen besteht, mindestens 32 Takte. Der Dekodierer muss also nur schnell genug sein, um die nötigen Befehle für ein Chromosom in 32 Takten zu bearbeiten. Die in Tab. 3.9 vorgestellte Befehlssequenz kann in 14 Takten bearbeitet werden, damit ist der Dekodierer folglich schnell genug, um die Pipeline voll auszulasten, es könnten sogar noch komplexere Befehlssequenzen verarbeitet werden.

3.4.5 Befehlsspeicher

Der Befehlsspeicher speichert in der gegenwärtigen Implementation bis zu 2048 Einträge mit der Breite von 32 Bit. Dafür werden im Darkwing-System 16 der insgesamt 72 verfügbaren Blockrams, im Nathan-System aufgrund der größeren Blockrams 4 von

insgesamt 44 verfügbaren Blockrams verwendet. Die Software hat entweder über den PCI-Bus im Darkwing-System bzw. über den OCM-Kanal des PowerPCs (vgl. Abschnitt 1.3.2) auf dem Nathan-System Schreib- und Lesezugriff auf den Befehlsspeicher.

3.4.6 Software-Ansteuerung und Simulation

Um ein oder mehrere Chromosomen bzw. mit Hilfe der Schleifenfunktion ein oder mehrere Individuen durch den Koprozessor berechnen zu lassen, muss die Software zuerst den nötigen Speicher reservieren, sowohl für das genetische Material im externen Speicherbaustein als auch für die Befehle und Daten im Befehlsspeicher. Im Anschluss daran werden die Befehle und Daten für den Koprozessor in den Befehlsspeicher geschrieben und es wird über den Slow-Control Klienten das Signal an den Koprozessor übermittelt, der daraufhin mit der Bearbeitung der Befehle beginnt.

Der Zugriff auf den Koprozessor und dessen Befehlsspeicher wird durch die Software-Klasse `EvoCop` gekapselt. Sie verwaltet den Speicherplatz des Befehlsspeichers und stellt die nötigen Methoden zum Schreiben der Befehle und Daten in den Befehlsspeicher zur Verfügung. Anhang C.2 gibt eine Übersicht der Funktionalität der `EvoCop` Klasse. Der externe Speicher wird nicht exklusiv vom Koprozessor genutzt und daher zentral verwaltet, die Klassenstruktur der Software auf dieser Ebene wird in Abschnitt 3.6 beschrieben.

Zur Verifikation enthält die `EvoCop` Klasse zusätzlich eine Software Simulation des Koprozessors. Dabei wird die Struktur der programmierbaren Hardware einschließlich des Befehlsspeichers, der 4fach-parallelen Pipeline und der Zufallsgeneratoren in Software beschrieben. Damit ist es möglich, den Koprozessor in Software zu simulieren und dadurch zu verifizieren. Wenn die Zufallsgeneratoren in Software und in der programmierbaren Hardware mit den gleichen Werten initialisiert werden, so muss das neu berechnete Genmaterial von Software Simulation und Hardware exakt übereinstimmen. Zum Selbsttest des Koprozessors sind außerdem verschiedene Testläufe in der `EvoCop` Klasse implementiert.

3.4.7 Leistungsfähigkeit und Ressourcenverbrauch

Bei der Bestimmung der Leistungsfähigkeit des evolutionären Koprozessors muss in ähnlicher Weise wie bei der Speicheransteuerung zwischen Auslastung und der maximal erreichbaren Taktrate unterschieden werden:

- Die maximale Taktrate gibt an, mit welcher Frequenz sich der Evolutionäre Koprozessor betreiben lässt. Die Geschwindigkeit der Befehlsausführung ist damit direkt proportional zur maximalen Taktrate. Da der gesamte Koprozessor mit einem separaten Taktgeber gesteuert wird[18], geben die Synthesewerkzeuge die längsten Logikpfade und damit die maximale Taktrate für den Koprozessor direkt an.

- Die Auslastung bezieht sich auf die Arbeitsweise der Pipeline. Sie kann nur Daten verarbeiten, wenn die Elterngene bereits in den Ramklienten und alle Multiplexer-Kontrollbits vorrätig sind. Zusätzlich muss der Ramklient, der die Nachkommen zurück in den externen Speicher schreibt, Daten akzeptieren können. Daher arbeitet

[18] Dies ist möglich, da die Schnittstellen zu den Komponenten Slow-Control-Klient und Ramklient jeweils als Taktgrenze verwendet werden können.

Gene pro Chromosom	Chromosomen pro Individuum	Individuen	Auslastung der Pipeline	max. Datenrate [10^6 Genen/s]
64	32	64	(56,11 ± 0,20) %	296,3
128	32	64	(71,41 ± 0,18) %	377,0
256	32	64	(82,83 ± 0,09) %	437,3
348	32	64	(87,82 ± 0,07) %	463,7
512	32	64	(90,04 ± 0,03) %	475,4
1024	32	64	(93,82 ± 0,01) %	495,4
2048	32	64	(95,97 ± 0,02) %	506,7
2048	1	64	(95,88 ± 0,28) %	506,2
2048	64	64	(95,98 ± 0,01) %	506,8
2048	32	1	(95,99 ± 0,64) %	506,8
2048	32	64	(95,98 ± 0,01) %	506,8

Tabelle 3.10: Auslastung und maximale theoretische Datenrate des evolutionären Koprozessors in Abhängigkeit von der Anzahl der Gene, Chromosomen bzw. Individuen. Für die Chromosomengröße des HAGEN Chips von 128 Genen wird eine Auslastung von 71,4% erreicht. Die Daten charakterisieren die Eigenschaften des Koprozessors unter der Voraussetzung einer deutlich schneller getakteten Speicheransteuerung.

die Pipeline in der Regel nicht in jedem Takt, sondern es gehen Takte durch Wartezyklen verloren. Die Gesamtzahl an Takten setzt sich folglich aus Arbeitstakten und Wartetakten zusammen. Die Auslastung ist das Verhältnis von Arbeitstakten zur Gesamtzahl an Takten.

Die maximale Taktrate für den Koprozessor im verwendeten Virtex-II Pro FPGA ist 132 Mhz, sie wurde mit den Analysewerkzeugen des FPGA-Herstellers ermittelt. Die Bestimmung der Auslastung ist aufwendiger. Die Auslastung ist einerseits abhängig von den gewählten Gensequenzen, andererseits ist aber auch die Geschwindigkeit des externen Speichers entscheidend, weil die Wartezyklen auf die Antwort der Speicheransteuerung eine grosse Rolle spielen.

Um den Einfluss der Geschwindigkeit des Speichers auf die Auslastung zu minimieren, wurde die Taktfrequenz des Koprozessors für die folgenden Messungen gezielt auf $\frac{1}{4}$ der Frequenz der Speicheransteuerung gesetzt, dadurch arbeitet die Speicheransteuerung deutlich schneller als die Pipeline des Koprozessors und auch die Bandbreite des externen DDR-SDRAM ist ausreichend, um die Anfragen des Koprozessors ohne bandbreitenbedingte Wartezyklen zu beantworten. Unter diesen Voraussetzungen wurde die Auslastung des evolutionären Koprozessors im Schleifenmodus gemessen. Für diese Messungen wurden zuerst die zu bearbeitenden Individuen in den externen Speicher und die erforderlichen Befehlssequenzen in den Befehlsspeicher geschrieben. Daraufhin wurde in Software die Zeit gemessen, die der Koprozessor für die Bearbeitung der Befehlssequenz benötigt. Diese Zeit ergibt mit der eingestellten Taktfrequenz die Gesamtzahl an Takten. Die Arbeitstakte dagegen errechnen sich aus der Zahl der bearbeiteten Gene geteilt durch vier, da die 4fach-parallel Pipeline in jedem Takt 4 Gene bearbeitet. Jede Zeile der Tab. 3.10 gibt

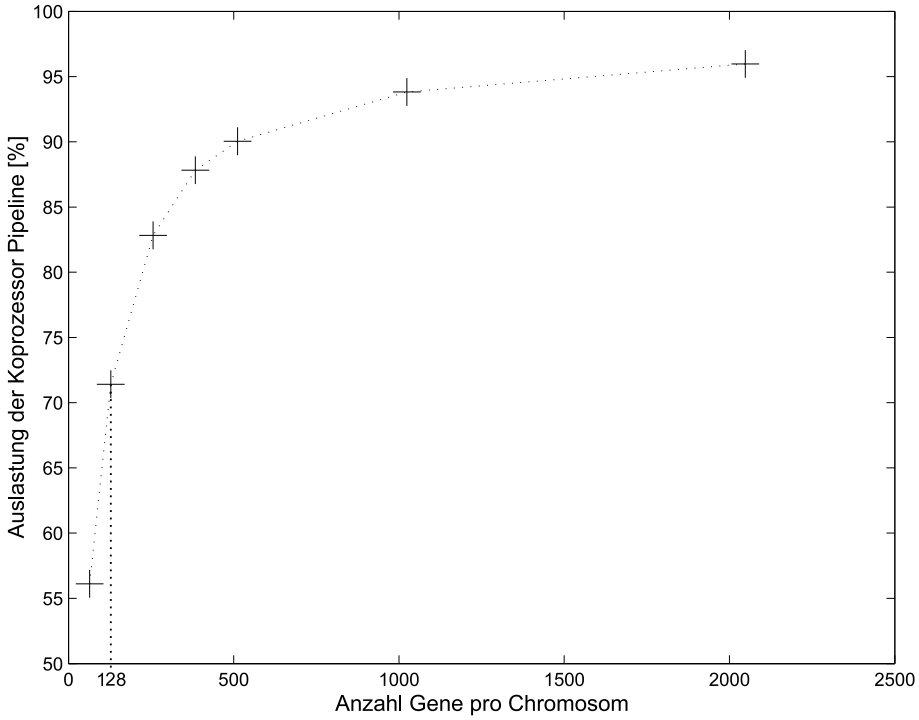

Abbildung 3.19: Abhängigkeit der Auslastung des evolutionären Koprozessors von der Anzahl an Genen pro Chromosom unter der Voraussetzung einer um den Faktor vier schneller getakteten Speicheransteuerung. Der Netzwerkchip HAGEN verwendet Chromosomen mit jeweils 128 Genen, was einer Auslastung von 71,4% entspricht.

den Mittelwert und die Standardabweichung von 100 Messungen an. Die Auslastung ist unabhängig von der Anzahl an Chromosomen und auch unabhängig von der Anzahl an Individuen. Die Abhängigkeit der Auslastung von der Anzahl der Gene pro Chromosom ist in Abb. 3.19 grafisch dargestellt.

Die Auslastung ist stark anhängig von der Chromosomengröße, d.h. der Anzahl an Genen pro Chromosom. Der Grund hierfür ist, dass der Dekodierer für jedes Chromosom Selektionsbefehle geben und auch neue Maskenbits setzten muss. Da bisher kein Prefetch der Selektionskommandos oder der Maskenlängen implementiert ist, kommt es zu Beginn jedes Chromosomes durch die Latenz der Ramklienten und des Lauflängendekodierers zu Wartezyklen, was die Auslastung herabsetzt. Dagegen hat die Anzahl an Chromosomen pro Individuum und auch die Anzahl an Individuen keinen entscheidenden Einfluss auf die Auslastung.

Die maximale Datenrate in Tab. 3.10 ergibt sich aus der maximalen Taktfrequenz des Koprozessors von 132 MHz und der jeweiligen Auslastung. Die Datenrate ist ein theoretischer Wert, der die Fähigkeiten des Koprozessors unter der Voraussetzung einer sehr schnellen Speicheransteuerung zeigt. In der vorgestellten Experimentalplattform kann die Speicheransteuerung diese Geschwindigkeit nicht erreichen, deshalb ist die tatsächliche Datenrate durch den externen Speicher begrenzt und dadurch niedriger.

Gene pro Chromosom	Chromosomen pro Individuum	Individuen	Auslastung der Pipeline	Auslastung der Speicheransteuerung
64	32	64	(39,83 ± 0,01) %	59,74 %
128	32	64	(44,04 ± 0,01) %	66,05 %
256	32	64	(45,10 ± 0,01) %	67,66 %
348	32	64	(47,00 ± 0,01) %	70,50 %
512	32	64	(46,85 ± 0,01) %	70,28 %
1024	32	64	(48,11 ± 0,01) %	72,17 %
2048	32	64	(48,41 ± 0,01) %	72,61 %

Tabelle 3.11: Messung der Auslastung des evolutionären Koprozessors und der Speicheransteuerung bei gleicher Taktfrequenz. Der Koprozessor ist durch die Speicherbandbreite limitiert und erreicht nur Auslastungen unter 50 %.

Komponente	FFs	LUTs	Blockrams
Dekodierer	375	870	0
Pipeline ohne Ramklienten	553	632	0
Pipelinesteuerung	853	732	2
Befehlsspeicher	0	0	4
Gesamt (ohne Ramklienten)	1781	2234	6
in Relation zu den vorhandenen Ressourcen	16.1%	20.0%	13.6%

Tabelle 3.12: Ressourcenverbrauch der einzelnen Module des evolutionären Koprozessors.

Für die Messungen der folgenden Tabelle 3.11 werden Koprozessor und Speicheransteuerung mit der gleichen Frequenz betrieben. Um die gleichen Bedingungen wie bei der Durchführung der evolutionären Experimente anzuwenden, werden die beiden Elternindividuen aus der gleichen Speicherbank des externen Speichers gelesen, der Nachkomme in eine zweite Speicherbank geschrieben (siehe Abschnitt 5.2). Die Speicheransteuerung überträgt die Daten mit doppelter Datenrate, kann also pro Takt 128 Bit lesen oder schreiben. Die 4fach parallele Pipeline liest in jedem Takt 4 Gene des Elternteils A, 4 Gene des Elternteils B und schreibt 4 Gene des Nachkommens. Da jedes Gen einschließlich der genabhängigen Bits 2 Bytes beansprucht, benötigt die Pipeline eine Transferrate von 192 Bits pro Takt. Dadurch ist die Auslastung der Pipeline des Koprozessor in dieser Messung durch die Speicherbandbreite begrenzt. Zur Verdeutlichung ist in Tab. 3.11 zusätzlich die jeweils erreichte Auslastung der Speicheransteuerung angegeben (vgl. Abschnitt 3.1.5). Da jeweils 2/3 der Zugriffe aus einer Speicherbank und 1/3 der Zugriffe exklusiv aus einer zweiten Speicherbank erfolgen, liegt die Auslastung der Speicheransteuerung ab 128 Genen pro Chromosom zwischen den in Abschnitt 3.1.5 gemessenen Werten.

Den Ressourcenverbrauch des evolutionären Koprozessors gibt Tab. 3.12 an, der gesamte Koprozessor benötigt etwa 16% der FFs und 20% der LUTs. Dabei sind die Ramklienten nicht mit einbezogen, da diese bei der Speicheransteuerung gewertet wurden. Insgesamt wird folglich etwa ein Fünftel der auf dem Nathan-System vorhandenen Ressourcen für den evolutionären Koprozessor benötigt.

3.4.8 Zusammenfassung

In diesem Abschnitt wurde der evolutionäre Koprozessor vorgestellt, der nach den Vorgaben der Trainingssoftware neue Individuen mit den genetischen Standardoperatoren Mutation und Crossover erzeugt. Die 4fach parallele *Pipeline* führt die Berechnungen aus. Der *Dekodierer* liest und interpretiert die Befehle der Software und kontrolliert die Pipeline. Da das Training und die Trainingsalgorithmen Gegenstand aktueller Forschung sind, ist es wichtig, die Implementation in der programmierbaren Logik möglichst variabel und erweiterbar zu gestalten.

Diese Variabilität wird hauptsächlich gewährleistet durch die Methode der Pipelinesteuerung mit Hilfe der in Abschnitt 3.4.3 eingeführten Steuerbits und Multiplexer-Kontrollbits. Dadurch ist es ohne eine Änderung der programmierbaren Logik, also ohne eine Neukonfiguration des FPGAs möglich, wesentliche Eigenschaften der genetischen Operatoren zu ändern. Die Einführung der genabhängigen Bits erlaubt eine genweise Steuerung der Operatoren. So können unterschiedliche Gene beispielsweise anhand ihrer Position im Chromosom mit unterschiedlicher Wahrscheinlichkeit oder gar nicht mutiert werden. Eine weitere Ebene der Variabilität und Erweiterbarkeit wird möglich, wenn die programmierbare Logik geändert wird. Durch den einfachen, modularen Aufbau des Koprozessors lassen sich durch Erweiterung des Befehlssatzes und evtl. weiteren Pipelinestufen nahezu beliebige Berechnungen auf der Basis von einzelnen oder aufeinander folgenden Genen einbauen. Die Unterstützung für Integergewichte in der 4. Pipelinestufe ist ein Beispiel für eine derartige Erweiterung, die benachbarte Gene miteinander verknüpft.

Die erreichte Auslastung der Datenpipeline des Koprozessors ist einerseits abhängig von der Anzahl an Genen pro Chromosom, andererseits abhängig von der verfügbaren Speicherbandbreite. Die Auslastung der Datenpipeline bleibt wegen der begrenzten Speicherbandbreite unter 50 %, wenn Speicheransteuerung und Koprozessor mit der gleichen Frequenz betrieben. Da in der vorgestellten Experimentalplattform die Speicheransteuerung eine maximale Frequenz von 140 MHz erreicht und der Koprozessor mit bis zu 132 MHz betrieben werden kann, tritt dieser Fall ein. Um Ressourcen zu sparen, könnte also die Anzahl der parallelen Pipelines von vier auf zwei reduziert werden, um den Koprozessor an die verfügbare Speicherbandbreite anzupassen.

Der gegenwärtig Funktionsumfang entspricht den Anforderungen der aktuell verwendeten Trainingsalgorithmen, eine Weiterentwicklung und Erweiterung des Funktionsumfangs wird sich an den Erfordernissen zukünftiger Algorithmen orientieren und ist aufgrund der modularen Struktur leicht zu implementieren.

3.5 Software-Hardware-Kommunikation

Da die Steuerung der Trainingsalgorithmen in Software, die Durchführung aber teilweise durch den gemischt analog-digitalen HAGEN Chip und den evolutionären Koprozessor ausgeführt wird, ist eine schnelle Kommunikation zwischen dem jeweiligen Mikroprozessor und der programmierbaren Logik wichtig. In diesem Abschnitt werden diejenigen Komponenten innerhalb des FPGAs vorgestellt, die diese Kommunikation zwischen der Trainingssoftware und der programmierbaren Logik ermöglichen.

Der Zugriff der Software auf den externen Speicher und den Befehlsspeicher des evolutionären Koprozessors muss mit hoher Datenrate erfolgen. Im Gegensatz dazu können die Kontrollzugriffe auf die HAGEN-Ansteuerung den evolutionären Koprozessor und den externen DAC-Baustein (siehe 3.2.3) langsam über die Slow-Control abgewickelt werden. Die Software muss also mit geringer Bandbreite an die Slow-Control und mit möglichst hoher Bandbreite an die Speicheransteuerung und an den Befehlsspeicher des Koprozessors angeschlossen werden. Auf dem Darkwing-System steht dafür nur der PCI-Bus zur Verfügung, im Nathan-System können dagegen der PLB, das OCM und das DCR (siehe 1.3.2) verwendet werden.

Um das Nathan-System und den eingebetteten PowerPC überhaupt vom Kontroll-PC initialisieren und ansprechen zu können, sind weitere Kommunikationskanäle erforderlich, die am Endes dieses Abschnittes erläutert werden.

Eine Übersicht über die Kommunikationskette von der Software, über Linux-Treiber zu den Komponenten der programmierbaren Logik zeigt Abb. 3.21.

3.5.1 Darkwing-System

Auf dem Darkwing-System ist der PCI-Bus der einzige Kommunikationskanal zwischen der Trainingssoftware, die durch den PC Prozessor ausgeführt wird, und der programmierbaren Logik im FPGA der Darkwing-Karte. Auf der Hardwareseite wird die PCI-Spezifikation durch den PLX 9054 Baustein auf der Darkwing-Karte eingehalten (siehe Abschnitt 1.4.1). Der PLX Baustein kommuniziert mit einem vereinfachten Protokoll über einen lokalen Bus mit dem FPGA. Innerhalb der programmierbaren Logik kontrolliert die Komponente `PLX-Slave` den lokalen Bus zwischen FPGA und PLX Baustein. Die Zugriffe werden anhand der Adresse sortiert und an die verschiedenen Komponenten Speicheransteuerung, an den Befehlsspeicher des Koprozessors oder an die Slow-Control weitergeleitet. Sowohl der Zugriff auf den externen Speicher als auch auf den Befehlsspeicher des Koprozessors sind burstfähig, d.h. es können beliebig große Blöcke am Stück übertragen werden. Dies ist vor allem für die Software wichtig. Durch die Burstfähigkeit kann auf der Seite des PCs *Direct Memory Access* (DMA) [53] verwendet werden, wodurch vor allem Lesezugriffe stark beschleunigt werden.

3.5.2 Nathan-System

Auf dem Nathan-System stehen drei Kanäle zur Kommunikation zwischen dem eingebetteten PowerPC und der programmierbaren Logik zur Verfügung (vgl. Abschnitt 1.3.2): Das *On-Chip-Memory* (OCM) ermöglicht hohe Transferraten, setzt aber eine feste Antwortzeit voraus und wird deshalb an Blockrams angeschlossen. Der *Processor Local Bus* (PLB) setzt keine festen Antwortzeiten voraus und erreicht ebenso hohe Transferraten,

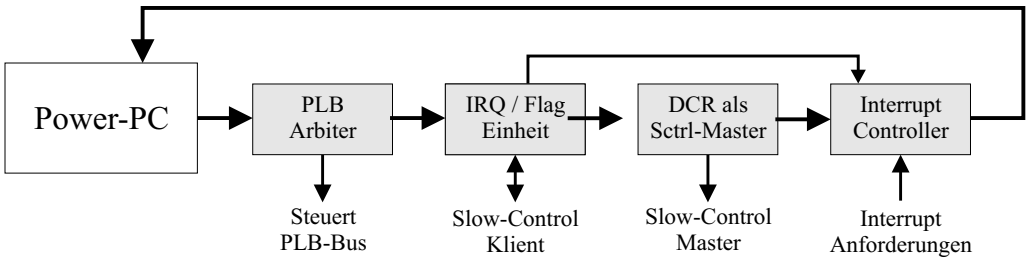

Abbildung 3.20: Verkettung der Komponenten, die über das DCR angesprochen werden

er wird über einen Ramklienten mit der Speicheransteuerung und damit an den externen Speicher angeschlossen. Das *Device Control Register* (DCR) liest und schreibt Kontrollregister mit geringer Geschwindigkeit. Über den DCR-Kanal wird der lokale Slow-Control Zugriff realisiert. Zusätzlich können über das DCR Status Register (Flags) und Interrupts für den PowerPC geschrieben, gelesen bzw. gelöscht werden. Die Komponenten der programmierbaren Logik werden nun im einzelnen beschrieben.

Processor Local Bus (PLB)

Die Komponente `plb-ramklient` verknüpft die PLB Schnittstelle mit einem RamklientenSie unterstützt das Lesen und Schreiben ganzer Cachelines, also von Blöcken der Größe 32 Bytes, d.h. es werden immer vier konsekutive Zugriffe von jeweils 8 Byte Größe an den Ramklienten gestellt. Zusätzlich erlaubt die PLB-Spezifikation [14] die Stapelung von bis zu zwei Zugriffsanforderungen. Dadurch wird die Antwortzeit der Speicheransteuerung teilweise versteckt. Der gesamte PLB-Bus besteht aus 2 PLB-Mastern, die beide der Prozessorkern des PowerPC stellt und einem PLB-Klienten, der über einen Ramklienten an die Speicheransteuerung angeschlossen ist. Der PowerPC verwendet einen Master zum Schreiben und Lesen von Daten, der zweite Master wird nur zum Lesen von Instruktionen benötigt. Details zur Die Komponente `plb-ramklient` finden sich in [58].

On-Chip-Memory (OCM)

Ähnlich der beiden PLB-Master gibt es zwei Arten von OCM Zugriffen, einerseits zum Lesen von Instruktionen, den IS-OCM, andererseits zum Schreiben und Lesen von Daten, den DS-OCM. Im Gegensatz zu den zwei PLB-Mastern sind beim OCM diese beiden Kanäle komplett getrennt und mit verschiedenen Blockrams verbunden. Während der Boot Phase des PowerPC Linux werden beide OCM Kanäle zur Ausführung des Bootloaders verwendet. Aus dem IS-OCM wird der Programmcode gelesen, während der DS-OCM zum Schreiben und Lesen von Daten, Registern und für den Stack benutzt wird. Nach dem erfolgreichen Linux Start wird das DS-OCM zum Schreiben der Befehle für den Koprozessor verwendet, die angeschlossenen Blockrams stellen dann den Befehlsspeicher dar.

Komponente	FFs	LUTs	Blockrams
`plb-ramklient`	69	221	0
`IRQ-Flag-Einheit`	26	111	0
`DCR-Sctrl-Master`	145	130	0
Zwischenregister DCR-Bus	84	1	0
PLB-Arbiter und Interrupt-Controller	266	833	0
Gesamte PowerPC-Ansteuerung	590	1296	0
in Relation zu den vorhandenen Ressourcen	5.3%	11.7%	0.0%

Tabelle 3.13: Ressourcenverbrauch der PowerPC-Ansteuerung

Device-Control Register (DCR)

Über das DCR werden insgesamt vier Komponenten angeschlossen: Von der Firma Xilinx wird ein Arbiter für den PLB-Bus sowie eine Interrupt-Kontroll-Einheit bereitgestellt [82]. Zusätzlich wurden zwei Komponenten entwickelt, die den PowerPC an die Slow-Control anschließen: Über die Komponente `IRQ-Flag-Einheit` wird der DCR und damit der PowerPC mit einem Slow-Control-Klienten verbunden. Dadurch können Statusregister (Flags) sowohl vom Kontroll-PC, als auch vom PowerPC geschrieben und gelesen, also ausgetauscht werden. Diese Komponente wird unter anderem verwendet, um eine Kommunikation zwischen dem Kontroll-PC und dem eingebetteten PowerPC herzustellen (siehe Abschnitt 3.6.2). Die Komponente `DCR-Sctrl-Master` aus Abb. 3.20 schließt den DCR als lokalen Slow-Control-Master an die Slow-Control an. Damit kann die Software des PowerPC gemäß Abb. 3.7 auf alle Slow-Control-Module des lokalen Nathan Netzwerkmodules zugreifen.

Ressourcenverbrauch

Die in diesem Kapitel beschriebenen Komponenten, die den PowerPC an die programmierbare Logik anschließen, benötigen zusammen 590 FFs und 1297 LUTs, wie Tab. 3.13 zeigt.

3.6 Software

Um die beiden vorgestellten Experimentalplattformen zu betreiben und Experimente mit hardwarebasierten neuronalen Netzwerken durchzuführen, wurde in der Electronic Vision(s) Group das Softwarepaket HANNEE (*Heidelberg Analog Neural Network Evolution Environment*) entwickelt [20]. Ursprünglich wurde HANNEE für das Darkwing-System und den PC-Prozessor entwickelt. Die gegenwärtige Softwareplattform ist Linux mit dem Kernel 2.4 und einem GCC 3.3 Compiler [23]. Um die bestehende Software aber mit minimalem Aufwand auch auf andere Plattformen und Betriebssysteme, wie zum Beispiel den eingebetteten PowerPC im Nathan-System portieren zu können, wurden spezielle Vorkehrungen getroffen:

- Bis auf wenige Ausnahmen (s.u.) wurden nur freie Software-Bibliotheken eingebunden, die auf allen Plattformen, die den GCC unterstützen, kompiliert werden können. Ein Beispiel ist die ACE [1] Bibliothek für Multi-Thread-Anwendungen.
- Die grafische Benutzerschnittstelle verwendet QT der Firma Trolltech [77]. QT kann sowohl für Windows als auch für Linux kompiliert werden. Um auch ohne grafische Unterstützung arbeiten zu können, wurde eine Kommandozeilenschnittstelle entwickelt [18].
- Der Zugriff der Software auf die programmierbare Logik ist abhängig von der Hardwareplattform, sei es nun das Darkwing-System, das Nathan-System oder zukünftige Hardwaresysteme. Um Programme, Anwendungen und auch die Trainingsalgorithmen unabhängig von der aktuellen Hardwareimplementation zu entwickeln, wurde eine *Hardware-Abstraktionsebene* (HAE) eingeführt. Die HAE kapselt die Zugriffe auf die programmierbare Logik und stellt der restlichen Software eine einheitliche, plattformunabhängige Schnittstelle zur Verfügung.

Im Nathan-System greift die Software auf die programmierbare Logik über die in Abschnitt 3.5 beschriebenen Kommunikationskanäle PLB, OCM und DCR zu. Der PowerPC erlaubt diese Zugriffe zum Teil nur aus dem privilegierten Modus, deshalb mussten Kernel-Treiber entwickelt werden, die diese Zugriffe im privilegierten Modus ausführen und eine Schnittstelle im Benutzermodus für das Softwarepaket HANNEE bereitstellen (vgl. Abschnitt 1.3.5).

Abb. 3.21 zeigt eine Übersicht über die Software und die Kommunikation mit der programmierbaren Logik. Im Folgenden werden die einzelnen Teile des Softwarepakets HANNEE und die Kernel-Treiber erläutert. Die Komponenten der programmierbaren Logik, die den Zugriff auf den externen Speicherbaustein, die Slow-Control und den Befehlsspeicher ermöglichen, wurden Abschnitt 3.5 vorgestellt.

3.6.1 Das Softwarepaket HANNEE

Das Softwarepaket HANNEE fasst die Teile der Software, die im Benutzermodus ausgeführt werden, zusammen. Die Software auf der obersten Ebene, d.h. die Entwickelung der Anwendungen und Trainingsalgorithmen und die Durchführung der Experimente ist nicht Thema dieser Arbeit. Innerhalb der Electronic Vision(s) Group am Kirchhoff-Institut

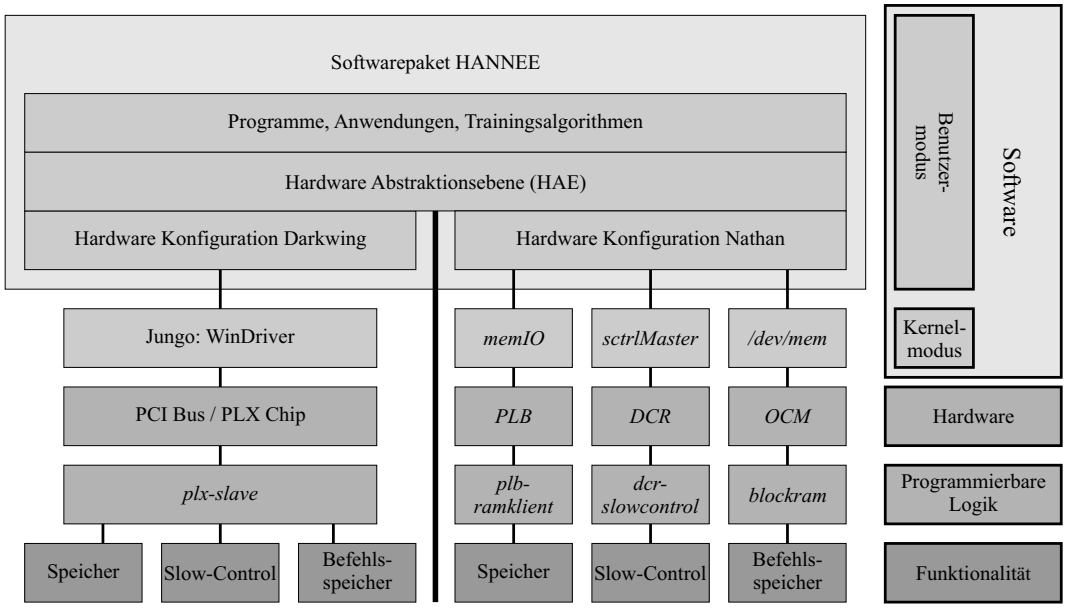

Abbildung 3.21: Übersicht über die Software und die Kommunikationskanäle zwischen Software programmierbarer Logik. Die Programme, Anwendungen und Trainingsalgorithmen können durch die Kapselung der Hardwarezugriffe in der HAE plattformunabhängig formuliert werden. Der Hardwarezugriff im Darkwing-System nutzt das Treibermodul WinDriver, für das Nathan-System wurden eigene Treiber entwickelt.

für Physik in Heidelberg wurden aber verschiedene Ansätze verfolgt, um den neuronalen Netzwerkchip HAGEN zu trainieren. Neben evolutionären Trainingsalgorithmen wurde eine Adaption von *Liquid Computing* [41] [6] auf die Eigenschaften des HAGEN Chips [11] [68] [69] vorgenommen. Ein anderer Ansatz setzt den HAGEN Chip im Nathan-System zur Mustererkennung ein [19] [18].

Hardware-Abstraktionsebene (HAE) Die HAE kapselt den Zugriff die programmierbare Logik. Die `HNetData`-Klasse ist das zentrale `C++`-Objekt, das die Schnittstelle zwischen der HAE und den darüberliegenden Anwendungen und Trainingsalgorithmen implementiert [29]. In der gegenwärtigen Version speichert ein `HNetData`-Objekt die Gewichts-, Eingabe- und Ergebnisdaten für einen Netzwerkblock, zusätzlich werden Parameter für den analogen Betrieb des HAGEN Chips in der `HNetData`-Klasse gesichert.

Die vorgestellten Experimentalplattformen Darkwing und Nathan können in drei unterschiedlichen Konfigurationen verwendet werden:

- Darkwing-System
  Im Darkwing-System wird die Trainingssoftware auf der x86-Architektur des Host-Computers, einem Standard-PC, ausgeführt. Die programmierbare Logik und der HAGEN Chip befinden sich auf der PCI-Karte Darkwing und sind über den PCI-Bus erreichbar.

- Nathan-System
  Im Nathan-System wird die Trainingssoftware auf dem eingebetteten PowerPC des

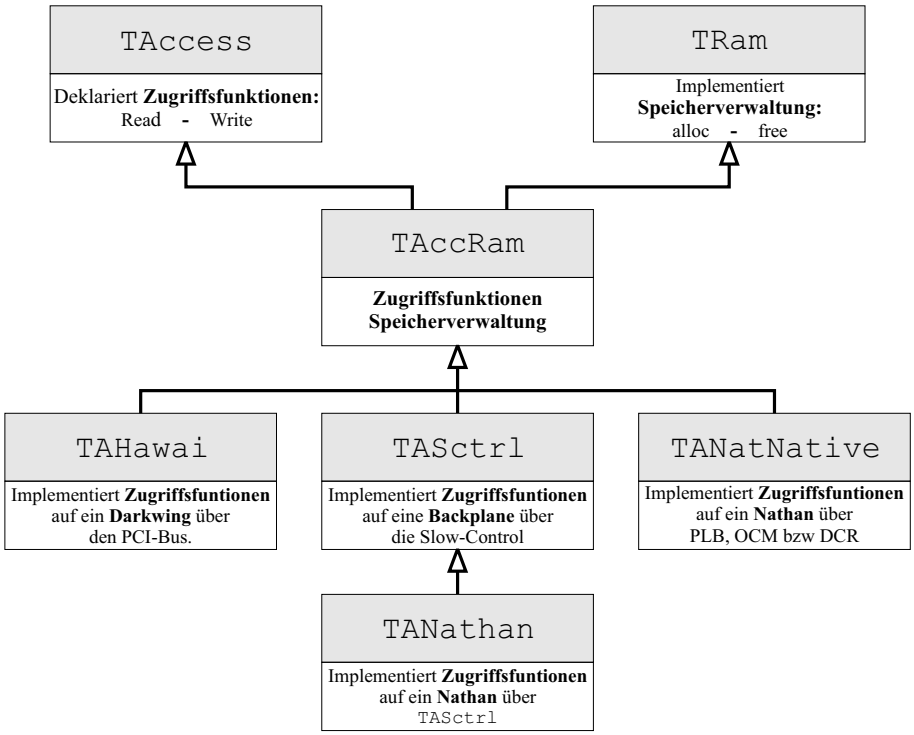

Abbildung 3.22: Kapselung der plattformspezifischen Details durch die TAccess-Klassen der HAE.

FPGAs ausgeführt. Die programmierbare Logik ist über die in Abschnitt 3.5 beschriebenen Kommunikationskanäle PLB, OCM und DCR erreichbar.

- Kombiniertes System
  Zu Testzwecken wird noch eine dritte Konfiguration unterstützt. Die Software wird durch den Kontroll-PC ausgeführt, während die programmierbare Logik und auch der HAGEN Chip eines Nathan Netzwerkmodules verwendet wird. Alle Zugriffe auf die Hardware werden dabei vom Kontroll-PC zum Nathan über die Slow-Control übertragen, daher ist diese Methode sehr langsam und wird nur zu Testzwecken verwendet.

Die in dieser Arbeit weiterentwickelte Klassenstruktur der HAE, die die Zugriffe auf die Hardware kapselt, orientiert sich an der vorhandenen Struktur der Hardware. Sie ist in Abb. 3.22 dargestellt. Die Klasse `TAccess` definiert allgemeine Zugriffsfunktionen als Schreiben und Lesen auf bzw. von einer Adresse. Die Klasse `TRam` verwaltet Speicher, es können Speicherbereiche beliebiger Größe reserviert bzw. freigegeben werden. `TRam` wird beispielsweise verwendet, um den externen Speicher und auch den Befehlsspeicher für den evolutionären Koprozessor zu verwalten. Die Klasse `TAccRam` kombiniert diese beiden Eigenschaften. Die folgenden Klassen in Abb. 3.22 verwalten jeweils eine der drei Zugriffsarten auf die Hardware und implementieren die in `TAccess` definierten Zugriffsfunktionen: `TAHawai` spricht vom PC eine PCI-Karte Darkwing an. Die Klassen `TASctrl` und `TANathan` werden benutzt, um ein Nathan Netzwerkmodul vom Kontroll-PC über die Slow-Control

zu steuern und `TANatNative` schließlich verwaltet vom eingebetteten PowerPC lokal die Ressourcen eines Nathan Netzwerkmodules.

Durch diese Klassenstruktur können die in `TAccess` definierten Zugriffsfunktionen verwendet werden, um Slow-Control-Module, den externen Speicher oder den Befehlsspeicher plattformunabhängig anzusprechen. Durch Instantiierung entweder von `TAHawai`, `TANatNative` oder `TANathan` wird entschieden, auf welcher Plattform die Zugriffe ausgeführt werden.

3.6.2 Kerneltreiber

Um die programmierbare Logik mit der HAE der HANNEE-Software ansprechen zu können, müssen im Darkwing-System der PCI-Bus, im Nathan-System die Kommunikationskanäle PLB, OCM und DCR verwendet werden. Dies ist nur im Kernel-Modus möglich (siehe Abschnitt 1.3.5). In diesem Abschnitt werden die Kernel-Treiber beschrieben, die diese I/O-Funktionen für die restliche Software, die im Benutzermodus ausgeführt wird, kapseln.
Die Linux-Portierung, d.h. die Kompilierung des Kernels auf die Architektur des PowerPCs, die Erstellung einer Ramdisk und die Konfiguration des Bootloaders und der Bootvorgang [9] ist in [75] [55] dokumentiert.

Darkwing-System

Der Zugriff auf den FPGA im Darkwing-System verläuft über den PCI-Bus. Die PCI-Schnittstelle auf der Darkwing-Karte wird durch den PCI-Bridge-Baustein PLX PCI 9054 der Firma PLX [33] versorgt. Unter anderem für die Kommunikation mit PCI-Bridge-Bausteinen der Firma PLX bietet die Firma Jungo Ltd. [39] das Treiberpaket WinDriver 6.x [40] an. Das Paket WinDriver stellt sowohl einen Kernel-Treiber für alle größeren Plattformen als auch Beispielquellcode der Ansteuerungssoftware zur Verfügung und wird im Darkwing-System für die Kommunikation zwischen Software und der PCI-Karte Darkwing verwendet.

Nathan-System

Für das Nathan-System wurden eigene Komponenten entwickelt, um die Kommunikationskanäle zwischen Prozessorkern und der programmierbaren Logik zu verwenden (vgl. Abschnitt 3.5). Zur Ansteuerung dieser proprietären Schnittstellen mussten daher auch eigene Kernel-Treiber geschrieben werden, die im Folgenden beschrieben werden.

Direkter Speicherzugriff: `memIo` Um große Datenblöcke zwischen Software und der programmierbaren Logik auszutauschen, wird die Speicheransteuerung und der externe Speicherbaustein verwendet. Sowohl die HAGEN-Ansteuerung als auch der evolutionäre Koprozessor beziehen ihre Daten – sei es genetisches Material oder Eingabe- bzw. Ergebnisdaten für den HAGEN Chip – aus dem externen Speicher.

Die Software ist auch in der Lage, über den PLB-Bus auf den externen Speicher zuzugreifen. Der für das Linux Betriebssystem reservierte Speicher ist aber in Stücken der *pagesize*[19] fragmentiert und wird zudem durch das Betriebssystem, bzw. die *Memory Ma-*

[19] Eine page hat eine Größe von 4kB.

nagement Unit (MMU) verwaltet. Für eine detaillierte Übersicht der Speicherhierachie sei auf [26] verwiesen. Daher kann dieser Speicherbereich nicht als konsekutiver Speicher für die Übertragung großer Blöcke verwendet werden. Der verfügbare physikalische Speicher wird daher aufgeteilt: Die untere Hälfte wird durch das Betriebssystem verwaltet, während die obere Hälfte für die Kommunikation zwischen Software und programmierbarer Logik reserviert ist, bzw. exklusiv von der programmierbaren Hardware genutzt wird[20]. Das Kernel-Modul `memIo` macht Teile dieses Speicherbereichs für die Software im Benutzermodus sichtbar. Der Treiber wird aus dem Benutzermodus aufgerufen, dabei wird ein physikalischer Speicherbereich mittels Startadresse und Größe vorgeben. Dieser Speicherbereich wird durch den Treiber in den virtuellen Adressraum des aufrufenden Prozesses abgebildet ("gemappt"). Daraufhin kann die Software diesen selbst gewählten Speicherbereich aus dem Benutzermodus mit einer einfachen Zeigeroperation ansprechen.

Register und Kontrollzugriff: `sctrlMaster` Über das DCR kann der PowerPC als Slow-Control-Master alle Slow-Control-Klienten des lokalen Nathan Netzwerkmodules ansprechen. Der Treiber `sctrlMaster` stellt der Software im Benutzermodus die nötige Funktionalität zur Verfügung. Das Schreiben auf den Slow-Control-Master ist mittels `ioctl`-Kommandos realisiert, das Lesen wird über das normale `read`-Kommando abgewickelt (vgl. [62]).

Zugriff auf den Befehlsspeicher Der Befehlsspeicher wird über den DS-OCM angesprochen. Der OCM wird nicht durch den Cache und die MMU verwaltet, er kann aus dem Benutzermodus angesprochen werden. Dazu wird die zeichenbasierte Gerätedatei `/dev/mem`, die eine Abbildung des Arbeitsspeichers des Computers enthält, verwendet und die Adresse des DS-OCM in den Benutzermodus abgebildet [62].

IP-Netzwerkschnittstelle: `netPC` Bisher wurden die Kommunikationswege zwischen der Software und der programmierbaren Logik auf beiden Systemen gemäß Abb. 3.21 beschrieben. In diesem Abschnitt wird nun erläutert, wie das eingebettete Linux, das auf den Nathan Netzwerkmodulen ausgeführt wird, angesteuert wird.

Zwischen dem eingebetteten Linux und dem Kontroll-PC gibt es zwei Kanäle, auf denen Daten ausgetauscht werden können: Die Komponente `IRQ-Flag-Einheit` implementiert Register und Flags, die sowohl vom Kontroll-PC als auch vom PowerPC geschrieben und gelesen werden können, zusätzlich kann mit Hilfe dieser Flags ein Interrupt für den PowerPC ausgelöst werden. Der zweite Kommunikationskanal zwischen Kontroll-PC und PowerPC ist der externe Speicher: Der PC kann über die Slow-Control und einen Ramklienten auf den externen Speicher jedes Nathan Netzwerkmodules zugreifen (vgl. 3.2.3.). Der PowerPC ist über den PLB und die Komponente `plb-ramklient` ebenfalls an den externen Speicher angeschlossen. Diese beiden Kommunikationskanäle werden genutzt, um eine IP-basierte Netzwerkverbindung zwischen PC und PowerPC herzustellen. Die einzelnen IP-Pakete werden dabei über den externen Speicher ausgetauscht, die Steuerung der Schnittstelle wird aber über die Komponente `IRQ-Flag-Einheit` abgewickelt.

[20] Bei der gegenwärtigen Bestückung von 256 MB in 8 Speicherbänken stehen somit 4 Speicherbänke mit zusammen 128 MB für das Betriebssystem und eben soviel Speicher für die Kommunikation bzw. die programmierbare Logik zur Verfügung.

Der Kerneltreiber `netPC` implementiert diese Netzwerkschnittstelle, die wichtigsten Funktionen sind:

- `netPC-open`
  Öffnet die Netzwerkschnittstelle, fordert einen Interruptkanal und reserviert den externen Speicherbereich, der für die Übertragung der IP-Pakete benötigt wird[21].

- `netPC-close`
  Schließt die Netzwerkschnittstelle, gibt den Interruptkanal und das Mapping des reservierten Speicherbereichs wieder frei.

- `netPC-sendPacket`
  Wird vom Kernel aufgerufen und sendet ein IP-Packet an den PC. Das Packet wird in den externen Speicher geschrieben und der PC über ein Flag in der Komponente `IRQ-Flag-Einheit` davon unterrichtet.

- `netPC-irqHandler`
  Wird im Falle eines Interrupts aufgerufen. Der Kontroll-PC sendet zwei verschiedene Interrupts an den PowerPC. Einerseits wenn der Kontroll-PC ein vom Power PC bereitgestelltes Paket fertig gelesen hat und andererseits wenn der Kontroll-PC ein eigenes Packet gesendet, d.h. in den externen Speicher geschrieben hat. Der Interrupt Handler reagiert entsprechend mit dem Wiedereröffnen der Sendewarteschlange, bzw. durch Lesen und Weiterreichen des IP-Packets an den Kernel.

Durch diese Funktionen kann die Netzwerkschnittstelle `netPc` IP-Pakete an den Kontroll-PC senden sowie vom Kontroll-PC empfangen. Auf der Seite des Kontroll-PCs wurde die Netzwerkschnittstelle im Benutzermodus implementiert, da die WinDriver Software die Zugriffe auf die Hardware kapselt.

Mit Hilfe der Netzwerkschnittstelle `netPC` können IP-Pakete zwischen dem Kontroll-PC und dem PowerPC ausgetauscht werden. Damit steht für die weitere Kommunikation der machtvolle Apparat der Linux-Netzwerk-Werkzeuge zur Verfügung, der auf dem IP-Protokoll aufbaut [76]. In der Praxis wird die Netzwerkschnittstelle `netPC` direkt nach dem Booten des Linux-Kernels von der Ramdisk aus gestartet. Danach kann man sich über `rlogin` auf jedem Nathan Netzwerkmodul einwählen, die Programme der Ramdisk aufrufen, oder über einen smb-mount [74] Verzeichnisse auf dem Kontroll-PC in die Ramdisk einblenden.

[21] Dafür wird, ähnlich dem `memIo`-Treiber, der obere Bereich des externen Speichers verwendet, der nicht vom Betriebssystem verwaltet wird.

Kapitel 4

Eigenschaften der vollständigen Experimentalplattform

In diesem Kapitel wird der Aufbau und die Eigenschaften des Nathan-Systems zusammengefasst und der Ressourcenverbrauch der einzelnen Komponenten gegenübergestellt. Es werden die maximalen Taktraten und die während der Experimente verwendeten Taktraten angegeben und das Nathan-System mit ähnlichen Implementationen genetischer Algorithmen in Hardware verglichen.

4.1 Aufbau

Um Experimente mit evolutionären Algorithmen auf dem Nathan-System durchzuführen werden alle in Kapitel 3 beschriebenen Komponenten benötigt. Eine genaue Übersicht der programmierbaren Logik zeigt Abb. 4.1. Der Aufbau sowie die Arbeitsweise wird im Folgenden erläutert.

4.1.1 Infrastruktur

Die *Speicheransteuerung* und die *Slow-Control* stellen die Infrastruktur im FPGA zur Verfügung. Die Speicheransteuerung ist darauf ausgelegt, Datenströme von verschiedenen unabhängigen Ramklienten simultan mit hoher Geschwindigkeit in den externen Speicher zu schreiben und von dort zu lesen. Wie in Abb. 4.1 dargestellt ist, wird dies im vollständigen Design massiv genutzt. Der evolutionäre Koprozessor und die HAGEN-Ansteuerung nutzen jeweils drei Ramklienten, der eingebettete PowerPC und auch die Slow-Control jeweils einen. Da die Speicheransteuerung prinzipiell beliebig viele Ramklienten unterstützt[1], können Komponenten ohne Aufwand entfernt oder durch neue ersetzt werden. Beispielsweise kann für nicht-evolutionäre Trainingsalgorithmen der Koprozessor ersetzt werden, oder für die nächste Generation des Netzwerkchips kann die HAGEN-Ansteuerung durch eine neue Schnittstellenkomponente ausgetauscht werden.

Die Schnittstellen des PowerPCs PLB, OCM und DCR sind mit der programmierbaren Logik verbunden, neben der Anbindung des externen Speichers über den PLB kann der

[1] Mit steigender Anzahl an Ramklienten steigt natürlich der Ressourcenverbrauch und vor allem der Routingaufwand, so dass die maximale Taktrate bei wachsender Anzahl an Ramklienten sinkt.

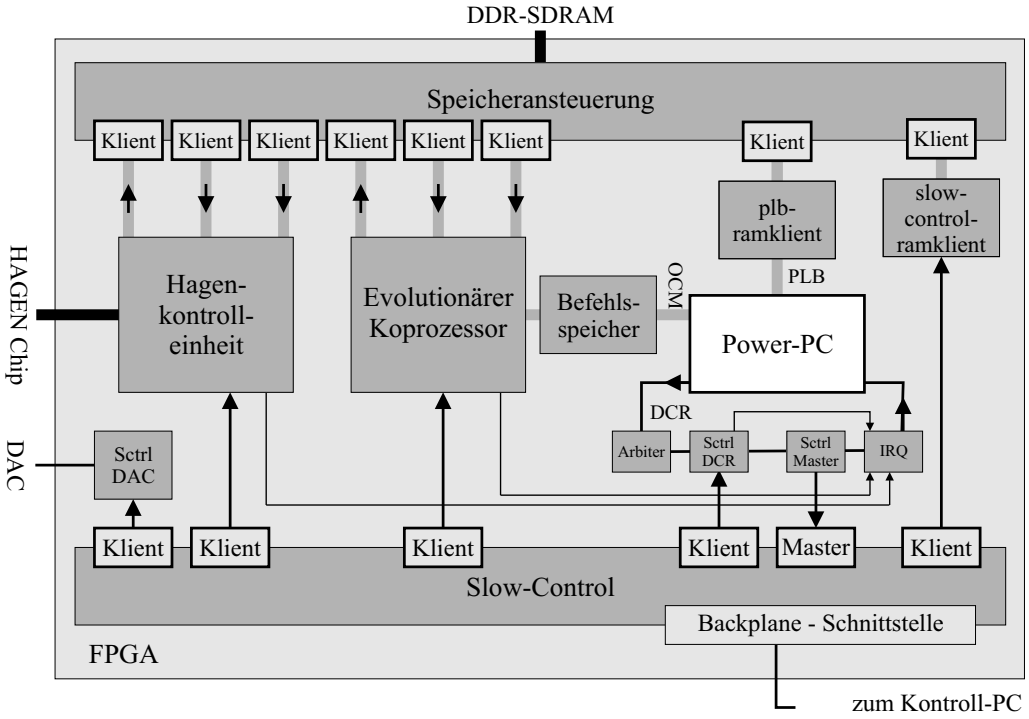

Abbildung 4.1: Übersicht über die programmierbare Logik im FPGA des Nathan-Systems

Befehlsspeicher des Koprozessors direkt über den OCM angesprochen werden. Der DCR steuert die beiden von Xilinx bereitgestellten Komponenten Arbiter und IRQ-Einheit und stellt die Verbindung zur Slow-Control her.

4.1.2 Arbeitsaufteilung

Die Aufteilung der Arbeitsschritte in Hard- und Software in der vorgestellten Experimentalplattform orientiert sich an den Anforderungen des evolutionären Algorithmus und den Stärken von Software und paralleler Hardware. Die Berechnung der Fitness und sämtliche Kontrollaufgaben des Trainings werden in Software ausgeführt. Die Netzwerkberechnung erfolgt in analoger Hardware, die Kommunikation innerhalb des FPGAs und die Kommunikation mit den externen Bausteinen sowie die Bearbeitung des genetischen Materials wird in programmierbarer Logik implementiert.

Kontrollaufgaben in Software Die Trainingssoftware wird lokal auf dem eingebetteten PowerPC ausgeführt, durch die Portierung des Linux Kernels 2.4 steht ein vollständiges Betriebssystem zur Verfügung. Mit Hilfe speziell entwickelter Treiber kann direkt auf die programmierbare Logik zugegriffen werden. Über das DCR kann der PowerPC als Master auf die lokale Slow-Control zugreifen und erhält so Zugriff auf alle lokalen Slow-Control-Klienten.

Um Befehle in den Befehlsspeicher des Koprozessors zu schreiben, wird das OCM verwendet. Der Zugang zum Hauptspeicher wird über den PLB und einen Ramklienten

realisiert. Der PowerPC ist im Vergleich mit einem modernen PC deutlich langsamer, führt im vorgestellten System aber nur Steuerungsaufgaben und die Berechnung der Fitness anhand der Ergebnisdaten des HAGEN Chips aus.

Netzwerkberechnung und Ansteuerung der analogen Hardware Die Netzwerkberechnung erfolgt in den analogen Blöcken des HAGEN Chips. Wegen der binärer Eingabe- und Ausgabeneuronen und der Integration der DACs in den Netzwerkchip ist die Schnittstelle zwischen FPGA und HAGEN Chip rein digital (vgl. Abschnitt 1.1.3). Die HAGEN-Ansteuerung versorgt die physikalische Schnittstelle zwischen FPGA und HAGEN Chip und organisiert den Datenpfad von den Ramklienten bis in die I/O-Zellen. Durch den internen Aufbau des HAGEN Chips müssen die Eingabe-, Ergebnis- und Gewichtsdaten umsortiert werden, bevor sie an die physikalische Schnittstelle weitergegeben werden (vgl. Abschnitt 3.3.3). Diese Umsortierung erfolgt durch spezielle Sortierer, die in den Datenpfad eingesetzt werden und beeinflusst durch die Bearbeitung in der programmierbaren Logik die Ansteuerung bis auf eine erhöhte Latenz in den Datenpfaden nicht.

Erzeugung neuer Generationen durch den Koprozessor Der evolutionäre Koprozessor erzeugt neue Generationen nach den Anweisungen und Parametern der Software. Dazu wurde ein spezieller Befehlssatz definiert. Über Selektionskommandos können beliebige Gensequenzen ausgewählt werden, auch die gesonderte Behandlung von Chromosomen bis hin zu einzelnen Genen ist durch eine Reihe von Steuersignalen möglich. Der Koprozessor übernimmt also die vollständige Berechnung der Individuen einer neuen Generation. Die Software initialisiert zu Beginn der Evolution die erste Generation im externen Speicher. Während der Evolution müssen nur noch Befehle an den Koprozessor gesendet werden, weil der gesamte Transfer des genetischen Materials zwischen Koprozessor und HAGEN-Ansteuerung über den externen Speicher abläuft.

4.1.3 Parallelität

In einem HAGEN Chip arbeiten 256 Ausgabeneuronen und 32768 Synapsen parallel. Der evolutionäre Koprozessor nutzt die mögliche Parallelität der programmierbaren Logik und führt die Berechnung der neuen Individuen simultan in 4 Pipelines mit je 4 Pipelinstufen aus. Diese Parallelisierung lässt sich im Nathan-System weiter ausdehnen: Jede Backplane bietet Platz für bis zu 16 Nathan Netzwerkmodule. Diese sind über ein Token Ring Netzwerk und die Slow-Control mit einem Kontroll-PC verbunden. Über die in Abschnitt 3.6.2 vorgestellte Netzwerkschnittstelle ist es möglich, sich auf jedem der 16 Nathan Netzwerkmodule einzuloggen, Verzeichnisse auf dem Kontroll-PC zu mounten und lokal evolutionäre Experimente durchzuführen. Dabei teilt sich die Transfergeschwindigkeit der Slow-Control (vgl. Abschnitt 3.2) allerdings auf alle 16 Nathan Netzwerkmodule auf. Um Gewichts- oder Ergebnisdaten des Netzwerkchips mit hoher Geschwindigkeit zwischen verschiedenen Nathan Netzwerkmodule auszutauschen, können die seriellen Multi-Gigabit-Transceiver, die der FPGA bereitstellt (vgl. Abschnitt 1.3.2), verwendet werden. Somit können auch Experimente mit größeren Netzwerken, die sich über mehrere Nathan Netzwerkmodule erstrecken, durchgeführt werden. Auf diesen Punkt wird in Kapitel 5.4 näher eingegangen.

Komponente	FFs	LUTs	Tristates	Blockrams
Speicheransteuerung	1532	1738	546	20
Slow-Control	819	843	197	0
HAGEN-Ansteuerung	763	2253	0	7
Evolutionärer Koprozessor	1781	2234	0	6
Slow-Control-Hilfsmodule	168	98	0	0
PowerPC Anbindung	590	1296	0	4
Taktgenerierung und Resetlogik	12	19	0	0
Summe:	5665	8481	743	37
Gesamtsystem	5427	8241	645	37
in Relation zu den vorhandenen Ressourcen	48.9%	74.3%	26.2%	84.1%

Tabelle 4.1: Die Komponenten der programmierbaren Logik mit Ressourcenverbrauch in der vollständigen Experimentalplattform. Die Speicheransteuerung umfasst die Ramkontrolleinheit, den Manager und insgesamt 8 synchrone Ramklienten. Vier der 8 Ramklienten führen allerdings nur Lesezugriffe, zwei nur Schreibzugriffe aus. Der Ressourcenverbrauch des Manager wurde aus technischen Gründen mit 6 Schreib-Leseklienten bestimmt und ist damit nur eine Näherung an den tatsächlichen Verbrauch. Die Slow-Control beinhaltet die Backplane-Schnittstelle und fünf Slow-Control-Klienten. Die Slow-Control-Hilfsmodule sind die DAC-Ansteuerung `Sctrl-DAC` und die Slow-Control-Schnittstelle zur Speicheransteuerung `Sctrl-SDRAM`. Der Logik zur Ansteuerung des PowerPCs benötigt insgesamt 8 Blockrams, die allerdings nur während des Bootvorgangs durch den Bootloader verwendet werden. Vier dieser Blockrams können nach dem Bootvorgang als Befehlsspeicher wiederverwendet werden, sie werden in dieser Aufstellung deshalb unter 'Evolutionärer Koprozessor' aufgeführt.

4.2 Ressourcenverbrauch

In Kapitel 3 wurde zu jeder Komponente innerhalb des FPGAs der Ressourcenverbrauch in Registern (FFs), Funktionsgeneratoren (LUTs), Tristate-Treibern und internen Blockrams angegeben. In diesem Abschnitt werden diese Werte zusammengefasst und der Ressourcenverbrauch des Gesamtsystems ermittelt. Der Ressourcenverbrauch der einzelnen Komponenten ist in Tab. 4.1 aufgelistet. Insgesamt werden etwa 50% der verfügbaren 11088 FFs und 75% der verfügbaren 11088 LUTs verwendet. Aus der Tabelle wird klar, dass die Komponenten mehr LUTs als FFs benötigen. Damit limitieren die LUTs am ehesten die Größe des Designs bzw. den weiteren Ausbau. Die Komponenten Speicheransteuerung, HAGEN-Ansteuerung und Koprozessor verbrauchen jeweils etwa 2000 LUTs, für die Anbindung des PowerPC und die Slow-Control werden noch einmal etwa 2000 LUTs benötigt. Als Gesamtbedarf an Ressourcen ist einerseits die Summe der Einzelposten und der exakte Wert für die Synthese des Gesamtsystems angegeben. Durch Logik-Optimierungen der Synthesewerkzeuge werden FFs und LUTs im Vergleich zu den Einzelposten eingespart. Das vollständige Design belegt etwa 75 % der LUTs und etwa 50% der FFs in Relation zu den im FPGA vorhandenen Ressourcen.

4.3 Taktversorgung

Sowohl die Speicheransteuerung als auch die Slow-Control implementieren eine asynchrone Schnittstelle in jedem Ramklienten bzw. in jedem Slow-Control-Klienten. Daher können prinzipiell die Speicheransteuerung, der evolutionäre Koprozessor und auch die HAGEN-Ansteuerung durch beliebige asynchrone Taktgeber gesteuert werden. Die maximale Frequenz der einzelnen Module wurde in Kapitel 3 ermittelt:

- Die maximale Taktrate, mit der die Speicheransteuerung betrieben werden kann, ist aufgrund des steigenden Routingaufwandes abhängig von der Anzahl der verwendeten logischen Speicherschnittstellen. Werden 8 Ramklienten benutzt, so kann die Speicheransteuerung mit 120 MHz betrieben werden. Da die Daten mit doppelter Datenrate getaktet werden, entspricht dies einer Transferleistung von 240 Mbit/s pro Datenleitung oder 1,9 GB/s für den 8 Byte breiten Datenbus.
- Die Slow-Control dient nur dem langsamen Austausch von Kontrollsignalen, bei denen die Geschwindigkeit eine untergeordnete Rolle spielt. Um das Routing zu allen Slow-Control-Klienten möglichst einfach zu halten, sollte die Slow-Control mit weniger als 60 MHz betrieben werden.
- Um die physikalische Schnittstelle zum HAGEN Chip zuverlässig betreiben zu können, mussten Signallaufzeiten innerhalb des FPGA gezielt variiert und für die Synchronisation optimiert werden. Wegen der damit verbundenen Frequenzabhängigkeit wurde die physikalische Schnittstelle zwischen FPGA und HAGEN Chip nur bei der festen Frequenz von 78,125 MHz des Zustandsautomaten, 156,25 MHz des HAGEN-Versorgungstaktes und damit einer Transferleistung von 312,5 Mbit/s pro Leitung oder 0,626 GB/s für den 2 Byte breiten Datenbus verifiziert.
- Der evolutionäre Koprozessor wurde einschließlich der Ramklienten und des Befehlsspeicher bis zu einer Frequenz von 132 MHz synthetisiert.
- Der PowerPC ist vom FPGA-Hersteller bis zu einer Frequenz von 350 MHz spezifiziert.

Zur Taktung der Nathan Netzwerkmodule stehen zwei mögliche Taktgeber zur Verfügung. Zum einen kann der Takt verwendet werden, der das Token Ring Netzwerk vom Kontroll-PC aus taktet (vgl. Abschnitt 1.4.2)., wobei das Token Ring Netzwerk in einem Frequenzbereich von 60-80 MHz zuverlässig betrieben werden kann. Zum anderen befindet sich auf der Backplane ein Oszillator mit einer Ausgangsfrequenz von 156,25 MHz. Für die Taktung des FPGAs wird dieser Oszillator verwendetet.

Dadurch ist die Taktqualität besser als bei der Verwendung des Token Ring Taktes, zudem kann die Frequenz des Token Ring Netzwerkes dann unabhängig vom internen Takt des FPGAs gewählt werden.

Der FPGA stellt 4 DCMs zur Manipulation der Taktfrequenzen zur Verfügung. Ein DCM ist zum Betrieb des externen SRAM-Bausteins reserviert. Jeweils ein DCM wird zur Synchronisation der physikalischen Schnittstelle zum externen DDR-SDRAM-Speicherbaustein bzw. zum HAGEN Chip verwendet. Damit steht nur ein DCM zur Erzeugung der internen Taktsignale zur Verfügung, welcher mit der Frequenz von 156,25 MHz

gespeist wird. Daraus werden die Frequenzen 39,0625 MHz für die interne Slow-Control[2], 78,125 MHz für die HAGEN-Ablaufsteuerung und die Speicheransteuerung, 156,25 MHz für den HAGEN-Versorgungstakt und den RAM-Versorgungstakt und 312,5 MHz für den Koprozessor generiert.

Da nur ein DCM zur Generierung aller FPGA-internen Takte zur Verfügung steht, können die einzelnen Komponenten nicht mit ihrer jeweils maximalen Taktfrequenz versorgt werden. Durch diese Einschränkung werden die Speicheransteuerung und der evolutionäre Koprozessor nur mit 65% bzw. mit 59% ihrer maximalen Taktfrequenz betrieben. Auch der PowerPC wird nur mit 312,125 MHz statt mit der maximal spezifizierten Frequenz von 350 MHz betrieben. Da sowohl der Koprozessor als auch die Speicheransteuerung mit der gleichen Frequenz von 78,125 MHz getaktet werden, ist die Datenrate des Koprozessors in dieser Konfiguration durch die Speicherbandbreite begrenzt (vgl. Abschnitt 3.4.8). Die Speicheransteuerung kann zwar durch die doppelte Datenrate pro Takt 128 Bit an Daten verarbeiten, die Pipeline des Koprozessors kann allerdings pro Takt zwei Eltern und einen Nachkommen mit je 64 Bit Datenbreite verarbeiten.

In Kapitel 5.4 werden Möglichkeiten erörtert, wie eine größere Differenzierung in der Taktversorgung im Nathan-System erreicht werden kann.

4.4 Vergleichbare Implementationen

In der hier vorgestellten Experimentalplattform werden die Aufgaben des genetischen Algorithmus aufgeteilt: Der gemischt analog-digitale HAGEN Chip führt die *Netzwerkberechnung* für jedes Individuum durch, daraufhin vergleicht die Software die Ergebnisdaten des Netzwerkes mit den Solldaten und führt darauf die *Fitnessberechnung* aus. Die *Selektion* wird ebenfalls in Software ausgeführt. Die gesamte Kommunikation und vor allem die *Erzeugung der neuen Generation* wird in programmierbarer Logik durchgeführt.

In vergleichbaren Arbeiten zu genetischen Algorithmen und deren Realisierungen in Hardware werden meist keine neuronalen Netzwerke trainiert und es wird daher auch nicht zwischen *Netzwerkberechnung* und *Fitnessberechnung* unterschieden, sondern die gesamte Evaluation als Fitnessberechnung bezeichnet. Zudem werden meist Evaluation, Selektion und Erzeugung der neuen Generation spezifisch für eine oder für einige wenige Problemstellungen gemeinsam in einem System oder FPGA implementiert. Das hat zur Folge, dass einerseits die Variabilität sehr gering ist, denn für eine neue Problemstellung muss neue Hardware bzw. muss die programmierbare Logik neu entwickelt werden. Andererseits müssen die vorhandenen Logik-Ressourcen auf alle Teile des genetischen Algorithmus aufgeteilt werden. Da insbesondere die Netzwerk- bzw. Fitnessberechnung in digitaler Hardware sehr ressourcenintensiv ist, können nur Individuen mit wenigen Chromosomen bzw. wenigen Genen betrachtet werden, d.h. die Komplexität der bearbeiteten Probleme ist beschränkt.

Scott et al. [71] schlägt zwar eine externe Fitnessberechnung optional vor, die vorgestellten Problemstellungen implementieren allerdings die Fitnessberechnung innerhalb des FPGAs, es werden weder Ressourcenverbrauch noch Taktraten angegeben. Ein vorgestelltes Beispiel zur Funktionsoptimierung verwendet eine Individuengröße von 33 Bit.

[2]Nachdem die Daten über das Token Ring Netzwerk angekommen sind, werden sie intern mit diesem Takt weiterverarbeitet, ebenso wird der Zugriff des lokalen Slow-Control-Master mit diesem Takt betrieben.

Wakabayashi et al. [79] stellt eine VLSI-Implementation eines genetischen Algorithmus vor. Die Fitnessberechnung wird hierbei ausgelagert, der *Genetic algorithm accelerator GAA-II* kann Individuen mit einer maximalen Größe von 2048 Bits verarbeiten und arbeitet mit einem Takt von 50 MHz.

Shackleford et al. [73] implementiert Fitnessberechnung, Selektion und Erzeugung der neuen Individuen in programmierbarer Logik, es werden zwei Beispielprobleme für effiziente Fitnessberechnungen angegeben, die Größe der Individuen liegt dabei unter 100 Bit, es werden je 2000 LUTs für die Fitnessberechnung und den restlichen genetischen Algorithmus benötigt.

Areibi et al. [3] verwendet genetische Algorithmen zur Partitionierung von Schaltungen. Dabei werden die Aufgaben des genetischen Algorithmus (Selektion, Berechnung der Fitness und Erzeugung der neuen Generation) in einzelnen Modulen innerhalb eines FPGAs implementiert. Die Erzeugung der neuen Generation mittels Kreuzung und Mutation erfolgt in einer Datenpipeline, so dass prinzipiell Chromosomen beliebiger Größe bearbeitet werden können. Die vorgestellten Beispiele betrachten Individuengrößen von 10-300 Bit. Das Design wurde für einen großen VirtexE FPGA[3] mit 123 MHz synthetisiert.

Weitere Beispiele für Hardware-Realisierungen genetischer Algorithmen mit problemspezifischer Fitnessfunktion finden sich in [89]: *prisoners' dilemma*, [54]: *1-D Signal Reconstruction* und [2]: *Kompakter Genetischer Algorithmus*, dabei werden Individuengrößen von 32 Bit bis 32 Byte verwendet.

Im Gegensatz zu den meisten Beispielen aus der Literatur kann der in dieser Arbeit vorgestellte evolutionäre Koprozessor beliebige Chromosomen- und Individuengrößen verarbeiten. Zudem bietet er zwei Ebenen der Variabilität:

- Über den Koprozessoransatz und die Komponenten der Pipelinesteuerung lassen sich die genetischen Operatoren auch ohne eine Änderung der programmierbaren Logik auf der Basis von Chromosomen, Gensequenzen oder auch für einzelne Gene selektiv ändern.

- Durch den modularen Aufbau lassen sich sehr einfach neue Funktionen in den bestehenden Koprozessor einfügen, beispielsweise durch weitere Pipelinestufen oder durch neue Instruktionen. Dies erfordert aber eine Änderung der programmierbaren Logik.

Aufgrund der Variabilität des Koprozessors können beliebige Problemstellungen, die sich auf das neuronale Netzwerk innerhalb des HAGEN Chips abbilden lassen, variabel in Software beschrieben, aber mit der Geschwindigkeit von dedizierter Hardware ausgeführt werden.

[3] Die genaue Bezeichnung ist `Virtex XCV2000E`, der je 43200 FFs und LUTs bereitstellt.

Kapitel 5

Experimente

Diese Arbeit konzentriert sich auf die programmierbare Logik und die hardwarenahe Software des PCI-basierten Darkwing-Systems und des verteilten, parallelen Nathan-Systems. Die zugrunde liegende Hardware, d.h. die in Abschnitt 1.4 vorgestellte Darkwing-Karte, die Leiterplatten der Nathan Netzwerkmodule und die Backplane sind in [5] [25] und [69] dokumentiert. Die Trainingssoftware, die Entwicklung der Trainingsalgorithmen und die durchgeführten evolutionären Experimente werden in [29] beschrieben.

Dieses Kapitel erläutert kurz die Art dieser Experimente und vergleicht die technische Umsetzung auf dem Darkwing- und dem Nathan-System. Daraufhin werden die in beiden Systemen erreichten Trainingsgeschwindigkeiten gegenübergestellt und diskutiert. Schließlich werden die auf dem Darkwing-System erreichten Trainingsergebnisse zusammengefasst und mit den Ergebnissen aus dem Nathan-System verglichen.

5.1 Klassifikationsprobleme

Klassifikationsprobleme sind eines der Haupteinsatzgebiete von neuronalen Netzwerken [7]. In [29] werden insgesamt 9 standardisierte Benchmark Klassifikationsprobleme aus dem UCI KDD Online Archiv [27] untersucht. Dabei werden verschiedene Trainingsstrategien angewendet:

Der *Einfache Evolutionäre Ansatz* versucht, alle vorhandenen Muster in einem Evolutionsschritt zu lernen. Diese Trainingsmethode erreicht bei Problemen mit nur zwei Klassen annehmbare Ergebnisse, kann aber Probleme mit mehreren Klassen nicht annähernd zufriedenstellend lösen. Die *Schrittweise Trainingsmethode* dagegen trainiert in jedem Evolutionsschritt nur einen Teil, d.h. ein Subnetzwerk des gesamten neuronalen Netzwerkes und kann so auch bei Problemen mit mehr als zwei Klassen annehmbare Ergebnisse liefern. Die *Erweiterte Schrittweise Trainingsmethode* schließlich trainiert mehrere Subnetzwerke für eine Klasse unabhängig voneinander, und fügt erst nach dem Einzeltraining das vollständige Netzwerk zusammen.

Die *Erweiterte Schrittweise Trainingsmethode* erreicht die höchsten Klassifikationsgenauigkeiten, diese können sehr gut mit den Leistungen von softwarebasierten neuronalen Netzwerken konkurrieren und sind vergleichbar mit den besten Ergebnissen anderer Klassifikationsverfahren aus der Literatur.

5.2 Durchführung

Durch die Modularität und die Kapselung der Hardwareansteuerung können auf dem Darkwing- wie auf dem Nathan-System die gleichen Trainingsalgorithmen verwendet werden. Das Softwarepaket HANNEE wird dabei entweder für die x86-Architektur des PCs oder für den PowerPC kompiliert. Im Folgenden werden die Arbeitsschritte des evolutionären Algorithmus erläutert.

- Initialisierung
  Zur Initialisierung des Systems schreibt die Software die Parameter für die analoge Netzwerkberechnung in die Status Register der HAGEN-Ansteuerung, initialisiert die Zufallsgeneratoren des evolutionrren Koprozessors und schreibt die Eingabedaten der zu untersuchenden Problemstellung in den externen Speicher. Das erste Individuum wird ebenfalls durch die Software erzeugt, dabei werden die genabhängigen Bits für den Koprozessor nach den Anforderungen den evolutionären Algorithmus gesetzt. Dieses erste Individuum kann mit dem Koprozessor beliebig oft kopiert werden, um die erste Generation zu erhalten. Durch Setzen der gleichverteilten Mutationswahrscheinlichkeit auf 100% werden neue Individuen mit zufälligen Genwerten erzeugt.

- Netzwerkberechnung
  Um ein Individuum zur Netzwerkberechnung an den HAGEN Chip zu senden, muss die Software die Speicheradresse der Gewichtsdaten, der Eingabedaten und die Zieladresse für die Ergebnisdaten an die HAGEN-Ansteuerung übermitteln. Die HAGEN-Ansteuerung liest daraufhin die Gewichtsdaten aus dem Speicher, sortiert sie auf die Chipkoordinaten um und konfiguriert den HAGEN Chip. Daraufhin werden die Eingabedaten sukzessive angelegt, die Netzwerkoperation ausgeführt und die Ergebnisdaten zurück in den externen Speicher geschrieben.

- Einlesen der Ergebnisdaten
  Nach erfolgter Netzwerkberechnung stehen die Ergebnisdaten im externen Speicher. Für die Trainingsalgorithmen und die folgende Fitnessberechnung müssen die Ergebnisdaten aber in die `HNetData`-Klasse (vgl. Abschnitt 3.6.1) übertragen werden. Da jedes `HNetData`-Objekt einen Block des HAGEN Chips repräsentiert, müssen die zusammenhängend im Speicher stehenden Ergebnisdaten der vier Blöcke auf vier unterschiedliche `HNetData`-Objekte aufgeteilt werden.

- Fitnessberechnung
  Die Fitnessberechnung für ein Individuum läuft zweistufig ab: Zuerst werden die Ergebnisdaten eines Individuums einzeln betrachtet und mit den Solldaten verglichen. Jedes mit den Solldaten übereinstimmende Ergebnisdatum – dies entspricht einer richtigen Neuronenantwort des HAGEN Chips – wird mit einem Punkt belohnt. Falsche Antworten tragen nicht zur Erhöhung der Fitness bei. In einem zweiten Bearbeitungsschritt werden diese Punktsummen auf die Klassenstärken der Klassen des aktuellen Datensatzes normiert. Für weitere Details sowie die mathematische Beschreibung der Fitnessfunktion sei erneut auf [29] verweisen.

- Selektion und Erzeugung der neuen Generation
  Für die Selektion muss pro Individuum nur der Fitnesswert betrachtet werden. Die Erzeugung der neuen Generation dagegen ist aufwendig und läuft in zwei Stufen ab. Zuerst schreibt die Software die nötigen Befehle und die Daten für die indirekte Adressierung in den Befehlsspeicher des Koprozessors. Nachdem alle Daten übertragen worden sind, wird der Koprozessor über die Slow-Control gestartet und bearbeitet die Befehle eigenständig. Dabei wird die in Tab. 3.8 eingestellte Abhängigkeit zwischen Steuerbits und Multiplexer-Kontrollbits verwendet.

5.2.1 Darkwing-System

Im Darkwing-System muss die gesamte Kommunikation zwischen der Software und der programmierbaren Logik über den PCI-Bus, den PLX-Chip und die Komponente `PLX-Slave` (vgl. Abschnitt 3.5) abgewickelt werden. Da die in der Praxis erreichbaren Transferraten von 80-90 MByte/s des PCI-Bus nur bei DMA-Blockzugriffen realisiert werden können, werden die zu übertragenden Daten vorher im PC-Speicher akkumuliert. Ein Beispiel ist die Übertragung der Befehle und Daten für den Koprozessor: Zuerst schreibt die Software die Befehle in ein Datenfeld, das im PC-Speicher vorgehalten wird. Erst wenn alle Befehle für diesen Koprozessoraufruf geschrieben wurden, wird das gesamte Datenfeld mittels DMA auf die Darkwing-Karte übertragen. Ebenso werden die Ergebnisdaten erst als kompletter Block aus dem Darkwing-Speicher in den PC-Speicher übertragen und dann auf die vier `HNetData`-Objekte, die die HAGEN-Blöcke repräsentieren, umverteilt.

Durch die Verwendung des Koprozessors muss das genetische Material der Individuen nur während der Initialisierung über den PCI-Bus transportiert werden. In der laufenden Evolution muss der PC-Prozessor die genetischen Daten nicht betrachten, sie werden nur zwischen Koprozessor und HAGEN-Ansteuerung ausgetauscht.

5.2.2 Nathan-System

Vor der Benutzung des Nathan-Systems muss das eingebettete Linux gebootet werden. Nachdem Ramdisk und Kernel-Code über die Slow-Control in das DDR-SDRAM des Nathan Netzwerkmodules transferiert wurden, wird der Bootloader gestartet. Der Bootloader beschreibt die internen Register im PowerPC und bereitet den Bootvorgang vor. Direkt nach dem Start des Kernels wird der Netzwerktreiber `netPC` (vgl. 3.6.2) geladen. Nun steht die Netzwerkschnittstelle zur Verfügung und das Anmelden auf dem Linux des Nathan Netzwerkmodules vom Kontroll-PC aus ist möglich.

Auf dem Nathan-System kann der PowerPC den externen Speicher über den PLB, den Befehlsspeicher des Koprozessors über den OCM und die Slow-Control als lokaler Master über das DCR ansprechen (vgl. Abschnitt 3.5). Die Zugriffe auf den externen Speicher und den Befehlsspeicher können nach Laden der entsprechenden Treiber (vgl. Abschnitt 3.6.2) mit einem Zeigerzugriff erfolgen. Dadurch können die Befehle und Daten direkt in den Befehlsspeicher des Koprozessors geschrieben werden, ebenso können über den PLB die Ergebnisdaten direkt bzw. über den Datencache des PowerPCs, aus dem externen DDR-SDRAM gelesen werden.

Da im Nathan-System wie im Darkwing-System das Softwarepaket HANNEE verwendet wird, werden auch auf dem eingebetteten PowerPC die Ergebnisdaten, die zu einem

Eingabedatum gehören und konsekutiv im externen Speicher stehen, auf vier `HNetData`-Objekte, welche die vier HAGEN-Blöcken repräsentieren, umverteilt, bevor die Fitnessberechnung gestartet werden kann. Die dadurch entstehenden Nachteil werden im folgenden Abschnitt diskutiert.

5.3 Zeitmessungen

Um die Geschwindigkeit des evolutionären Algorithmus zu bestimmen und zwischen beiden Plattformen zu vergleichen, wurden die Laufzeiten der einzelnen Schritte am Beispiel des *Liver Disorder*-Datensatzes aus [27] gemessen. Der *Liver Disorder*-Datensatz besteht aus 1550 Eingabedaten, aufgrund der für den Test angewendeten *Schrittweisen Trainingsmethode* wird in jedem Evolutionsschritt nur ein Teil, d.h. 10 der insgesamt 256 Chromosomen des HAGEN Chips, trainiert. Eine Population besteht aus 20 Individuen. Es wird die für die einzelnen Arbeitsschritte benötigte Zeit einerseits des Nathan-Systems und des Darkwing-System verglichen. Der Kontroll-PC des Darkwing-System ist dabei ein Pentium IV, der mit 2,4 GHz getaktet ist. Die Zeitmessungen wurden aus der Trainingssoftware mit der Funktionalität der `ctime`-Bibliothek durchgeführt [29]. Die Messzeiten sind in Tab. 5.1 zusammengefasst.

	Darkwing-System	Nathan-System
Netzwerkberechnung [ms]	1.26 ± 0.01	0.904 ± 0.005
Zurücklesen der Ergebnisdaten [ms]	0.78 ± 0.02	1.593 ± 0.014
Fitnessberechnung [ms]	0.28 ± 0.01	6.354 ± 0.198
Selektion und Erzeugung eines neuen Individuums [ms]	0.17 ± 0.01	0.109 ± 0.003

Tabelle 5.1: Laufzeiten der Arbeitsschritte des evolutionären Algorithmus am Beispiel des *Liver Disorder*-Datensatzes für die beiden Plattformen

Die Laufzeiten der einzelnen Arbeitsschritte werden im Folgenden diskutiert.

- Die Netzwerkberechnung ist auf dem Nathan-System aufgrund der höheren Frequenz der Schnittstelle zum HAGEN Chip schneller als auf dem Darkwing-System. Gemäß Abschnitt 3.3.5 benötigt die Konfiguration des HAGEN Chips 163,84 μs, die Verarbeitung eines Eingabedatums einschließlich des Datentransfers zum und vom HAGEN Chip benötigt 435,2 ns, in der Summe beträgt die theoretische Laufzeit der Ablaufsteuerung für 1550 Eingabedaten folglich 838,4 μs. Die von der Trainingssoftware gemessene Zeit ist aufgrund der Hardware-Software Kommunikation und der Tatsache, dass das simultan laufende Betriebssystem auf dem PowerPC auch Speicherbandbreite beansprucht, geringfügig größer.

- Auf dem Darkwing-System müssen die Ergebnisdaten aus dem Speicher der Darkwing-Karte über die Komponente `PLX-Slave` in den PC-Speicher gelesen werden, im PC dann auf die `HNetData`-Objekte umkopiert werden. Auf dem Nathan-System können die Ergebnisdaten nach dem Aufruf des Treibers `memIo` direkt aus dem externen Speicher gelesen werden. Bis auf diese Änderung wurde die auf dem PC entwickelte Software ohne weitere Optimierungen direkt auf den PowerPC übertragen. Das Umkopieren der Ergebnisdaten auf die 4 unterschiedlichen `HNetData`-Objekte ist aber auf dem eingebetteten Nathan-System aufgrund des relativ kleinen, 2fach assoziativen Caches sehr ineffizient und benötigt etwa doppelt so lange wie das Transferieren der Ergebnisdaten im Darkwing-System über die PCI-Bus. Durch eine optimal Ausnutzung des PowerPC-Caches könnte das Auslesen der Ergebnisdaten deutlich beschleunigt werden.

- In der Fitnessberechnung zeigt sich deutlich die Überlegenheit des mit 2,4 GHz getakteten PCs gegenüber dem eingebetteten PowerPC, der mit 312 MHz getaktet wird. Der PowerPC im Nathan-System braucht mit 6,354 ms deutlich länger als der PC im Darkwing-System (0,28 ms). Auch hier muss darauf hingewiesen werden, dass die Software ursprünglich für den PC entwickelt und ohne Optimierungen auf den PowerPC portiert wurde. Durch eine Anpassung der relevanten Softwareteile vor allem auf die Cache-Struktur des PowerPC ist eine deutliche Steigerung der Geschwindigkeit zu erwarten.

  In Kapitel 5.4 werden verschiedene Möglichkeiten diskutiert, wie sich das Zurücklesen der Ergebnisdaten und die Fitnessberechnung im Nathan-System beschleunigen ließe, einerseits durch Optimierungen der Software, andererseits durch eine Vorverarbeitung der Ergebnisdaten in der programmierbaren Logik.

- Die Zeiten für Netzwerkberechnung, Rücklesen der Ergebnisdaten und Fitnessberechnung bezogen sich jeweils auf ein Individuum. Für die Selektion und die Erzeugung der neuen Generation wird die gesamte Population betrachtet. Der *Liver Disorder*-Datensatzes wird mit einer Populationsgröße von 20 Individuen trainiert. Wie aus Tab. 5.1 abzulesen ist, ist sowohl auf dem Darkwing- als auf dem Nathan-System der Zeitbedarf für die Selektion und die Erzeugung der neuen Generation gering, da aufgrund der *Schrittweisen Trainingsmethode* pro Evolutionsschritt nur 10 der maximal 256 Chromosomen des HAGEN Chips trainiert werden. Um ein neues Individuum zu erzeugen, müssen daher auch nur diese 10 Chromosomen neu berechnet werden.

  Um abzuschätzen, wie der Zeitbedarf der Erzeugung der neuen Generation mit der Anzahl der bearbeiteten Chromosomen skaliert, wurden die Anzahl der Chromosomen variiert und jeweils die Laufzeit des Koprozessors und die Softwarezeit zum Schreiben der Befehle gemessen. Die Ergebnisse stellt Abb. 5.1 dar[1].

  Die nötige Zeit zur Selektion und zur Erzeugung der neuen Generation von 20 Individuen setzt sich folglich aus einem großen Offset von 1.9 ms und einem kleinen von der Chromosomenanzahl abhängigen Anteil zusammen. Der Koprozessor benötigt

[1] Die Ausgleichsgraden wurden mit matlab [42] berechnet, indem die quadratischen Fehler eines Polynoms ersten Grades zu den Messdaten minimiert wurden.

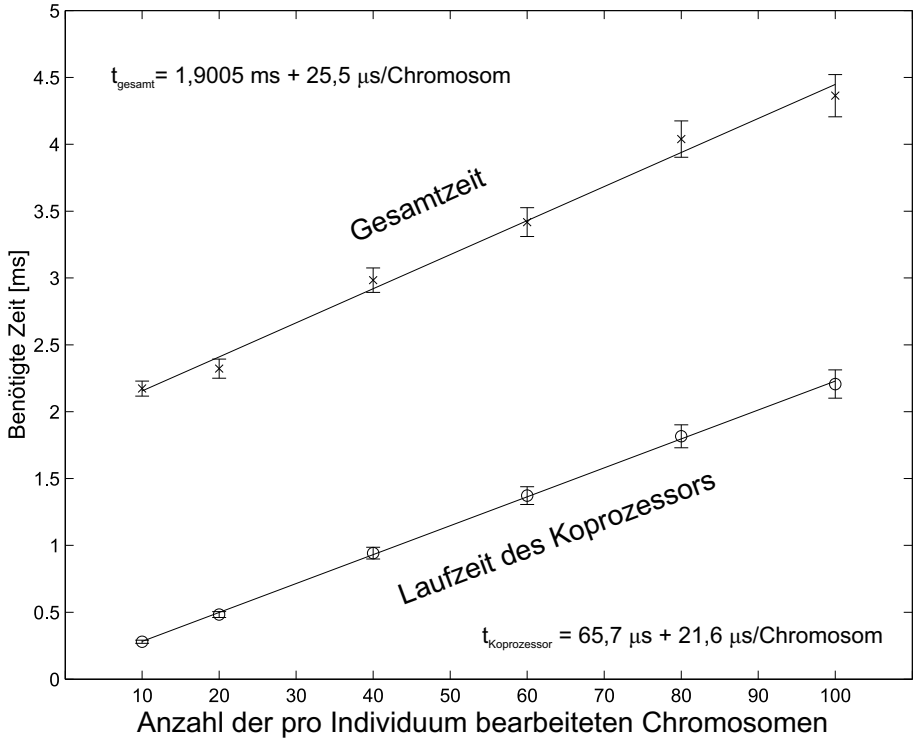

Abbildung 5.1: Zeitbedarf für Selektion und Erzeugung einer neuen Generation mit 20 Individuen am Beispiel des *Liver Disorder*-Datensatzes. Es wurde einerseits die reine Laufzeit des evolutionären Koprozessors, andererseits die Gesamtzeit einschließlich der Selektion und der Befehlserzeugung in Software gemessen. Die Ausgleichsgraden zeigen einen hohen Softwareoffset von knapp 1,9 ms an. Pro Individuum wird entsprechend $\frac{1}{20}$ der angegebenen Zeit benötigt.

einen geringen Offset und pro Chromosom 21,6 μs. Dies entspricht einer Rate von 118,5 $\cdot 10^6$ Genen/s. Da der Koprozessor in der gegenwärtigen Konfiguration der Experimentalplattform nicht mit maximaler Frequenz betrieben wird und zudem durch die Speicherbandbreite begrenzt ist (vgl. Abschnitt 4.3), wird nur knapp ein Drittel der maximalen Rate (vgl. Abschnitt 3.4.7) erreicht. Da in der *Schrittweisen Trainingsmethode* für den *Liver Disorder*-Datensatz nur 10 Chromosomen pro Evolutionsschritt optimiert werden, ist die Selektion und die Erzeugung der neuen Generation deutlich schneller als beispielsweise die Netzwerkberechnung. Die Daten aus Abb. 5.1 zeigen aber, dass auch die Erzeugung aller 256 Chromosomen des HAGEN Chips in kürzerer Zeit, als die Netzwerkberechnung benötigt, möglich ist: Mittels der Ausgleichsgraden kann auf die die Erzeugung von 256 Chromosomen pro Individuum hochgerechnet werden. Eine gesamte Population benötigt 8,43 ms, umgerechnet auf ein Individuum ergibt sich eine Rechenzeit von 0,421 ms und damit weniger als 50 % der Zeit der Netzwerkberechnung.

Fasst man die am Beispiel des *Liver Disorder*-Datensatzes gemessenen Laufzeiten zusammen, so entspricht die Zeit für die Netzwerkberechnung den Voraussagen aus Kapitel 3. Die

Erzeugung der neuen Generation erreicht aufgrund der langsameren Taktung der Speicheransteuerung und des Koprozessors nur etwa ein Drittel der maximalen Rate, ist damit aber trotzdem schnell genug, um ein vollständiges Individuum für den HAGEN Chip in der Zeit, die die Netzwerkberechnung eines Individuums benötigt, zu erzeugen. Das Zurücklesen der Ergebnisdaten und die Fitnessberechnung ist im Nathan-System deutlich langsamer, hier macht sich die niedrigere Rechenleistung und die ungünstige Nutzung des kleinen, 2fach assoziativen Caches des eingebetteten PowerPCs deutlich bemerkbar.

5.4 Trainingsergebnisse

Der Großteil der evolutionären Experimente wurde auf dem Darkwing-System durchgeführt, da das Darkwing-System mitsamt dem evolutionären Koprozessor seit 2004 zuverlässig arbeitet, das Nathan-System aber erst im Herbst 2005 fertig gestellt wurde. Die Entwicklung des *Einfachen Evolutionären Ansatzes*, der *Schrittweisen Trainingsmethode* und der *Erweiterten Schrittweisen Trainingsmethode* erfolgte deshalb auf dem Darkwing-System.

In [29] wird gezeigt, dass die auf dem Darkwing-System mit der *Erweiterten Schrittweisen Trainingsmethode* erreichten Klassifikationsgenauigkeiten sehr gut mit den Leistungen von software-basierten neuronalen Netzwerken konkurrieren können und auch vergleichbar mit den besten Ergebnissen anderer Klassifikationsverfahren aus der Literatur sind.

Um die Funktionsfähigkeit des Nathan-Systems zu demonstrieren, wurden 4 der insgesamt 9 Probleme mit der *Schrittweisen Trainingsmethode* erneut parallel auf 4 Nathan Netzwerkmodulen des Nathan-Systems durchgeführt und die erreichten Klassifikationsgenauigkeiten mit denen des Darkwing-Systems verglichen[2]. Die Ergebnisse gibt Tab. 5.2 wieder.

	Klassifikationsgenauigkeiten [%]			
	Training		Generalisierung	
Benchmark	Darkwing System	Nathan System	Darkwing System	Nathan System
heart disease	$90,96 \pm 0,01$	$91,17 \pm 0,12$	$80,03 \pm 0,54$	$81,86 \pm 0,75$
liver disorder	$77,24 \pm 0,11$	$77,32 \pm 0,22$	$66,51 \pm 0,72$	$67,41 \pm 0,95$
iris plant	$99,50 \pm 0,03$	$99,54 \pm 0,03$	$95,10 \pm 0,54$	$94,21 \pm 0,27$
wine	$100,0 \pm 0,0$	$99,98 \pm 0,01$	$95,31 \pm 0,28$	$93,63 \pm 0,65$

Tabelle 5.2: Vergleich der Klassifikationsgenauigkeiten, die auf dem Nathan-System und auf dem Darkwing-System erzielt worden sind [29].

Wenn man die Generalisierung betrachtet, so klassifiziert das Darkwing-System die Probleme *iris plant* und *wine* besser, das Nathan-System erzeugt aber bei den Problemen *heart disease* und *liver disorder* die besseren Ergebnisse. Dieses Verhalten ist möglicherweise durch den Versuchsaufbau bedingt: Sowohl auf dem Darkwing- als auch im

[2]Da die Experimente mit der *Erweiterten Schrittweisen Trainingsmethode* sehr zeitintensiv sind, wurde für die Demonstration der Funktionsfähigkeit die *Schrittweisen Trainingsmethode* verwendet.

Nathan-System wurde für eine Problemstellung ein festes PC-System, bzw. ein festes Nathan Netzwerkmodul verwendet, damit also immer nur jeweils ein bestimmter HAGEN Chip verwendet. Damit sind die erreichten Klassifikationsgenauigkeiten abhängig von der analogen Genauigkeit des für diese Problemstellung verwendeten HAGEN Chips.

Die mit der *Schrittweisen Trainingsmethode* erzielten Klassifikationsgenauigkeiten des Nathan-Systems sind mit den Ergebnissen, die auf dem Darkwing-System gewonnen wurden, vergleichbar. Es konnte folglich die Funktionsfähigkeit und Zuverlässigkeit des vorgestellten verteilten, parallelen Nathan-Systems demonstriert werden. Damit bietet das parallele, verteilte Nathan-System ideale Voraussetzungen, um einerseits umfangreiche Messungen mit großer Statistik zum Vergleich der analogen Genauigkeit verschiedener HAGEN Chips durchzuführen, oder um andererseits weitere Problemstellungen oder Trainingsalgorithmen zu untersuchen.

Zusammenfassung

Künstliche neuronale Netzwerke stellen ein alternatives Berechenbarkeitsmodell zum klassischen Computer dar. Da die Informationsverarbeitung in neuronalen Netzwerken verteilt in den einzelnen Synapsen und Neuronen abläuft, eigenen sie sich sehr gut für eine parallele Implementation in Hardware. Der gemischt analog-digitale HAGEN Chip ermöglicht die flexible Ausführung mehrschichtiger sowie rückgekoppelter neuronaler Netzwerke. Durch die analoge Implementation kann die Netzwerkberechnung mit hoher Geschwindigkeit und niedrigem Stromverbrauch durchgeführt werden, die rein digitale Schnittstelle ermöglicht den direkten Anschluss an digitale Kontrolllogik und damit eine einfache Skalierbarkeit.

In der Electronic Vision(s) Gruppe am Kirchhoff-Institut für Physik wurden zwei Experimentalplattformen entwickelt, um hardwarebasierte neuronale Netzwerke in Betrieb zu nehmen und zu trainieren. Diese Arbeit konzentriert sich einerseits auf die Aspekte der Systemintegration wie die programmierbare Logik und die hardwarenahe Software. Andererseits wurde ein Koprozessor entwickelt, der die Parallelität der programmierbaren Logik ausnutzt, um die daten- und rechenintensiven Teile der zum Training verwendeten evolutionären Algorithmen auszuführen. Durch die Implementation einer dedizierten Datenpipeline kann dieser Koprozessor abhängig von der Struktur des genetischen Materials zwischen 290 und 500 Millionen Gene pro Sekunde berechnen. Durch den Koprozessoransatz bleibt die Kontrolle der Trainingsalgorithmen in Software, die über einen speziellen Befehlssatz bestimmt, wie Individuen, Chromosomen oder auch einzelne Gene bearbeitet werden.

Der Aufbau beider Plattformen zeichnet sich durch eine hohe Modularität auf allen Ebenen der Hardware, der programmierbaren Logik und der Software aus: Der Zugriff der Software auf die programmierbare Logik wird durch einheitliche Schnittstellen gekapselt. Dadurch können die Anwendungen und Trainingsalgorithmen unabhängig von der Hardwareansteuerung formuliert werden und sind zwischen beiden Plattformen portierbar. Innerhalb der programmierbaren Logik wurde eine geeignete Infrastruktur aufgebaut: Über die *Slow-Control* können alle Komponenten mit niedriger Rate angesprochen und gesteuert werden. Die *Speicheransteuerung* ermöglicht es einer beliebigen Anzahl unabhängiger logischer Speicherschnittstellen mit einer Netto-Rate von insgesamt bis zu 1,32 GByte/s mit dem externen Speicherbaustein zu kommunizieren.

Die Logik zur Ansteuerung des Netzwerkchips HAGEN ist durch diese Infrastruktur weitgehend unabhängig von der verwendetet Hardwareplattform. Die physikalische Schnittstelle zwischen FPGA und HAGEN Chip wurde hingegen für das Nathan-System optimiert, so dass eine Transferrate von 0,63 GByte/s übertragen werden kann.

Aufgrund der Variabilität der Software können auf den vorgestellten Plattformen verschiedene Trainingsstrategien verfolgt werden: Beispiele sind Liquid Computing [41] [69] und die Objekterkennung durch mehrschichtige, hierarchische Netzwerke [19] [18]. Die Kombination aus Mikroprozessor, Koprozessor und dem Netzwerkchip HAGEN ermöglicht es, evolutionäre Algorithmen in Software zu beschrieben, aber mit der Geschwindigkeit von dedizierter Hardware auszuführen. So konnte in [29] mit der Untersuchung von Benchmark-Klassifikationproblemen auf dem Darkwing-System gezeigt werden, dass die Ergebnisse dieses evolutionären Trainings konkurrenzfähig zu anderen Lösungen auf der Basis neuronaler Netzwerke sind.

Anhand von evolutionären Experimenten konnte die Funktionsfähigkeit des Nathan-Systems demonstriert werden. Die in Software formulierten Trainingsalgorithmen können den Netzwerkchip und den evolutionären Koprozessor direkt ansprechen und effektiv nutzen. Durch zukünftige Verbesserungen der Fitnessberechnung und der Kommunikation zwischen den Nathan Netzwerkmodulen ließe sich die Leistung des Nathan-System noch steigern und die möglichen Einsatzgebiete erweitern:

Fitnessberechnung Die auf dem Nathan-System erreichten Geschwindigkeiten des evolutionären Trainings sind durch die in Software ausgeführte Fitnessberechnung begrenzt. Im Gegensatz zu der hardwarenahen Software und den Linux-Treibern wurden die Trainingsalgorithmen bisher nicht an das Nathan-System angepasst. Daher ist zu erwarten, dass sich die Berechnung der Fitness und damit das gesamte evolutionäre Training in drei Stufen deutlich beschleunigen ließe:

- Softwareoptimierung:
  Da der Mikroprozessor im Nathan-System direkt auf die Ergebnisdaten zugreifen kann, sollte ein zusätzliches Umkopieren vermieden werden. Ebenso könnte durch gezielte Wahl der physikalischen Adressen beim Zugriff auf Ergebnis- und Solldaten der Cache deutlich effektiver genutzt werden.

- Reduktion der Daten in der programmierbaren Logik:
  Ein Ergebnisdatum des HAGEN Chips besteht aus den Binärwerten der 256 Ausgabeneuronen. Im Allgemeinen benötigt der Trainingsalgorithmus aber nur einen Bruchteil dieser Information. Eine Vorsortierung innerhalb der programmierbaren Logik, etwa als zusätzliche Komponente im Datenpfad der HAGEN-Ansteuerung, könnte die Datenmenge, die die Software lesen und verarbeiten muss, deutlich reduzieren. Über eine Bitmaske könnte eine derartige Sortierung mit geringem Aufwand auch programmierbar implementiert werden.

- Verlagerung von Teilen der Fitnessberechnung in die programmierbare Logik:
  In der ersten Stufe der verwendeten Fitnessberechnung werden nur die Ergebnisdaten jedes Individuums mit den Solldaten verglichen. Jede übereinstimmende Neuronenantwort erhöht die Fitness um einen Punkt. Diese Vergleichsoperation mit Aufsummierung der Fitness eignet sich sehr gut für eine Implementation innerhalb der programmierbaren Logik, allerdings würde dadurch die Wahl der Fitnessfunktion eingeschränkt.

Taktversorgung Die Speicheransteuerung, der Koprozessor und die HAGEN-Ansteuerung werden im vollständigen Trainingssystem gegenwärtig mit der gleichen Frequenz von 78,125 MHz betrieben. Dadurch können die möglichen höheren Geschwindigkeiten der Speicheransteuerung und des Koprozessor nicht genutzt werden. Da sowohl die Speicheransteuerung als auch die Slow-Control als Taktgrenze zwischen asynchronen Taktdomänen verwendet werden können, fehlen zum Betrieb aller Komponenten mit der maximalen Frequenz nur eine hinreichende Anzahl Taktgeber der jeweils optimalen Frequenz.

Aufbau größerer Netzwerke Ein wesentlicher Vorteil des Nathan-Systems gegenüber dem Darkwing-System ist die Möglichkeit, Daten mit hoher Geschwindigkeit und kleiner, fester Latenz zwischen verschiedenen FPGAs zu transportieren. Jeder FPGA des Nathan-Systems stellt 8 Hochgeschwindigkeitsverbindungen (MGTs) zur Verfügung. Jeweils 4 dieser MGTs sind über die Backplane zu einem zweidimensionalen Torus verbunden. Über Kabel können die restlichen Leitungen auch mit Nathan Netzwerkmodulen anderer Backplanes verbunden werden.

Diese Struktur wird in [55] genutzt, um einerseits die Speicheransteuerungen der angeschlossenen Nathan Netzwerkmodule über ein *Shared-Memory* zu verbinden, andererseits werden die Daten der HAGEN-Ansteuerung mit hoher Priorität und garantierter Latenz übermittelt. Die schnelle Kommunikation zwischen den einzelnen Nathan Netzwerkmodulen ermöglicht somit es, die Parallelität des Nathan-Systems etwa zur Untersuchung von Insel-Modellen zu nutzen, ebenso können bis zu 16 Netzwerkchips pro Backplane miteinander zu einem großen Netzwerk verbunden werden.

Anhang A

Schnittstelle Ramnutzer - Ramklient

Schreibzugriff Wenn das `inhibit`-Signal durch den Wert '0' die Bereitschaft des Ramklienten anzeigt, kann der Ramnutzer synchron Schreibzugriffe in Auftrag geben. Da ein Zugriff auf den externen DDR-SDRAM-Baustein 128 Bit breit ist, müssen die 64 Bit breiten Schreibzugriffe immer paarweise auftreten, begonnen mit einer geraden Adresse. Nach Empfang des jeweils zweiten 64 Bit breiten Datenwortes ist ein Zugriff abgeschlossen und wird eine unbestimmte Zeit später über Manager und Ramkontrolleinheit durchgeführt.

```
if(rising_edge(takt_des_ramnutzers)) then
  if(inhibit='0') then
      req <= '1';
      rwb <= Schreibzugriff;
      adr <= Zieladresse;
      din <= zu_schreibende_daten;
  else req <= '0';
  end if;
end if;
```

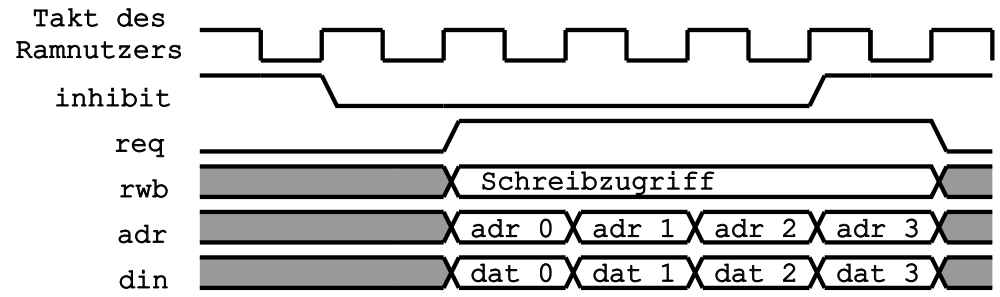

Abbildung A.1: Zur Definition der Schnittstelle zwischen Ramnutzer und Ramklienten. Durchführung von 2 Schreibzugriffen der Breite 128 Bit

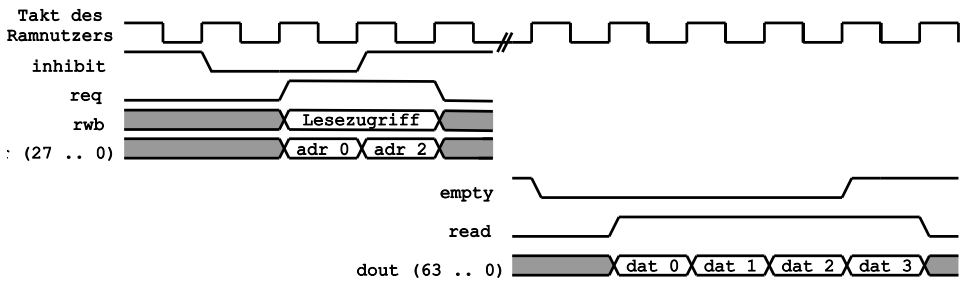

Abbildung A.2: Zur Definition der Schnittstelle zwischen Ramnutzer und Ramklienten. Durchführung von 2 Lesezugriffen der Breite 128 Bit. Eine Leseanforderung wird mit zwei Datenwörtern der Breite 64 Bit beantwortet.

Lesezugriff Der Lesezugriff teilt sich in den Anforderungspfad und den Datenlesepfad. Wenn das `inhibit` durch den Wert '0' die Bereitschaft des Ramklienten anzeigt, können Leseanfragen gestellt werden. Durch die Zugriffsbreite von 128 Bit des externen Speicherbausteins sind nur Lesezugriffe auf gerade Adressen erlaubt. Ein Leseanforderung wird mit 128 Bit Daten, aufgeteilt auf zwei Zyklen von jeweils 64 Bit, beantwortet (vgl. Abbildung A.2).

Lesezugriff - Anforderungspfad

```
if(rising_edge(takt_des_ramnutzers)) then
     if(inhibit='0') then
         req <= '1';
         rwb <= Lesezugriff;
         adr <= Leseadresse (gerade);
     else;
         req <= '0';
     end if;
end if;
```

Lesezugriff - Datenlesepfad

```
if(rising_edge(sys_clk)) then
     if(empty='0') then
         read <= '1';
     else;
         read <= '0';
     end if;
     if(read='1') then
         gelesene_daten <= dout;
     end if;
end if;
```

Anhang B

HAGEN Chip: Chipkoordinaten - Benutzerkoordinaten

Durch seine interne Struktur verwendet der HAGEN Chip für die Gewichts-, Eingabe- und Ergebnisdaten eine andere Sortierung als es für die Implementation der Trainingsalgorithmen zweckmäßig ist. Im Folgenden wird die Abbildung der Benutzerkoordinaten auf die Chipkoordinaten dargestellt.

Gewichtsdaten

Benutzerkoordinaten Die Benutzerkoordinaten indizieren die Gewichtsdaten erst block-, dann zeilen- dann spaltenweise. Im Folgenden wird die Notation (Block/Zeile/Spalte) verwendet. Die Blöcke sind in der Reihenfolge

```
unterer rechter Block (0),
oberer  rechter Block (1),
unterer linker  Block (2),
oberer  linker  Block (3)
```

durchnummeriert. Die Reihenfolge wird durch folgenden Pseudo-Code beschrieben:

```
for (block=0; block<4; block++){
  for (zeile=0; zeile<64; zeile++){
    for (spalte=0; spalte<128, spalte++){
      Gewicht(block, zeile, spalte);
    }
  }
}
```

Chipkoordinaten Die Chipkoordinaten indizieren die Gewichtsdaten spaltenweise. Pro Takt werden 4 Gewichtsdaten geschrieben, die Reihenfolge wird durch folgenden Pseudo-Code beschrieben:

```
for (spalte=0; spalte<128, spalte++){
  for (block=0; block<4; block+=2){
    for (startzeile=0; startzeile<8; startzeile++){
      for (zeile=startzeile; zeile<64; zeile+=16){
        Gewicht(block,   zeile,   spalte); // untere Chiphälfte
        Gewicht(block,   zeile+8, spalte);
        Gewicht(block+1, zeile,   spalte); // obere Chiphälfte
        Gewicht(block+1, zeile+8, spalte);
      }
    }
  }
}
```

Eingabedaten

Die Eingabedaten für einen Block werden im Folgenden als Zeilenvektor betrachtet.
Die Benutzerkoordinaten indizieren die Eingabedaten byteweise gemäss folgendem Pseudo-Code:

```
for (block=0; block<4; block++){
  for (zeile=0; zeile<16; zeile++){
    Eingabebyte(block, zeile);
  }
}
```

Die Chipkoordinaten werden durch folgenden Pseudo-Code beschrieben:

```
for (block=0; block<4; block+=2){
  for (zeile=0; zeile<8; zeile++){
    Eingabebyte(block,   zeile+8); // untere Chiphälfte
    Eingabebyte(block,   zeile);
    Eingabebyte(block+1, zeile+8); // obere Chiphälfte
    Eingabebyte(block+1, zeile);
  }
}
```

Ergebnisdaten

Die Ergebnisdaten für einen Block werden im Folgenden als Spaltenvektor betrachtet. Die Benutzerkoordinaten indizieren die Ergebnisdaten 16 Bit-wortweise gemäss folgendem Pseudo-Code:

```
for (block=0; block<4; block++){
  for (spalte=0; spalte<4; spalte++){
    16-Bit-Eingabewort(block, spalte);
  }
}
```

Die Chipkoordinaten werden durch folgenden Pseudo-Code beschrieben:

```
for (block=0; block<4; block+=2){
  for (spalte=0; spalte<4; spalte++){
    16-Bit-Eingabewort(block,   spalte); // untere Chiphälfte
    16-Bit-Eingabewort(block+1, spalte); // obere Chiphälfte
  }
}
```

Anhang C

Evolutionärer Koprozessor

C.1 Evolutionärer Koprozessor - Programmiermodell

Die gegenwärtig implementierten Befehle des evolutionären Koprozessors werden im Folgenden aufgelistet.

Selektionsbefehle Aus jedem Selektionsbefehl generiert der Dekodierer ein Selektionskommando, das an die Pipeline übermittelt wird. Dabei wird unterschieden zwischen eigenständigen Selektionsbefehlen oder solchen, die innerhalb einer Schleife gegeben werden. Die eigenständigen Selektionsbefehle enthalten als Parameter eine Befehlsspeicher-Adresse für die indirekte Adressierung. An dieser Befehlsspeicher-Adresse ist die SDRAM-Adresse des Chromosomes abgelegt. Die Selektionsbefehle zur Anwendung in Schleifen enthalten keine Parameter. Jeder Selektionsbefehl benötigt ein 32 Bit Befehlswort und 4 Takte zur Ausführung.

Eigenständige Selektionsbefehle	Selektionsbefehle in Schleifen
`parent 1`	`loopParent 1`
`parent 2`	`loopParent 2`
`parentOff 2`	`loopParentOff 2`
`offspring`	`loopOffspring`

Befehle zur Erzeugung der Steuerbits Die folgenden Befehle, die jeweils in einem Befehlswort kodiert sind, erzeugen die Steuerbits. Innerhalb von Schleifen werden mit dem Befehl `crossover` die Maskenlängen der 1-Punkt-Kreuzung für das aktuelle Chromosom erzeugt. Da hierfür zwei Maskenlängen geschrieben werden, benötigt dieser Befehl 2 Takte zur Ausführung.

Befehl	Parameter
`mask 0`	'0'-Maskenlänge
`mask 1`	'1'-maskenlänge
`probab-uni`	Wahrscheinlichkeit der gleichverteilten Mutation
`probab-gauss`	Wahrscheinlichkeit der gaußschen Mutation
`crossover`	

Befehle zur Programmierung von internen Registern und Schleifen Für den Schleifenbetrieb wird die SDRAM-Adresse des aktuellen Individuums und die Befehlsspeicher-Adresse der Chromosomenstruktur im Dekodierer gespeichert. Die folgenden Befehle schreiben bzw. aktualisieren diese Werte und implementieren Schleifenfunktionalität, bzw. initialisieren die Zufallsgeneratoren. Der Befehl `writeRegister` überträgt den Wert des Registers im Folgenden Befehlswort und benötigt daher zwei Takte zur Ausführung und auch zwei Befehlsworte zur Dekodierung.

Befehl	Parameter
`writeRegister`	Nummer des zu schreibenden Registers, das folgende Befehlswort enthält den neuen Wert des Registers
`nextChromo`	
`jumpNotZero`	Sprungpunkt
`randomInit`	Nummer des Zufallsgenerators, Initial-Wert
`allRandomOn`	

Halt und Debug Der `halt` Befehl versetzt den Kodierer zurück in den Ausgangszustand `IDLE`. Der `debug` Befehl wurde zur Fehlersuche eingesetzt.

Befehl	Parameter
`halt`	
`debugCmd`	Debug Wert

C.2 Evolutionärer Koprozessor - Softwareschnittstelle

Die Softwareklasse `evoCop` kapselt die Schnittstelle zum evolutionären Koprozessor und stellt die folgenden Methoden zur Verfügung:

Organisation des Befehlsspeichers Die folgenden Befehle reservieren und beschreiben Speicherbereiche im Befehlsspeicher, einerseits für die indirekt Daten wie beispielsweise die Chromosomenstruktur der Individuen, andererseits für die bereits vorgestellten Befehle.

Methode	Parameter
`allocIbufCmd`	`size`
`freeIbufCmd`	
`resetIbufCmd`	
`allocIbufData`	`size`
`freeIbufData`	`addr`
`writeIbufData`	`addr, data`

Organisation des externen SDRAM-Speichers Die folgenden Befehle reservieren Speicherbereiche im externen SDRAM-Speicher. Die Software greift in beiden Systemen über einen Zeigerzugriff auf den Speicher zu. Im Nathan-System kann dabei der externe SDRAM-Speicher über den `memIO`-Treiber direkt angesprochen werden (vgl. Abschnitt 3.6.2). Im Darkwing-System wird ein Speicherbereich im PC-Speicher angesprochen und mit den Methoden `loadMem` bzw. `getMem` zwischen PC-Speicher und Darkwing-SDRAM-Speicher transferiert. Dabei wird die virtuelle Adresse für den Zeigerzugriff der Software und die physikalische Adresse für den Zugriff des Koprozessors aus der programmierbaren Logik verwendet. Für einen effizienten Einsatz des SDRAMs muss die bevorzugte Speicherbank (vgl. Abschnitt 1.3.4) angegeben werden.

Methode	Parameter
`openMem`	`size, bank`
`freeVirtMem`	`*virtAddrPtr`
`freePhysMem`	`physAddr`
`*getVirtAddr`	`physAddr, size`
`loadMem`	`physAddr, *virtAddrPtr, size`
`getMem`	`physAddr, *virtAddrPtr, size`
`flushCache`	

Befehl an den Koprozessor Die folgenden Methoden schreiben jeweils die entsprechenden Befehle in den Befehlsspeicher des Koprozessors.

Methode	Parameter
`parent 1`	`bufferAddr`
`parent 2`	`bufferAddr`
`parentOff 2`	`bufferAddr`
`offspring`	`bufferAddr`
`jumpNotZero`	`jumpAddr`
`nextChromo`	
`onePointCross`	
`setProbabUni`	`probabUni`
`setProbabGau`	`probabGau`
`maskNull`	`maskLength`
`maskOne`	`maskLength`
`start`	`probabUni`
`sleep`	`probabGau`
`isAsleep`	`maskLength`
`setWidthGau`	`width`
`wDebugReg`	`debug-data`

Glossar

HAGEN	Heidelberg AnaloG Evolvable neural Network
ADC	Analog-Digital-Converter
ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Block
CPLD	Complex Programmable Logic Device
CPS	Connections Per Second
DAC	Digital-Analog-Converter
DCM	Digital Clock Manager
DCR	Device Control Register
DDR	Double Data Rate
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
FF	Flip-Flop
FIFO	First In First Out
FPGA	Field Programmable Gate Array
HAE	Hardware-Abstraktionsebene
HANNEE	Heidelberg Analog Neural Network Evolution Environment
I/O	Input/Output
IRQ	Interrupt Request
KNN	Künstliche Neuronale Netzwerke
LUT	Look-up Table
LVDS	Low Voltage Differential Signal
MGT	Multi-Gigabit-Transceiver
MMU	Memory Management Unit
OCM	On-Chip Memory
PCI	Peripheral Component Interconnect
PLB	Processor Local Bus
SDRAM	Synchronous Dynamic Random Access Memory
SMT	Surface Mount Technology
SO-DIMM	Small Outline Dual Inline Memory Module
SRAM	Static Random Access Memory
SRL	Shift Register LUT
VHDL	VHSIC Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very-Large-Scale Integration

Literaturverzeichnis

[1] ACE: The Adaptive Communication Environment. Distributed Object Computing (DOC) group, Vanderbilt University, Nashville, Washington University, and University of California, Irvine,
`http://www.cs.wustl.edu/ schmidt/ACE.html`.

[2] C. Aporntewan and P Chongstitvatana. A hardware implementation of the compact genetic algorithm. In *Proceedings of the 2001 Congress on Evolutionary Computation*, 2001.

[3] S. Areibi, M. Moussa, and Koonar G. A genetic algorithm hardware accelerator for VLSI circuit partitioning. *ISCA International Journal of Computers and Their Applications*, 12(3):163–180, 3.

[4] Thomas Bäck, F. Hoffmeister, and H. Schwefel. A survey of evolution strategies. In Richard K. Belew and Lashon B. booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 2–9, San Diego, CA, Jul 1991. Morgan Kaufmann.

[5] J. Becker. Ein FPGA-basiertes Testsystem für gemischt analog/digitale ASICs. Diploma thesis (german), University of Heidelberg, HD-KIP-01-11, 2001.

[6] N. Bertschinger and T. Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413 – 1436, Jul 2004.

[7] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Walton Street, Oxford, 1995.

[8] E. K. Blum and L. K. Li. Approximation theory and feedforward networks. *Neural Networks*, 4(4):511–515, 1991.

[9] D.P Bovet and M. Cesati. *Understanding the Linux Kernel*. O'Reilly & Associates, Inc., 2003.

[10] A. Breidenassel. *A High-Dynamic Range CMOS Image Sensor with Adaptive Integration Time Control*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2005.

[11] D. Brüderle. Implementing spike-based computation on a hardware perceptron. Diploma thesis, University of Heidelberg, HD-KIP-04-16, 2004.

[12] Alan Clements. *The Principles of Computer Hardware*. Oxford Science Publications, 1885.

[13] B. Cohen. *VHDL Coding Styles and Methodologies, 2nd Edition.* Kluwer Academic Publishers, 2001.

[14] IBM Corporation. *64-Bit Processor Local Bus - Architecture Specifications*, Mar 2001.

[15] Charles R. Darwin. *On the Origin of Species.* John Murray, London, 1859.

[16] K. DeJong. The analysis and behaviour of a class of genetic adaptive systems. *PhD thesis*, 1975.

[17] IBM Microelectronics Division. *CoreConnect Bus Architecture*, GK10-3116-00 edition, Jan 1999.

[18] J. Fieres. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2006. In Vorbereitung.

[19] J. Fieres, A. Grübl, S. Philipp, K. Meier, J. Schemmel, and F. Schürmann. A platform for parallel operation of VLSI neural networks. In *Proc. of the 2004 Brain Inspired Cognitive Systems Conference (BICS2004)*, University of Stirling, Scotland, UK, 2004.

[20] J. Fieres, S. Hohmann, E. Mueller, J. Schemmel, T. Schmitz, and F. Schürmann. Das Softwarepaket HANNEE (Heidelberg Analog Neural Network Evolution Environment) wurde in der Electronic Vision(s) Group entwickelt.

[21] L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence Through Simulated Evolution.* John Wiley & Sons, 1966.

[22] K. Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1:119–130, 1988.

[23] The GNU Compiler Collection. Free Software Foundation, Inc., 59 Temple Place, Boston, MA, USA, `http://gcc.gnu.org/`.

[24] Wulfram Gerstner and Werner Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity.* Cambridge University Press, 2002.

[25] Andreas Grübl. Eine FPGA-basierte Platform für Neuronale Netze. Diploma thesis (german), University of Heidelberg, HD-KIP-03-2, 2003.

[26] J. L. Hennessy and D. A. Patterson. *Computer Architecture - A Quantitative Approach.* Morgan Kaufmann Publishers, INC, 1990.

[27] S. Hettich and S. D. Bay. The UCI KDD archive. University of California, Department of Information and Computer Science, Irvine, USA, `http://kdd.ics.uci.edu`, 1999.

[28] S. Hohmann, J. Schemmel, and T. Schmitz. *Mixed-Mode Neural Networks Adapted for Evolutionary Algorithms*, in M. Valle: Smart Adaptive Systems on Silicon, pages 209–226. Kluwer Academic Publishers, ISBN 1-4020-2743-5, 2004.

[29] S.G. Hohmann. *Stepwise Evolutionary Training Strategies for Hardware Neural Networks.* PhD thesis, Ruprecht-Karls-Universität Heidelberg, Heidelberg, 2005.

[30] S.G. Hohmann, J. Fieres, K. Meier, J. Schemmel, T. Schmitz, and F. Schürmann. Training fast mixed-signal neural networks for data classification. In *Proceedings of the 2004 International Joint Conference on Neural Networks (IJCNN'04)*, pages 2647–2652. IEEE Press, 2004.

[31] Steffen G. Hohmann, Johannes Schemmel, Felix Schürmann, and Karlheinz Meier. Predicting protein cellular localization sites with a hardware analog neural network. In *Proceedings of the Int. Joint Conf. on Neural Networks*, pages 381–386. IEEE Press, Jul 2003.

[32] J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. of Computing*, 2:88–105, 1973.

[33] PLX Technology Inc. *PLX 9054 Datasheet*. 870 Maude Avenue, Sunnyvale, CA 94085, USA.

[34] Infineon Technologies, www.infineon.com. *256-MBit Double Data Rate SDRAM Data Sheet*, Jan 2003.

[35] Integrated Device Technology, Inc., www.idt.com. *128K x 36 Synchronous ZBT SRAMs*, Oct 2000. IDT71V2556, IDT71V2558.

[36] Teuvo Kohonen. Correlation matrix memories. *IEEE Transactions on Computers*, C-21:353–359, 1972.

[37] J. Langeheine. *Intrinsic Hardware Evolution on the Transistor Level*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2005.

[38] G. Lehmann, B. Wunder, and M. Selz. *Schaltungsdesign mit VHDL*. Franzis' Verlag, Poing, 1994. ISBN 3-7723-6163-3.

[39] Jungo Ltd. *Windriver*. 1 Hamachshev Street, P.O.Box 8493, Netanya 42504, ISRAEL.

[40] Jungo Ltd. *Windriver 6 User's Manual*. Netanya, 2003.

[41] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[42] MATLAB. version 6, release 12.1, The Mathworks Inc., 3 Apple Hill Drive, Natick, MA, USA,
`http://www.mathworks.com/products/matlab/`.

[43] Maxim Integrated Products. *+3V, Quad, 12-Bit Voltage-Output DAC with Serial Interface*, max5253 edition, September 1996. `www.maxim.com`.

[44] Maxim Integrated Products. *Remote/Local Temperature Sensor with SMBus Serial Interface*, max1617 edition, March 1998. `www.maxim.com`.

[45] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, pages 127–147, 1943.

[46] Mentor Graphics Corporation. *ModelSim SE Tutorial*, Version 6.1 edition, 2005. `http://www.model.com`.

[47] Micron Technology, Inc., `www.micron.com`. *Synchronous DRAM*, MT48LC8M16 edition, Mar 2001.

[48] Micron Technology, Inc., `www.micron.com`. *Double Data Rate (DDR) SDRAM*, Mar 2003. MT64V32M8.

[49] Micron Technology, Inc., `www.micron.com`. *Small-Outline DDR SDRAM Module*, Mar 2003. MT8VDDT3264HD.

[50] Marvin Minsky and Seymour Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[51] P. Moerland and E. Fiesler. Neural network adaptions to hardware implementations. In E. Fiesler and R. Beale, editors, *The Handbook of Neural Computation*, New York, Jan 1997. Institute of Physics Publishing and Oxford University Publishing.

[52] Dominik Niedenzu. Aufbau eines binären Neocognitrons. Diploma thesis (german), University of Heidelberg, HD-KIP-03-11, 2003.

[53] D. A. Patterson and J. L. Hennessy. *Computer Organization & Design*. Morgan Kaufmann Publishers, INC, 1998.

[54] S. Perkins, R. Porter, and N.R. Harvey. Everything on the chip: A hardware-based self-contained spatially-structured genetic algorithm for signal processing. In *Evolvable Systems: From Biology to Hardware: Proc. 3rd International Conference on Evolvable Systems (ICES 2000)*, 2000.

[55] S. Philipp. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2006. In Vorbereitung.

[56] W.H. Press, W.T. Vetterling, S.A. Teukolsky, and B.P. Flannery. *Numerical Recipes in C++: the art of scientific computing*. 2002.

[57] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[58] M. Reuß. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2006. In Vorbereitung.

[59] Paul Rojas. *Theorie der neuronalen Netze: Eine systematische Einführung*. Springer Verlag, Berlin, Heidelberg, New York, 1996.

[60] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[61] F. Rosenblatt. Perceptron simulation experiments. In *Proceedings of the IRE*, pages 301–309, 1960.

[62] Alessandro Rubini. *Linux Device Drivers*. O'Reilly, 101 Morris Street, Sebastopol, CA 95472, 1998.

[63] D. E. Rumelhart, G. E. Hinton, and Williams R.J. Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, I:318–362, 1986.

[64] Samsung Eletronics CO., LTD., `www.samsung.com`. *256Kx36 Flow-Through NtRAM*, Nov 1999. KM736V847, KM718V947.

[65] J. Schemmel, F. Schürmann, S. Hohmann, and K. Meier. An integrated mixed-mode neural network architecture for megasynapse ANNs. In *Proceedings of the 2002 International Joint Conference on Neural Networks IJCNN'02*, pages 2704–2710. IEEE Press, 2002.

[66] T. Schmitz, S. Hohmann, K. Meier, J. Schemmel, and F. Schürmann. Speeding up Hardware Evolution: A Coprocessor for Evolutionary Algorithms. In Andy M. Tyrrell, Pauline C. Haddow, and Jim Torresen, editors, *Proceedings of the 5th International Conference on Evolvable Systems ICES 2003*, pages 274–285. Springer Verlag, 2003.

[67] F. Schürmann, S. Hohmann, J. Schemmel, and K. Meier. Towards an Artificial Neural Network Framework. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R.S. Zebulum, editors, *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 266–273. IEEE Computer Society, 2002.

[68] F. Schürmann, K. Meier, and J. Schemmel. Edge of Chaos Computation in Mixed Mode VLSI – "A Hard Liquid". In L.K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, Cambride, 2005. MIT Press.

[69] Felix Schürmann. *Exploring Liquid Computing in a Hardware Adaptation: Construction and Operation of a Neural Network Experiment.* PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2005.

[70] Felix Schürmann, Steffen G. Hohmann, Karlheinz Meier, and Johannes Schemmel. Interfacing binary networks to multi-valued signals. In *Supplementary Proceedings of the Joint International Conference ICANN/ICONIP*, pages 430–433. IEEE Press, 2003.

[71] A.D Scott, S. Seth, and A. Samal. A synthesizable VHDL coding of a genetic algorithm. Technical Report UNL-CSE-07-009, University of Nebraska-Lincoln, Nov 1997.

[72] B. Shackleford, M. Tanaka, R.J. Carter, and G. Snider. High-performance cellular automata random number generators for embedded probabilistic computing systems. In *Evolvable Hardware, 2002. Proceedings. NASA/DoD Conference on*, 2002.

[73] Barry et. al. Shackleford. A high-performance, pipelined, FPGA-based genetic algorithm machine. *Genetic Programming and Evolvable Machines*, 1(2):33–60, Mar 2001.

[74] Ellen Siever. *Linux in a nutshell.* O'Reilly, 101 Morris Street, Sebastopol, CA 95472, 2000.

[75] Alexander Sinsel. Linuxportierung auf einen eingebetteten PowerPC 405 zur Steuerung eines neuronalen Netzwerkes. Diploma thesis (german), University of Heidelberg, HD-KIP-03-14, 2001.

[76] Andrew S. Tannenbaum. *Computer Networks*. prentice Hall PTR, Upper Saddle River, New Jersey 07458, 2003.

[77] Trolltech AS. The Qt application development framework. Waldemar Thranes gate, 98, NO-0175 Oslo, Norway, `http://www.trolltech.com/products/qt/`.

[78] M. Valle. Analog VLSI implementations of neural networks with supervised on-chip-learning. *Analog Integrated Circuits and Signal Processing*, 33:263–287, 2002.

[79] S. Wakabayashi, T. Koide, N. Toshine, M. Yamane, and H Ueno. Genetic algorithm accelerator gaa-ii. In *ASP-DAC00: Proceedings of the 2000 conference on Asia South Pacific design automation*, pages 9–10, New York, NY, USA, 2000. ACM Press.

[80] Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[81] Xilinx, Inc., `www.xilinx.com`. *Virtex E Datasheet*.

[82] Xilinx, Inc., `www.xilinx.com`. *Virtex-II Pro Platform FPGA Developer's Kit*, 2002.

[83] Xilinx, Inc., `www.xilinx.com`. *Synthesizable 400 Mb/s DDR SDRAM Controller*, 2003. XAPP253.

[84] Xilinx, Inc., `www.xilinx.com`. *PowerPC 405 processor block reference guide*, 2004.

[85] Xilinx, Inc., `www.xilinx.com`. *RocketIO Transceiver User Guide*, 2004.

[86] Xilinx, Inc., `www.xilinx.com`. *Virtex-II Pro Platform FPGAs*, 2004. DS083-1, DS083-2, DS083-3, DS083-4.

[87] Xilinx, Inc., `www.xilinx.com`. *Xilinx XC9536XL High Performance CPLD*, 2004.

[88] Xilinx, Inc., `www.xilinx.com`. *Xilinx ISE 7 Software Manuals and Help*, 2005.

[89] Y. Yamaguchi, A. Miyashita, T. Maruyama, and T Hoshino. A co-processor system with a Virtex FPGA for evolutionary computation. In *Field-Programmable Logic and Applications*, 2000.

Danksagung

an dieser Stelle möchte ich all denen ganz herzlich danken, die mich bei der Erstellung dieser Arbeit unterstützt haben. Mein besonderer Dank gilt:

- Herrn Prof. Dr. K. Meier für die nette und offene Aufnahme in die Arbeitsgruppe Electronic Vision(s).
- Herrn Prof. Dr. Reinhard Männer für die freundliche Übernahme der Zweitkorrektur.
- Herrn Dr. J. Schemmel für viele fachliche Diskussionen und seine kompetenten Ratschläge in allen Belangen von Soft- und Hardware, aus denen ich sehr viel gelernt habe.
- Herrn Andy Grübl für die hervorragende Nathanhardware, sehr gründliches Korrekturlesen und für viele interessante Gespräche bei Kaffee oder Tannenzäpfle.
- Herrn Philipp für produktiven Gedankenaustausch in Linuxkernel-Fragen, bezüglich programmierbarer Logik und in vielen anderen Gebieten.
- Herrn Dr. Hohmann für seine Trainingssoftware und die erfolgreichen evolutionären Experimente.
- Herrn Dr. Schürmann für konstruktive Zusammenarbeit am HAGEN Chip und für die Pflege der Linuxwelt der Arbeitsgruppe.
- Meinem ehemaligen Zimmernachbarn Dr. Maucher für ein offenes Ohr in allen Fragen.
- Allen Mitgliedern der Arbeitsgruppe Electronic Vision(s) für die freundschaftliche Arbeitsumgebung.
- Meinen Eltern für ihre liebevolle Unterstützung nicht nur während meiner Ausbildung.
- Meinem Freund Sven für unzählige Gespräche, Abende beim Bierchen und frühzeitiges "Wann krieg ich was zu lesen?"-Fragen.
- Meiner liebsten Tine für ihr Vertrauen und die Freude, die sie in mein Leben bringt.
- Meiner Tochter Hannah für ihr wunderbares zahnloses Lächeln.