Konfigurierbare Hardwarebeschleuniger für selbst-organisierende Karten

Zur Erlangung des akademischen Grades

DOKTOR-INGENIEUR (Dr.-Ing.)

der Fakultät für Elektrotechnik, Informatik und Mathematik,
Institut für Elektrotechnik und Informationstechnik
der Universität Paderborn

von

Dipl.-Ing. Christopher Pohl
aus Verl

Referent: Prof. Dr.-Ing. Ulrich Rückert
Koreferentin: Prof. Dr. Erzsébet Merényi

Paderborn, den 26. April 2010
Tag der mündlichen Prüfung: 6. Dezember 2010
Diss. EIM-E/269

Für Anne, Mathis und Line, und alle, die ich jetzt noch nicht sehen kann.

Inhaltsverzeichnis

Einleitung

Datenverarbeitung im Allgemeinen ist eine Technologie, die besonders in diesen Tagen breit in der Öffentlichkeit diskutiert wird. Dabei geht es vor Allem um die Abwägung von Persönlichkeitsrechten gegen Gemeinschaftsinteressen, die sich per se konträr gegenüberstehen. Diese Arbeit befasst sich mit einem Algorithmus aus dem Bereich der Datenverarbeitung, genauer der Datenbankauswertung (engl. Data-Mining), welcher anhand eines Anwendungsszenarios untersucht wird, das vermutlich niemandes Persönlichkeitsrechte tangiert. Die Rede ist von der Auswertung sog. hyperspektraler Bilder, die an Bord von Satelliten oder Drohnen von diesem oder entfernten Planeten aufgenommen werden.

Bei diesen Bildern entspricht jeder Bildpunkt einer spektralen Signatur des untersuchten Bereiches, und erlaubt Rückschlüsse auf die reflektierenden Materialien, etwa Mineralien, Pflanzen oder Kontaminationen. Die Auswertung dieser Bilder beginnt, nachdem atmosphärische Effekte heraus gerechnet wurden, mit der Gruppierung gleichartiger bzw. ähnlicher Bildpunkte, die auf ähnliche Begebenheiten im korrespondierenden Bereich am Boden schließen lassen. Die Suche nach solchen Ähnlichkeiten ist ein bekanntes Problem, zu dem verschiedene Lösungsansätze propagiert wurden. Für die Auswahl eines dieser Verfahren muss das Anwendungsszenario betrachtet werden, dessen hervorstechendste Merkmale große Datenmengen und die beschränkten Ressourcen hinsichtlich verfügbarer Energie und Platz sind. Ein Algorithmus, der das Potential hat diese Anforderungen zu erfüllen, ist die selbstorganisierende Karte (eng. self-organizing map, SOM) von Teuvo Kohonen [77].

Bei diesem Algorithmus handelt es sich um ein sog. künstliches neuronales Netz (KNN), da die Funktionsweise von biologischen neuronalen Netzen kopiert wurde. So finden sich im Neocortex sowohl ein- als auch zweidimensionale Strukturen, die für die Verarbeitung multidimensionaler Reize verantwortlich sind. Beobachtungen am lebenden Objekt zeigen, dass „ähnliche“ Reize in räumlich nahen Gebieten Aktivität auslösen, während „unähnliche“ Reize in entfernten Regionen Aktivität auslösen. Es wurde ein mathematisches Verfahren entwickelt, dass

dieses Verhalten für eine d_G-dimensionale Anordnung von Rechenelementen, den sog. Neuronen, nachbildet. Dieses sog. Lernverfahren unterscheidet sich von vielen anderen KNN dadurch, dass der Lernvorgang nach der Wahl geeigneter initialer Parameter nicht weiter beaufsichtigt werden muss, man spricht von einem unüberwachten Lernvorgang. Als Ergebnis des Lernvorganges stellt sich eine geordnete Karte dar, in der sich der oben beschriebene Effekt manifestiert: ähnliche Reize ordnen sich in benachbarten Gebieten an, so dass diese als Gruppen (engl. Cluster) aus der Karte extrahiert werden können.

Grundsätzlich hat dieses Verfahren eine Komplexität, die mit der Anzahl der Datenpunkte und der Anzahl der Rechenelemente wächst; nicht zuletzt aufgrund des biologischen Ursprungs dieses Verfahrens ist der Algorithmus aber inhärent parallel, d.h. die einzelnen Rechenelemente können weitgehend unabhängig voneinander arbeiten. Dies ist insbesondere hinsichtlich der eingeschränkten Ressourcen im oben beschriebenen Anwendungsszenario von Bedeutung, weil eine parallele Implementierung des Algorithmus wesentlich energieeffizienter arbeiten kann, als eine sequentielle. Da der Algorithmus bezogen auf die mathematischen Operatoren verhältnismäßig einfach aufgebaut ist, und die Anzahl potentiell parallel arbeitender Elemente deutlich größer ist, als parallele Einheiten in aktuellen, eng gekoppelten Multiprozessoren verfügbar sind, kommt eine Realisierung in dedizierter Hardware in Frage.

In der vorliegenden Arbeit werden verschiedene Aspekte einer Hardwareimplementierung für selbst-organisierende Karten untersucht und entsprechende Systeme entwickelt, deren Leistungsfähigkeit und Ressourcenbedarf dann anhand eines Anwendungsszenarios untersucht wird. Die Struktur der Arbeit orientiert sich am Entwicklungsprozess des Hardwarebeschleunigers: in **Kapitel 1** wird das generelle Umfeld der selbst-organisierenden Karten reflektiert und bestehende Implementierungen und Vergleichsmaße diskutiert. In **Kapitel 2** folgt eine Untersuchung zu möglichen Anpassungen des Algorithmus für effiziente Hardwarerealisierungen und deren Auswirkung auf das Verhalten des Algorithmus. Basierend darauf werden Mindestanforderungen an eine Hardwarerealisierung gestellt, die dann in die in **Kapitel 3** folgenden, prinzipiellen Architekturvarianten einfließen. Anhand von Vorhersagemodellen werden dann Leistungsfähigkeit und Ressourcenbedarf untersucht und für ein Anwendungsszenario evaluiert. **Kapitel 4** beschäftigt sich mit Techniken zu Test und Integration von digitalen Schaltungen in sog. Rapid-Prototyping Systeme im Allgemeinen und den SOM Beschleunigern im Besonderen. Abschließend werden in **Kapitel 5** die hier entwickelten Varianten mit den anfänglich erwähnten Referenzen verglichen, und die Zukunftsfähigkeit des Konzeptes untersucht.

Kapitel 1

Selbst-organisierende Karten

Im folgenden Kapitel wird zunächst der SOM-Algorithmus nach Kohonen beschrieben und in den Kontext der künstlichen neuronalen Netze eingeordnet. Im Anschluss werden verschiedene Maße für die Geschwindigkeit des Lernprozesses und die Qualität des Endergebnisses präsentiert. Danach werden Methoden zur Visualisierung der angelernten Karte sowie einige Anwendungsbeispiele gegeben.

1.1 Selbst-organisierende Karten nach T. Kohonen

Bei den künstlichen neuronalen Netzen (KNN) [59] handelt es sich um eine Gruppe von Algorithmen, die bestimmten Strukturen innerhalb biologischer neuronaler Netze nachempfunden sind. Die zentralen Einheiten der KNN sind wie bei den biologischen Vorbildern die Neuronen, die sich aus einer Recheneinheit (Soma), mehreren Eingängen (den Dendriten) und mindestens einem Ausgang (dem Axon) zusammensetzen. Grundidee der KNN ist es nun, das Verhalten einzelner Neuronen oder ganzer Netze durch mathematische Funktionen zu approximieren. Diese Approximation kann zeitkontinuierlich (analog) oder zeitdiskret (digital) erfolgen, wobei im Rahmen dieser Arbeit nur die digitalen KNN betrachtet werden sollen. In diesem Fall werden die Neuronen durch zeitdiskrete Transferfunktionen f_{Trans}, abhängig von den zu verarbeitenden Eingangsdaten $\mathbf{x}$ und dem internen Zustand $\mathbf{m}(\tau\Delta t)$, also $f_{Trans}(\mathbf{x}, \mathbf{m}(\tau\Delta t))$ mit $\tau \in \mathbb{N}$, bzw. einer Komposition beliebig vieler solcher Funktionen abgebildet. Die Komposition dieser Funktionen beschreibt die Struktur oder Verschaltung des KNN.

Neben der Struktur werden KNN durch das Lernverfahren charakterisiert, welches die Anpassung der internen Zustände des Netzes, die im Zusammenhang mit KNN *Gewichte* oder *Referenzen* genannt werden, auf Eingabedaten charakterisiert, also $\mathbf{m}((\tau+1)\Delta t) = f_{Lern}(\mathbf{x}, \mathbf{m}(\tau\Delta t))$. Die Funktion f_{Lern} wird wesentlich

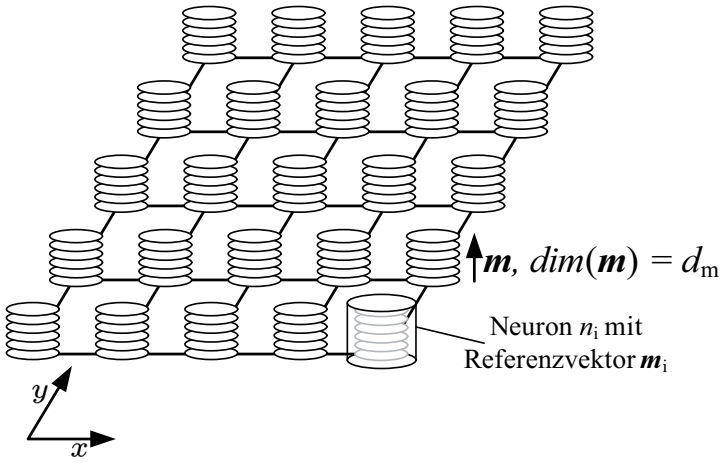


Abbildung 1.1: Eine zweidimensionale Karte mit 5×5 Neuronen $n_i \in N$, die Referenzvektoren m haben die Dimension 6

durch die Art der Rückkopplung gekennzeichnet; entweder wird diese von außerhalb des neuronalen Systems, z.B. durch einen menschlichen Beobachter erzeugt, oder aber intrinsisch durch das System selber. Für diese beiden Lernverfahren werden auch die Begriffe überwachtes bzw. unüberwachtes Lernen verwendet, wobei der Vorteil des unüberwachten Lernens für autonome Anwendungen offensichtlich ist. Einer der bekanntesten Vertreter der unüberwacht lernenden Netze ist die von Teuvo Kohonen im Jahre 1983 präsentierte selbst-organisierende Karte (engl. Self Organizing Map, SOM), die vor allem aufgrund ihrer Eigenschaften als Vektorquantisierer und aufgrund der Visualisierungsmöglichkeiten bei der Untersuchung hochdimensionaler Daten eingesetzt werden. Bei diesem Netztyp handelt es sich um eine Anordnung von Neuronen in einem d_G-dimensionalen Gitter, bei der alle Neuronen mit denselben Eingängen verbunden sind (siehe Abbildung 1.1). In der Regel wird $d_G \in [1, 2, 3]$ gewählt, da hier eine Visualisierung, siehe Abschnitt 1.2.1, direkt möglich ist.

Das biologische Vorbild der SOM befindet sich im Neokortex, dem entwicklungsgeschichtlich jüngsten Teil der Großhirnrinde, in dem beobachtet wurde, dass ähnliche sensorische Stimuli in benachbarten Regionen verarbeitet werden, während bei weniger ähnlichen Stimuli auch die Verarbeitung örtlich getrennt stattfindet. Dieser Effekt wird in der SOM über die Berücksichtigung von Nachbarschaftsbeziehungen zwischen Neuronen im Lernverfahren abgebildet. Der resultierende Algorithmus erzeugt eine sog. topografieerhaltende Abbildung $X \to M$ vom Eingaberaum $X = \{\mathbf{x}_1, ..., \mathbf{x}_k\}$ nach den Referenzen der Karte $M = \{\mathbf{m}_1, ..., \mathbf{m}_l\}$, wobei $k = |X|$ die Anzahl der Datenpunkte im Eingaberaum und $l = |M|$ die Anzahl der Neuronen im Netz bezeichnen. Näheres zu den Eigenschaften dieser Abbildung ist im Abschnitt 1.4 zu finden.

Im folgenden Abschnitt wird der SOM-Algorithmus definiert, wobei hier an-

stelle von $\tau\Delta t$ stets nur t notiert wird, da in dieser Arbeit ausschließlich zeitdiskrete Algorithmen betrachtet werden.

1.2 Der SOM-Algorithmus

Das Lernverfahren gliedert sich in drei Phasen:

1. Initialisierung

   In dieser Phase werden die Gewichtsvektoren $\mathbf{m}_i$ der einzelnen Neuronen mit Anfangswerten initialisiert. Die Gewichtsvektoren beschreiben einen Punkt im d_m-dimensionalen Raum und besitzen die gleiche Dimension wie die Datenvektoren $\mathbf{x}_i$ aus der Menge X, im Folgenden als Eingabedatensatz bezeichnet. Die Initialisierung der Gewichtsvektoren ($\mathbf{m}_i(t)$ mit $t = 0$) kann auf verschiedene Arten geschehen: Bei der *zufälligen Initialisierung* werden den Komponenten zufällige Werte aus dem Intervall [0..1] zugewiesen. Aus dieser ungeordneten Anfangskonfiguration entsteht in der Lernphase eine topografisch geordnete Konfiguration. Anstelle der zufälligen Initialisierung können auch Vektoren aus dem Eingabedatensatz X gewählt werden, um von Anfang an eine Approximation an $p(X)$ zu erhalten, auch wenn diese noch topografisch ungeordnet ist. Bei der sog. linearen Initialisierung wird eine topografisch bereits geordnete Anfangskonfiguration erzeugt. Dazu werden die beiden Eigenvektoren der Autokorrelationsmatrix von X mit den größten Eigenwerten bestimmt. In die von den beiden gewählten Eigenvektoren aufgespannte Ebene wird ein rechteckiges Gitter gelegt, dessen Schwerpunkt mit dem von X zusammenfällt, und die gleichen Dimensionen wie die gewählten Eigenvektoren hat. Die Gitterpunkte werden dann als Initialwerte für die $\mathbf{m}_i$ gewählt. Diese Initialisierung verringert in der Regel die Anzahl der Epochen, die benötigt werden, um ein bestimmtes Qualitätskriterium zu erreichen, ist aber sehr aufwändig zu berechnen. Im Folgenden wird von der zufälligen Initialisierung ausgegangen.

2. Abstandsberechnung und Suche des Erregungszentrums

   In dieser Phase wird ein Vektor $\mathbf{x}(t)$ zufällig aus X ausgewählt und dasjenige Neuron n_c gesucht, dessen Gewichtsvektor $\mathbf{m}_c$ den kleinsten Abstand zu $\mathbf{x}(t)$ hat.

   $$||\mathbf{x} - \mathbf{m}_c|| = \min_i\{||\mathbf{x} - \mathbf{m}_i||_p\} \tag{1.1}$$

   Das Distanzmaß ist hierbei nicht vorgegeben, allerdings werden in den meisten Fällen die euklidische ($p = 2$), die Manhattan- ($p = 1$) oder die Unendlichnorm ($p = \infty$) verwendet. Die Wahl der Norm spielt besonders in hochdimensionalen Eingaberäumen eine wichtige Rolle [138, 64, 21], hier können auch Normen mit $p < 1$ sinnvoll sein.

3. Adaption

   In dieser Phase werden die Referenzvektoren des Erregungszentrums (häufig auch Bestmatch genannt) und der umgebenden Neuronen in Richtung des Eingabevektors $\mathbf{x}(t)$ verschoben. Die Stärke der Verschiebung ist dabei abhängig von der Nachbarschaftsfunktion $h_{c,i}(t)$ und der Lernrate α:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \alpha(t) h_{c,i}(t)(\mathbf{x}(t) - \mathbf{m}_i(t)) \tag{1.2}$$

   Dabei ist $h_{c,i} = h(||\mathbf{r}_c - \mathbf{r}_i||, t)$, wobei $\mathbf{r}_c$ und $\mathbf{r}_i$ die Positionen des Erregungszentrums c und des Neurons i in der Karte bezeichnen. Die Nachbarschaftsfunktion muss für $t \to \infty$ und für $||\mathbf{r}_c - \mathbf{r}_i|| \to \infty$ gegen Null gehen, da andernfalls das Gelernte immer wieder verworfen wird (die Karte kann nicht konvergieren). Für die $h_{c,i}$ wird häufig eine Gaußfunktion verwendet:

$$h_{c,i} = \exp - \frac{||\mathbf{r}_c - \mathbf{r}_i||^2}{2\sigma^2(t)}, \tag{1.3}$$

   In der Regel wird am Anfang des Lernvorgangs eine breite Nachbarschaftsfunktion verwendet, die eine Grobordnung der Karte ermöglicht. In den anschließenden Schritten wird diese fortwährend schmaler gewählt, wobei sich eine *topografische* Ordnung der Gewichtsvektoren herausbildet. In der letzten Phase des Lernvorganges wird die Nachbarschaftsfunktion extrem schmal, so dass unter Umständen nur noch ein einziges Neuron adaptiert wird. In dieser Phase der *Konvergenz* wird die topografische Ordnung nicht mehr verändert, alleine die Fehlerfunktionen

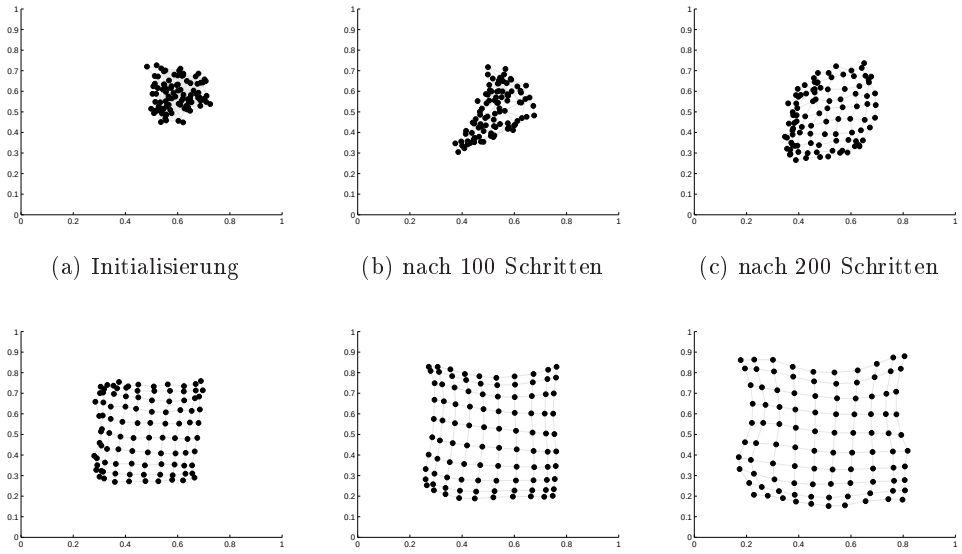
$$E_Q = \int \|\mathbf{x} - \mathbf{m}_c\|^2 p(x)\, dx \text{ im Kontinuierlichen bzw.} \tag{1.4}$$

$$E_Q = \frac{1}{k} \sum_{j=1}^{k} \|\mathbf{m}_c - \mathbf{x}_j\|^2 \text{ im Diskreten,} \tag{1.5}$$

   die den Quantisierungsfehler (siehe Abschnitt 1.3.1) darstellen, werden minimiert. Einige Ergebnisse zum Zusammenhang zwischen den Karten-Parametern und der Konvergenz der Karte finden sich in [53, 116, 48, 49]

Die Schritte 2 (Abstandsberechnung und Suche des Erregungszentrums) und 3 (Adaption) werden entweder eine vordefinierte Anzahl an Schritten, oder aber so lange bis ein bestimmtes Qualitätsmerkmal erreicht ist, wiederholt.

In Abbildung 1.2 ist exemplarisch ein Lernvorgang mit einer 10×10 SOM mit $\dim(\mathbf{m}) = d_m = 2$ dargestellt. Dabei ist in Abbildung 1.2(a) der Zustand nach der Initialisierung gezeigt, während sich die Gewichte in den nachfolgenden Lernschritten *entfalten* und dem Datensatz anpassen. Die Grafiken wurden mit der SOM-Toolbox für Matlab [141] erstellt.



(a) Initialisierung (b) nach 100 Schritten (c) nach 200 Schritten

(d) nach 300 Schritten (e) nach 400 Schritten (f) nach 500 Schritten

Abbildung 1.2: Lernvorgang bei einer 10×10 SOM, $\dim(\mathbf{x}) = \dim(\mathbf{m}) = 2$. Die Eingabedaten wurden zufällig aus dem Einheitsquadrat gewählt.

Im Anschluss an den Lernvorgang können eine Visualisierung (siehe Abschnitt 1.2.1) und eine Abrufphase erfolgen. In der Abrufphase wird wie in der Lernphase zu jedem Datenvektor $\mathbf{x} \in \hat{X}$, wobei $\hat{X}$ der zu klassifizierende Datensatz ist, das Erregungszentrum gesucht. Wenn die Zugehörigkeit des Erregungszentrums zu einer bestimmten Klasse $c_i \in C$ bekannt ist, wird auch der Eingabevektor dieser Klasse zugeordnet. Die Zugehörigkeit der Kartenelemente (Neuronen) zu einer bestimmten Klasse kann zum Beispiel visuell über die U-Matrix (siehe Abschnitt 1.2.1) oder automatisch bestimmt werden. Eine automatische Klassifizierung kann beispielsweise über den k-Means-Algorithmus [89], der auf die gelernte Karte angewandt wird, bestimmt werden. Prinzipiell könnte der k-Means Algorithmus auch auf den ursprünglichen Datensatz angewandt werden, die Laufzeit liegt typischerweise aber weit jenseits der des SOM-Algorithmus mit anschließender Klassifikation. Beide Methoden können auch kombiniert werden, denn während der k-Means-Algorithmus die genaue Anzahl der zu identifizierenden Klassen benötigt, kann diese Anzahl meist mit hinreichender Genauigkeit aus der U-Matrix, Abbildung 1.3, abgelesen werden.

1.2.1 Visualisierung

Eines der hervorstechendsten Merkmale der SOM ist die Verfügbarkeit von intuitiv verständlichen Visualisierungen. Verschiedene Darstellungsformen wie die Komponentenkarten, Unified Distance (U)-Matrix[135] oder P-Matrix[133] geben

Aufschluss über die gelernte Information. Wenn topografische Ordnung und Konvergenz gegeben sind, können so entsprechende Rückschlüsse auf Strukturen innerhalb des Eingabedatensatzes, insbesondere das Vorhandensein von Zusammenballungen von Datenpunkten in sog. Clustern gezogen werden.

Distanzmatrizen

Da die Dimension der Gewichtsvektoren der einzelnen Neuronen in der Regel größer als drei ist, ist eine Abbildung $\mathbb{R}^{d_m} \rightarrow \mathbb{R}^2$ für eine graphische Darstellung der Gewichte sinnvoll. Eine solche Darstellung erzeugen die sog. Distanzmatrizen GD, bei der die Neuronen abhängig vom durchschnittlichen euklidischen Abstand zwischen dem eigenen Gewichtsvektor und dem der Nachbarn eingefärbt wird, also z.B.

$$GD_i = \frac{1}{|J_i|} \sum_{\forall j \in J_i} \|\mathbf{m}_j - \mathbf{m}_i\|, \text{ mit } J_i = (j \mid 0 < \|r_j - r_i\|_2 \leq 2) \tag{1.6}$$

Neben der weit verbreiteten U-Matrix [135] wurden verschiedene weitere Distanzmatrizen, z.B. die P und die U*-Matrix [133, 134] definiert, die außer dem durchschnittlichen Abstand der Gewichtsvektoren noch weitere Größen berücksichtigen. In Abbildung 1.3b ist die U-Matrix einer 7×13 SOM mit hexagonaler Neuronenanordnung nach dem Lernvorgang des in Abbildung 1.3a gezeigten Datensatzes dargestellt. Die dunkel gefärbten Neuronen bezeichnen einen sehr hohen, die heller gefärbten einen niedrigeren Abstand zwischen benachbarten Neuronen (bzw. Gewichtsvektoren). Die Unterteilung in drei zusammenhängende Gebiete ist deutlich zu erkennen, die mit den Clustern im Eingabedatensatz korrelieren. Diese Gebiete wurden in Abbildung 1.3c mit Hilfe des k-Means-Algorithmus separiert.

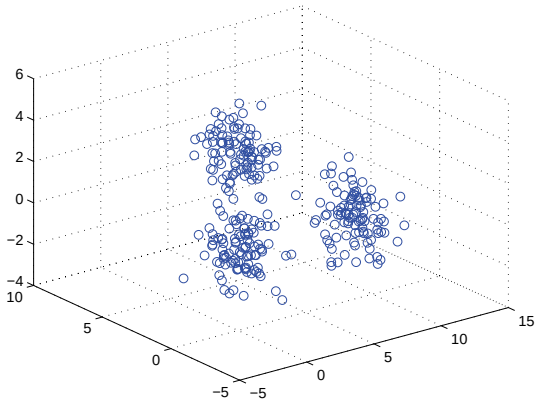
Komponentenkarten

Bei einer Komponentenkarte GK [140] handelt es sich um die graphische Darstellung einer Komponente aller Gewichtsvektoren, also einer Abbildung $\mathbb{R}^{d_G} \rightarrow \mathbb{R}^{d_G}$. Im Prinzip werden die Neuronen in einem Bild bezüglich einer Komponente ihrer Referenzvektoren eingefärbt.

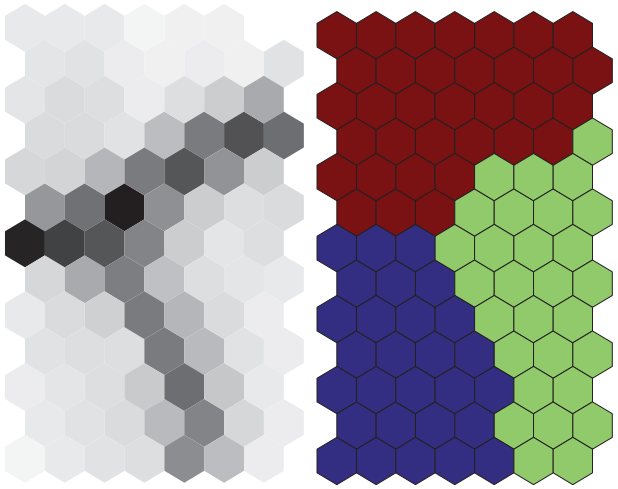
$$GK_i = \mathbf{m}_{i,j}, \tag{1.7}$$

wobei j die gewählte Komponente bezeichnet. Es entstehen d_m d_G-dimensionale Darstellungen der Karte. Diese sind besonders hilfreich beim Auffinden von Zusammenhängen zwischen den einzelnen Komponenten, welche ebenfalls Rückschlüsse auf die Eigenschaften des Eingabedatensatzes zulassen.

In Abbildung 1.4 sind die drei Komponentenkarten zu der zuvor gelernten Karte dargestellt.



(a) Daten



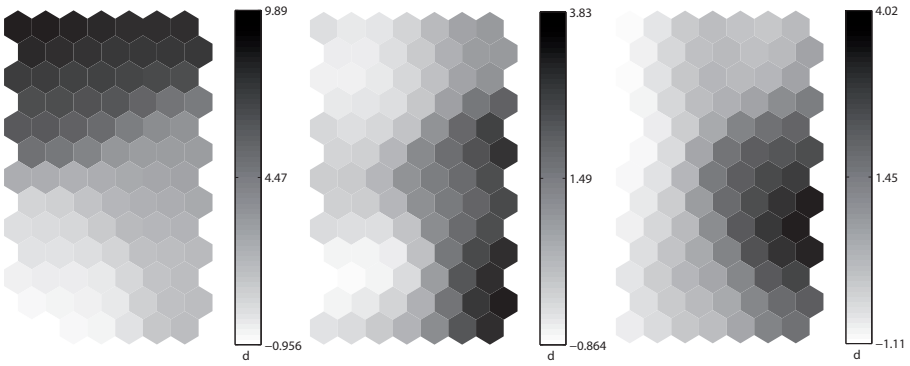
(b) U-Matrix (c) k-Means sep. Karte

Abbildung 1.3: Visualisierung eines künstlichen Datensatzes

Weitere Darstellungsvarianten

Es existiert eine Vielzahl von SOM-spezifischen und auf SOM anwendbaren Visualisierungstechniken, von denen hier nur wenige dargestellt werden können. Detaillierte Erläuterungen zu den Methoden finden sich in [140]

- **Treffer Histogramme:** Diese geben Auskunft über die Verteilung des Datensatzes auf die Elemente der Karte. Dabei wird der gelernte Datensatz (oder ein Validierungsdatensatz) abgerufen, und für jedes Neuron bestimmt, wie häufig es Erregungszentrum war. Eine Darstellungsmöglichkeit ist, die Neuronen der Karte in Abhängigkeit der bestimmten Trefferhäufigkeit einzufärben und/oder die Größe der Neuronen zu verändern.



(a) Komponente 1 (b) Komponente 2 (c) Komponente 3

Abbildung 1.4: Darstellung der Komponentenkarten GK_i zu der Karte aus Abbildung 1.3

- **Markierte Karten:** Bei einem Datensatz mit zuvor bekannten Clustern, können die Neuronen anhand ihrer Clusterzugehörigkeit markiert werden. Auf diese Weise kann sehr einfach die Funktionsfähigkeit einer Implementierung überprüft werden, denn nach einem erfolgreichen Lernvorgang sollten sich Cluster voneinander abgrenzen.

- **Trajektorien:** Bei der Analyse von Zeitreihen kann im Anschluss an den Lernvorgang der Verlauf der Zeitreihe in der Karte dargestellt werden. Dazu wird der Datensatz in der Reihenfolge der Zeitreihe abgerufen und die ermittelten Erregungszentren entsprechend miteinander verbunden.

Eine Übersicht über weitere Standardverfahren liefert [141].

1.2.2 Einsatzgebiete

In den vorangegangenen Abschnitten wurde der SOM-Algorithmus definiert und seine Eigenschaften bezüglich der Visualisierung hochdimensionaler Datensätze beschrieben. Die zugrunde liegende Fähigkeit zur Dimensionsreduktion wird in einigen Anwendungen, hauptsächlich in Forschung und Entwicklung, eingesetzt. Im Folgenden werden kurz die Anwendungsgebiete in der Medizin, Geologie, Regelungstechnik und Robotik umrissen.

Medizin

In der medizinischen Forschung werden im Rahmen von Untersuchungen häufig große Datensätze erstellt, bei denen beispielsweise Zusammenhänge zwischen bestimmten messbaren Faktoren und Krankheiten, insbesondere Krebs, gefunden werden sollen. In [33] werden Zusammenhänge zwischen dem Auftreten be-

stimmter Gensequenzen und dem Auftreten von Krebs im Allgemeinen untersucht. Wang et al. untersuchen in [149] begünstigende Gensequenzen für lymphatische Tumore. Pomeroy et al. [105] beschäftigen sich mit embryonalen Karzinomen des zentralen Nervensystems. Vijayakumar et al. [143] zeigen den Einsatz der SOM in der Diagnose bestimmter Tumore, die anhand verschiedener bildgebender Verfahren analysiert werden. Als Vorteile der SOM wird vor allem das unüberwachte Lernverfahren genannt, das bis dahin unbekannte Zusammenhänge in den großen Datensätzen aufgedeckt hat.

Ein anderes Einsatzgebiet innerhalb der Medizin ist die Untersuchung von Medikamenten. In [23, 131] werden Methoden aufgezeigt, mit denen anhand von selbst-organisierenden Karten die Oberflächen bestimmter Moleküle untersucht werden können. Auf diese Weise werden neue Erkenntnisse über die Rezeptorbindungen von Medikamenten gewonnen; zusätzlich entstehen neue Kategorisierungen von Medikamenten anhand ihrer Oberflächenstruktur.

Geologie

In der Geologie werden selbst-organisierende Karten vorwiegend zur Analyse sog. hyperspektraler Bilder benutzt. Die Unterschiede zur klassischen Fotografie liegen (a) im aufgenommenen Spektrum und (b) in der spektralen Auflösung der Bilder. Systeme wie z.B. AVIRIS [137] lösen den Bereich von 400 bis 2500 nm in 224 Spektralbänder auf, die jedem Bildpunkt zugeordnet werden. Die Bildpunkte können als hochdimensionale Vektoren aufgefasst und mit der SOM analysiert werden. Auf diese Weise lassen sich Informationen zur Bodenbeschaffenheit, Vegetation, Mineralisierung usw. automatisiert gewinnen [95, 145]. In diesem Bereich kommen u. A. auch Standardverfahren [120, 121] zum Einsatz.

Regelungstechnik

In der Regelungstechnik werden selbst-organisierende Karten sowohl als Beobachter als auch als Regler eingesetzt. Chen et al. [38] setzen die SOM als Beobachter in einem passiven Autofocus System (PAF) in einer Miniatur Kamera ein. Die SOM beschleunigt die Scharfstellung, indem sie auf der Basis zuvor gelernter Daten und dem empfangenen Bild eine neue Einstellung der Optik berechnet, die der optimalen Einstellung näher ist als die Nullposition. Dadurch, dass nicht mehr wie im klassischen PAF-System der gesamte Parameterraum durchsucht wird, wird der Autofocusprozess stark beschleunigt. M. Motter benutzt ein Ensemble von SOMs (eine hierarchische Verschaltung) um die Steuerung für einen Transschall Windkanal auf Basis schon bekannter Konfigurationen zu realisieren. Dabei wird der Zustandsraum derartig zerlegt, dass je nach Zustand ein anderer, SOM-basierter Regler die Kontrolle übernimmt. Die Zerlegung des Zustandsraumes und die Aktivierung der entsprechenden Regler werden ebenfalls von einer SOM geregelt.

Der Vorteil dieser Methode ist, dass ein komplettes Reglersystem ohne erneute explizite Identifikation auf Basis von Messdaten erzeugt werden kann. Cho et al. [39] verwenden die SOM ebenfalls, um den Einsatz verschiedener Regler zu koordinieren, allerdings sind hier die eigentlichen Regler auf klassischem Wege dimensioniert worden. Auch in dieser Anwendung wird durch die Zerlegung des Zustandsraumes ein Vorteil gegenüber einem monolithischen Regler erarbeitet, da die Komplexität der einzelnen Regler auf einfache lineare Modelle reduziert wird. Cho et al. heben insbesondere die höhere Effizienz bei der Entwicklung komplexer Regler hervor.

Autonome Roboter

In der Robotik werden SOMs im Wesentlichen in zwei Teildisziplinen eingesetzt, zur Bewegungsplanung und zur Lösung des sog. SLAM Problems. Das SLAM Problem (Simultaneous Location And Mapping, Gleichzeitige Lokalisation und Kartografierung) beschreibt die Notwendigkeit für den Roboter, sowohl die Umgebung zu Kartografieren, als auch die eigene Position in der Umgebung festzustellen bzw. nachzuführen. Neben einigen Ansätzen [58], bei denen die SOM nur zur Auflösung von uneindeutigen Situationen dient, gibt es auch vollständig SOM-basierte Verfahren: Werner et al. [150] beschreiben ein Verfahren zur Lösung des SLAM Problems, bei dem eine selbst-organisierende Karte die Histogramme einer omnidirektionalen Kamera erlernt, und so die eigene Position wiedererkennen kann. Dieser Ansatz unterscheidet sich insbesondere dadurch von Übrigen Ansätzen, dass hier kein geografisches, sondern ein topografisches Modell der Umgebung erlernt wird. In [31] wird ein SOM-basierter Algorithmus verwendet, um Sensorwerte zu abstrahieren. Diese dienen als Eingangswerte für weitere neuronale Netze, die komplexe Verhaltensweisen erlernen können. In allen Beispielen wird die SOM eingesetzt, weil sie selbstständig Repräsentationen der Eingabedaten erlernen, und für andere Anwendungen höherer Schichten zur Verfügung stellen kann.

Bildverarbeitung

Im Rahmen der Bild- und Videoverarbeitung werden selbst-organisierende Karten zur Berechnung einer reduzierten Farbpalette eingesetzt, weil sie schnell eine Lösung nahe am Optimum finden [123]. Kim et al. verwenden in [74] einen SOM-basierten Quantisierer zur Komprimierung medizinischer Bilder. In [109, 82, 124] werden spezielle SOM-Hardwarebeschleuniger für die Farbquantisierung entwickelt, die in Abschnitt 1.6 verglichen werden. Eine Zusammenfassung älterer Publikationen auf diesem Gebiet findet sich in [47].

1.3 Evaluierung des Lernvorganges

Um die Vergleichbarkeit verschiedener Implementierungen des SOM-Algorithmus bzw. seiner Varianten (siehe Abschnitt 1.5) gewährleisten zu können, müssen Maße definiert werden, die unabhängig von der Implementierung den Lernvorgang und das Lernergebnis bewerten können. Diese werden, teilweise unter Zuhilfenahme der Ergebnisse aus Gleichung 1.4, in Kapitel 2 eingesetzt, um verschiedene Hardware-nahe Implementierungen zu vergleichen.

1.3.1 Qualitätsmaße

Die Definition von aussagekräftigen Qualitätsmaßen für die Bewertung der Lernergebnisse ist insbesondere deshalb wichtig, weil die Eigenschaften des betrachteten Datensatzes in der Regel nicht a-priori bekannt sind. Daher muss festgestellt werden, ob die gelernte Karte tatsächlich die topografischen Eigenschaften der Eingabedaten widerspiegelt und ob die Karte konvergiert ist. Dazu wurden verschiedene Maße präsentiert:

Quantisierungsfehler

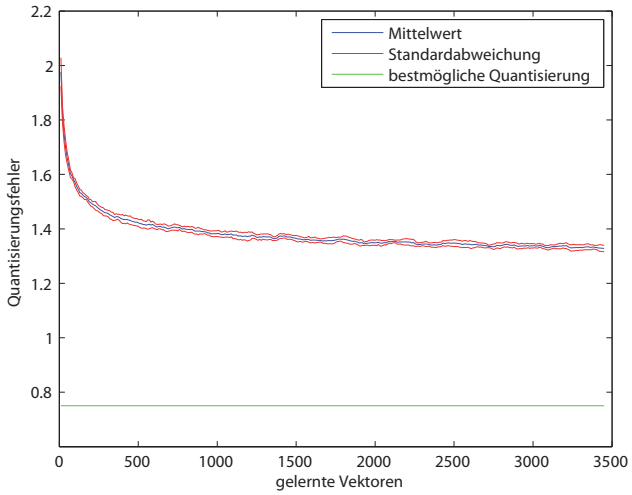
Der SOM-Algorithmus dient in seiner Eigenschaft als Vektor-Quantisierer dazu, zu einer gegebenen Menge von Punkten $\mathbf{x} \in \mathbb{R}^{d_m}$ eine repräsentative Punktmenge $\mathbf{m} \in \mathbb{R}^{d_m}$, die sog. Referenzen zu finden, die die folgende Fehlerfunktion, den *Quantisierungsfehler* minimieren:

$$E_Q = \frac{1}{k} \sum_{j=1}^{k} \|\mathbf{m}_{c,j} - \mathbf{x}_j\|^2, \tag{1.8}$$

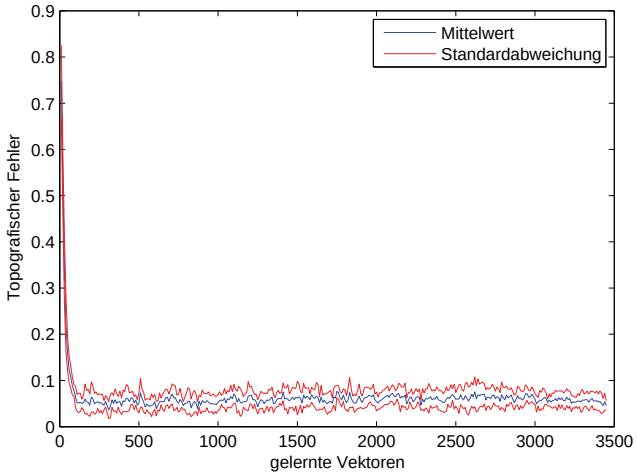
Mit Hilfe des Quantisierungsfehlers lassen sich verschiedene Implementierungen, die mit gleicher Rechenpräzision arbeiten, vergleichen (siehe Abschnitt 1.5). Der Quantisierungsfehler gibt Auskunft über die Güte der Aufteilung der Eingabevektoren $\mathbf{x}_i$ auf die Kartenelemente $\mathbf{m}_j$.

Das erreichbare Minimum für E_Q hängt von der Anzahl der Referenzvektoren l (im k-Means Jargon auch Centroiden genannt), der Zahldarstellung und dem Datensatz ab. Um $\min(E_Q)$ zu bestimmen, ist es nötig, die optimale Partitionierung der Punktmenge X in l Gebiete zu bestimmen. Diese Aufgabe entspricht dem k-Means Problem, das in der Regel sehr gut durch Lloyds Algorithmus [87] angenähert wird. In [81] zeigen die Autoren einen genetischen k-Means Algorithmus, der zuverlässig gegen das Minimum konvergiert.

In Abbildung 1.5a ist ein typischer Verlauf eines Lernvorganges dargestellt, wobei über zehn Durchläufe gemittelt wurde. Dabei ist deutlich zu erkennen, dass der minimale Fehler der SOM noch erheblich (etwa Faktor zwei) über dem



(a) Quantisierungsfehler



(b) Topografischer Fehler

Abbildung 1.5: Typischer Verlauf von Quantisierungs- und topografischem Fehler

erreichbaren Fehler liegt. Dies hängt mit der topografischen Ordnung der Kartenelemente und mit dem Vergrößerungsexponenten ρ der SOM zusammen, beide werden in den folgenden Abschnitten erläutert.

In der Praxis eignet sich der Quantisierungsfehler sehr gut dazu, den Verlauf des Lernvorgangs zu beobachten, und die Auswirkungen von Parametervariationen können beurteilt werden. Als alleiniges Maß für die Qualität einer Karte ist der Quantisierungsfehler aber ungeeignet, da immer nur zwei Lernvorgänge verglichen werden. Eine absolute Aussage ist nicht möglich, da der minimale Quantisierungsfehler in der Regel nicht bekannt ist. Darüber hinaus beschreibt dieses

Fehlermaß nur einen Aspekt des Lernvorganges, z.B. die Topografieerhaltung, die der Quantisierungseigenschaft entgegenwirkt, wird hier nicht berücksichtigt.

Topografischer Fehler

Eine besondere Eigenschaft des SOM-Algorithmus ist die Topografieerhaltung. Während andere Vektor-Quantisierer wie z.B. Lloyds Algorithmus eine optimale Partitionierung des Datensatzes finden können, gehen topografische Informationen, also die räumliche Anordnung der Vektoren zueinander, völlig verloren. Die Nachbarschaftsfunktion im SOM-Algorithmus führt hingegen dazu, dass diese Informationen in die Abbildung einfließen; benachbarte Datenpunkte im Eingabedatensatz finden sich in benachbarten Kartenelementen wieder. Um die Güte der Topografieerhaltung beurteilen zu können, wurden verschiedene Metriken definiert, u. A. das Topographische Produkt [26] und die Topografische Funktion [144]. Da die Topografische Funktion intuitiv nachvollziehbare und (im Gegensatz zum Topografischen Produkt) in allen möglichen Arbeitsbereichen der SOM korrekte Ergebnisse liefert, wird im Folgenden diese Metrik beschrieben[1]: Zunächst muss die Topologieerhaltung allgemein definiert werden. Da für eine Karte M_X sowohl die Abbildung $\Psi_{X\to M}$ als auch die inverse Abbildung $\Psi_{M\to X}$ definiert sind, müssen beide in das Maß einfließen. Dazu wird zunächst die Nachbarschaft zwischen zwei Referenzvektoren $\mathbf{m}_i, \mathbf{m}_j$ definiert: Zwei Referenzvektoren $\mathbf{m}_i, \mathbf{m}_j$ sind genau dann adjazent, wenn ihre Rezeptiven Felder R_i, R_j in M, definiert durch die maskierten Voronoi Gebiete $R_i = \tilde{V}_i$ und $R_j = \tilde{V}_j$ mit

$$\tilde{V}_i = \{\mathbf{v}|\ \|\mathbf{v} - \mathbf{m}_i\| \leq \|\mathbf{v} - \mathbf{m}_j\|\ \forall \mathbf{m}_j \in M\} \tag{1.9}$$

adjazent sind. Auf diese Weise kann die Topografieerhaltung, unter Berücksichtigung besonderer Eigenschaften der Delaunay-Triangulation in rechteckigen Neuronen-Anordnungen [144] definiert werden: Eine Karte $M_X(\Psi_{X\to M}, \Psi_{M\to X})$ ist genau dann topografieerhaltend, wenn sowohl die Abbildung $\Psi_{X\to M}$ von X nach M als auch die inverse Abbildung $\Psi_{M\to X}$ von M nach X topografieerhaltend sind.

1. Die Abbildung $\Psi_{X\to M}$ ist topografieerhaltend genau dann, wenn Vektoren $\mathbf{x}_i$ und $\mathbf{x}_j$ aus adjazenten Gebieten in X auch in M nach der Maximum-Norm $\|\cdot\|_{max}$ adjazent sind.

2. Die Abbildung $\Psi_{M\to X}$ ist topografieerhaltend genau dann, wenn nach der euklidischen Norm oder der Summations-Norm adjazente Gebiete in M auch auf benachbarte Neuronen abgebildet werden.

Auf dieser Basis entwickelt Villmann eine Metrik zur Bewertung der Topografieerhaltung, die Topografische Funktion Φ_M^X:

[1] Die Nomenklatur wurde dieser Arbeit angepasst

Die sog. induzierte Delaunay-Triangulation[91] $\mathcal{D}_M$ über die $\mathbf{m}_i$ definiert einen Graphen, der genau die Punkte in M verbindet, die adjazente Voronoi-Polygone haben. Darüber lässt sich ein Distanzmaß $\|\cdot\|_{\mathcal{D}_M}$ für die $\mathbf{m}_i$ definieren:

$$d_{\mathcal{D}_M}(i,j) = \|\mathbf{m}_i - \mathbf{m}_j\|_{\mathcal{D}_M} \tag{1.10}$$

Dabei entspricht $d_{\mathcal{D}_M}(i,j)$ der kleinsten Distanz zwischen $\mathbf{m}_i$ und $\mathbf{m}_j$ im Graphen $\mathcal{D}_M$. Aufbauend auf das Distanzmaß zwischen den Gewichtsvektoren wird für jedes Neuron n_i definiert:

$$f_{T,i}(\kappa) = |\{j|\ \|i-j\|_{max} > \kappa\ ;\ d_{\mathcal{D}_M}(i,j) = 1\}| \tag{1.11}$$

$$f_{T,i}(-\kappa) = |\{j|\ \|i-j\|_2 = 1\ ;\ d_{\mathcal{D}_M}(i,j) > \kappa\}| \tag{1.12}$$

mit $\kappa = 1, ..., l$. Die Funktion $f_i(\kappa)$ bewertet die Topografieerhaltung von $\Psi_{M\to X}$, während $f_i(-\kappa)$ entsprechend die Topografieerhaltung $\Psi_{X\to M}$ bewertet. Daraus wird die topografische Funktion definiert:

$$\Phi_M^X(\kappa) = \begin{cases} \frac{1}{N}\sum_{j\in M} f_{T,i}(\kappa) & \kappa > 0 \\ \Phi_M^X(1) + \Phi_M^X(-1) & \kappa = 0 \\ \frac{1}{N}\sum_{j\in M} f_{T,i}(\kappa) & \kappa < 0 \end{cases} \tag{1.13}$$

Eine Karte mit $\Phi_M^X = 0$ ist perfekt topografieerhaltend. Das größte κ^+ mit $\Phi_M^X(\kappa^+) \neq 0$ beschreibt die *Länge* der größten Faltung der X in M, wenn die effektive Dimension der X größer ist als die der M. Das kleinste κ^- mit $\Phi_M^X(\kappa^-) \neq 0$ beschreibt die Größe der Verwerfungen innerhalb der Karte, wenn die effektive Dimension der X kleiner ist als die der M. In der Praxis ist es nicht notwendig die vollständige Delaunay-Triangulation $\mathcal{D}_M$ zu berechnen, stattdessen werden zu jedem Vektor $x_i \in X$ die Neuronen mit dem kleinsten und dem zweitkleinsten Abstand zu x_i bestimmt, anhand derer die Gleichungen 1.11, 1.12 und 1.13 ausgewertet werden können.

Eine Variation zur Topografischen Funktion ergibt sich, wenn diejenigen Kartenelemente, denen kein Datenpunkt aus X zugeordnet ist, nicht in die Berechnung der topographischen Funktion mit einbezogen werden. Diese *toten* Elemente stellen gewöhnlich Lücken oder Grenzen zwischen den Clustern in der Karte dar und helfen somit, die topografische Abbildung zu verbessern. Da tote Elemente Gewichte haben, die zwischen angrenzenden Gebieten liegen, und damit typischerweise keinem Datenpunkt in X sinnvoll zugeordnet werden können, werden die Ergebnisse der Topografischen Funktion verfälscht.

Zusätzlich zur topografischen Funktion Φ_M^X, die besonders zur Beurteilung des Lernergebnisses geeignet ist, ergibt sich die Definition des topografischen Fehlers E_T als Spezialfall von Φ_M^X:

$$E_T = \Phi_M^X(1) \tag{1.14}$$

Der topografische Fehler beschreibt den Anteil der Vektoren aus X, deren Abbildung einen topografischen Fehler aufweisen. Dieses skalare Fehlermaß eignet sich besonders, um den Verlauf des Lernprozesses zu beurteilen. Ein Beispiel für einen typischen Verlauf des topografischen Fehlers zeigt Abbildung 1.5b. Es wird ersichtlich, dass die topografische Ordnung vor allem am Anfang des Lernvorganges erfolgt, und dann nur noch wenig verändert wird. Im gezeigten Beispiel fällt der topografische Fehler bereits nach der Verarbeitung von einem Viertel der Eingabedaten in der ersten Epoche (Eine Epoche bezeichnet $|X| = k$ Lernschritte) unter 10% und bleibt dann relativ stabil.

Differentielle Topografische Funktion

Bei der Differentiellen Topografischen Funktion [160] (DTF) handelt es sich um eine Variante der Topografischen Funktion, in der die Stärke der Faltung mit einer Länge κ besser sichtbar wird:

$$\mathrm{DTF}_M^X(\kappa) = \begin{cases} \Phi(\kappa - 1) - \Phi(\kappa) & \kappa \geq 2 \\ \Phi(\kappa + 1) - \Phi(\kappa) & \kappa \leq -2 \end{cases} \tag{1.15}$$

Gewichtete Differentielle Topografische Funktion

Aufbauend auf die DTF wird in der Gewichteten DTF [160] (engl. Weighted Differential Topographic Function, WDTF) die *Stärke* der topografischen Verletzungen mit einbezogen: Die Verbindungsstärke wird definiert als $CONN(i, j)$ [129]. Die Werte dieser $l \times l$ Matrix sind definiert durch die Anzahl der Datenpunkte, deren nächstes (Erregungszentrum) und zweitnächstes Neuron i und j bzw. j und i entsprechen. Dann wird für jedes Neuron i definiert:

$$h_i(\kappa) = \sum_{\|i-j\|_{max}=\kappa, d_{D_M}(i,j)=1} CONN(i, j), \kappa = 2, ..., max_{i,j \in A \| i-j \|_{max}} \tag{1.16}$$

Daraus ergibt sich die Definition der WDTF zu

$$\mathrm{WDTF}(\kappa) = \frac{1}{2k} \sum_{i \in A} h_i(\kappa) \tag{1.17}$$

Klassifizierungsfehler

Der *Klassifizierungsfehler* E_K wird in der Evaluationsphase bei Datensätzen mit bekannter Partitionierung, also bekannten Clustern, berechnet. Er gibt das Verhältnis zwischen inkorrekt klassifizierten Vektoren und der Gesamtanzahl der präsentierten Vektoren aus einem Testdatensatz an. Um den Klassifizierungsfehler experimentell zu bestimmen, wird ein Testdatensatz benötigt, bei dem die Zugehörigkeit aller Vektoren zu einer Klasse bekannt ist[2]. Dieser Datensatz wird in

[2] Dies gilt nicht für vergleichende Analysen, siehe Kapitel 2

einen Lern- und einen Evaluierungsdatensatz aufgeteilt. Im ersten Schritt wird der SOM der Lerndatensatz gemäß Gleichung 1.1 und 1.2 präsentiert. Im Anschluss daran muss eine Zuordnung von Kartengebieten zu den Clustern im Eingaberaum gefunden werden. Dies kann manuell unter Zuhilfenahme grafischer Darstellungen (z.B. U-Matrix, Abbildung 1.3b), oder aber automatisiert z.B. mit Hilfe eines K-Means Algorithmus geschehen. Anschließend werden die Vektoren aus dem Evaluationsdatensatz mit Hilfe der Karte klassifiziert, und das Ergebnis mit der Kennzeichnung aus dem ursprünglichen Datensatz verglichen.

Entropie

Die Entropie, als Maß für den Informationsgehalt der Karte, ist ein weiterer Indikator für die Vergleichbarkeit verschiedener Karten. Die Entropie H ist definiert durch

$$H = \sum_{j=1}^{l} P(j) \log_2 P(j) \tag{1.18}$$

Wobei $P(j)$ die Wahrscheinlichkeit bezeichnet, mit der ein Datenvektor auf das Neuron j abgebildet wird. $P(j)$ lässt sich durch

$$P(j) = \frac{|RF_j|}{k} \tag{1.19}$$

annähern, wobei $|RF_j|$ die Größe des Rezeptiven Feldes des Neurons j, also die Anzahl der auf das Neuron j abgebildeten Datenvektoren und k die Anzahl der Datenvektoren bezeichnet. Die Entropie ist nur abhängig vom Datensatz und vom Vergrößerungsexponenten ρ (Gleichung 1.24), eine Variation zwischen verschiedenen Implementierungen zeigt damit die Unterschiedlichkeit des Ergebnisses.

1.3.2 Leistungsmaße

Neben der Qualität des Anlernvorganges ist auch die Verarbeitungszeit von großer Bedeutung für die Bewertung einer Implementierung, insbesondere dann, wenn Echtzeit-Kriterien einzuhalten sind. Im Folgenden werden verschiedene Maße zur Bewertung der Lernleistung definiert und auf ihre Tauglichkeit hinsichtlich des Vergleichs verschiedener Implementierungen untersucht.

Lernleistung und Abfrageleistung

Um verschiedene Implementierungen von SOM und anderen neuronalen Netzen vergleichen zu können, wurden die Lernleistung $P_{C,l}$ (engl. learn) und die Abfrageleistung $P_{C,r}$ (engl. recall) definiert. Diese sind immer im Zusammenhang mit dem Netzmodell zu sehen, daher sind Vergleiche zwischen unterschiedlichen Netzwerktypen anhand dieser Maße wenig sinnvoll. Die Lernleistung $P_{C,l}$ gibt an, wie viele synaptische Verbindungen pro Sekunde an den Lerndatensatz angepasst

werden können. Sie wird in der Einheit CUPS (engl. Connection Updates Per Second, Verbindungsanpassungen pro Sekunde) angegeben. Dementsprechend gibt die Abfrageleistung $P_{C,r}$ an, wie viele Vektoren pro Sekunde ihrem Erregungszentrum zugeordnet werden können. Die Einheit der Abfrageleistung ist CPS (engl. Connections Per Second, Verbindungen pro Sekunde). In Abhängigkeit von der Dimension der Gewichtsvektoren d_m und der Anzahl der Neuronen l bzw. der Neuronen, die in einem Lernschritt adaptiert werden l_A, ergibt sich P zu

$$P_{C,r} := \frac{d_m \cdot l}{L_r} \tag{1.20}$$

$$P_{C,l} := \frac{d_m \cdot l_A}{L_l} \tag{1.21}$$

wobei L_r und L_l die Zeiten zur Bestimmung des Erregungszentrums bzw. des Anlernens eines Vektors bezeichnen.

Ein in diesen Maßen nicht berücksichtigter Parameter liegt in der Rechengenauigkeit verschiedener Implementierungen. Diese hat insbesondere bei der Betrachtung von ASIC-basierten Implementierungen einen Einfluss auf die Berechnungsgeschwindigkeit, ausgedrückt durch die Latenz L und die maximale Taktfrequenz F. Zusätzlich wirkt sich die Rechenpräzision aber auch auf den berechneten Algorithmus aus, die Vergleichbarkeit solcher Systeme mittels der oben angegebenen Maße ist also nur sehr eingeschränkt gegeben. Ein Vorschlag, die Rechenpräzision in Form der Wortbreite der Eingabedaten w_x und der Wortbreite der Gewichtsvektoren w_m mit einzubeziehen wurde von E. van Keulen et al. [136] in Form der abgewandelten Maße für die Lernleistung $P_{Cb,l}$ und die Abfrageleistung $P_{Cb,l}$ diskutiert. Mit den Einheiten CPPS (Connection Primitives Per Second) und CUPPS (Connection Update Primitives Per Second) bezieht er die Wortbreite mit ein:

$$P_{Cb,r} := w_x \cdot w_m \cdot P_{C,r} \tag{1.22}$$

$$P_{Cb,l} := w_x \cdot w_m \cdot P_{C,l} \tag{1.23}$$

w_m und w_x sind dabei als Wortbreiten einer Festkommadarstellung von Eingabedaten $\mathbf{x}$ und Referenzvektoren $\mathbf{m}$ zu verstehen. Auch dieser Ansatz spiegelt die Realität nur sehr verzerrt wider, da hier im Prinzip eine quadratische Abhängigkeit zwischen Wortbreite und Lernleistung unterstellt wird. Eine ausgiebige Untersuchung dieses Zusammenhangs wird in Kapitel 2 vorgenommen.

1.4 Theoretische Analyseergebnisse

In den nachfolgenden Abschnitten werden einige wichtige Erkenntnisse aus theoretischen Betrachtungen des SOM-Algorithmus zusammengestellt, die große praktische Relevanz haben. Trotz der langen Zeit seit der Präsentation des Algorithmus konnte noch kein allgemeiner Beweis für die Konvergenz der SOM gefunden

werden, daher beziehen sich die Ergebnisse zumeist nur auf Spezialfälle, z.B. die Simulation einer eindimensionalen SOM. In der praktischen Anwendung lassen sich die Ergebnisse oft auch auf allgemeinere Fälle ausdehnen, zahlreiche Veröffentlichungen zeigen dies. In [53] werden aktuelle analytische Ergebnisse und deren Geltungsbereich zusammengefasst.

1.4.1 Vergrößerungsexponent

Der Begriff *Vergrößerungsexponent* bezeichnet im Zusammenhang mit Vektor-Quantisierern das Verhältnis zwischen der statistischen Häufigkeit des Auftretens eines *Eingangsmusters* und der Größe des Karten-Gebietes, das dieses Muster repräsentiert [77]. Dieses Verhältnis wird durch

$$p(M) \propto [p(X)]^{\rho}, \tag{1.24}$$

beschrieben, wobei $p(X)$ die Verteilung der Punkte (WDF) im Eingaberaum X und $p(M)$ die Verteilung der Punkte im Gewichtsraum M bezeichnet. Der Vergrößerungsexponent ρ ist abhängig von der Nachbarschaftsfunktion, und ergibt sich im Falle der eindimensionalen Kohonenkarte [113] zu

$$\rho = \frac{2}{3} - \frac{1}{3r_h^2 + 3(r_h+1)^2}, \tag{1.25}$$

wobei r_h die Anzahl der Nachbarn auf jeder Seite des Erregungszentrums angibt, die in die Adaption einbezogen werden. In mehrdimensionalen Karten gilt für separierbare Datensätze $\rho = \frac{2}{3}$ [27]. Aus Sicht der Anwendung wird dadurch die Abbildung verzerrt, so dass häufig auftretende Eingabemuster in der Karte bevorzugt werden, während seltene Muster unterrepräsentiert sind bzw. verschwinden. Andere Werte für ρ können die Eigenschaften der Abbildung hinsichtlich ihrer Optimierungseigenschaften verändern; während $\rho = 1$ den Informationsgehalt der Karte optimiert, wird der mittlere Quantisierungsfehler durch $\rho = 1/3$ minimiert. Ein praktisch sehr relevanter Fall tritt für $\rho < 0$ ein, denn hier werden selten auftretende Muster hervorgehoben; dieser Effekt wird auch *perceptual magnet* genannt. Um diese Veränderung von ρ zu erreichen, wurden verschiedene Ergänzungen des Algorithmus vorgeschlagen, die in Abschnitt 1.5.1 vorgestellt werden.

1.4.2 Konvergenz

Hinsichtlich der Konvergenz des SOM-Algorithmus, also des Erreichens eines geordneten Zustandes der nicht mehr verlassen wird, liegen keine allgemeinen Ergebnisse vor. Im Folgenden werden die vorliegenden Teilergebnisse beschrieben. Die grundlegende Beweisidee ist, die Gewichtsvektoren als Markov-Prozesse aufzufassen, die mit einer Wahrscheinlichkeit größer Null und in endlicher Zeit in

einen geordneten Zustand übergehen. Im eindimensionalen Fall ist dies möglich, da ein geordneter Zustand existiert und absorbierend ist. Für den mehrdimensionalen Fall kommt diese Vorgehensweise nicht in Frage, da die Gewichte nicht als reduzierbare Markov Prozesse aufgefasst werden können und zusätzlich keine klare Definition für einen geordneten Zustand existiert.

Für den eindimensionalen Fall konnte Sadeghi [115] nachweisen, dass für eine beliebige monoton fallende Nachbarschaftsfunktion, die mindestens einen Nachbarn des Erregungszentrums erfasst, ein organisierter Zustand erreicht wird. Voraussetzung des Beweises ist allerdings eine Lebesgue-kontinuierliche Verteilung der Eingangs-Vektoren, was für den praktischen Fall im Allgemeinen nicht zutrifft. Darüber hinaus hat Flanagan [52] dies für eine streng monoton fallende Nachbarschaftsfunktion sowohl für Lebesgue-kontinuierliche als auch nicht-Lebesgue-kontinuierliche Verteilungen bewiesen. Im nicht-Lebesgue-kontinuierlichen Fall ist die Fähigkeit zur Selbstorganisation abhängig von der Anzahl der Eingabedaten und der Weite der Nachbarschaftsfunktion. Auf empirischer Basis konnte für k Eingabevektoren und l Neuronen bei einer Nachbarschaftsfunktion mit einem effektiven Radius r_h (das größte r für das gilt $h_{c,i}(r) \geq 0$) die folgende Bedingung als hinreichend identifiziert werden:

$$\frac{l}{r_h} \leq \log_2(k) \tag{1.26}$$

Das weitreichendste Ergebnis für den multidimensionalen Fall stammt ebenfalls von Sadeghi, der in seinen Beweisen von einer konstanten Lernrate ausgeht, d.h. die Nachbarschaftsfunktion $h_{c,i}$ bleibt über t konstant. In [116] zeigt Sadeghi, dass der Algorithmus mit konstanter Lernrate von einem beliebigen Startpunkt aus schwach konvergiert. Dabei wird das Maß für die Konvergenz als endlich und unveränderlich angenommen, woraus kein konkreter Anhaltspunkt für die Art der Ordnung gewonnen werden kann.

1.4.3 Topografieerhaltung

Die Topografieerhaltung (siehe Abschnitt 1.3.1) ist durch die Ergebnisse von Villmann et al. [144] exakt definiert und messbar. Trotzdem konnte bisher nicht nachgewiesen werden, dass ein topografieerhaltender Zustand erreicht wird. Das liegt insbesondere daran, dass für die Kohonen-Karten keine Potentialfunktion bekannt ist, deren Minima geordnete Zustände darstellen. Anschaulich lässt sich dies über die Natur des geordneten Zustandes im n-Dimensionalen erläutern: während im eindimensionalen der geordnete Zustand unmittelbar ersichtlich ist, fehlt für den mehrdimensionalen Fall die Definition.

1.4.4 Einfluss beschränkter Präzision

Die analytischen Ergebnisse zum Einfluss beschränkter Berechnungspräzision auf den Lernvorgang der SOM sind nur für den eindimensionalen Fall hinreichend geklärt. In [132] zeigt Thiran für den eindimensionalen Fall, dass die diskretisierte SOM unter bestimmten Randbedingungen sicher (Wahrscheinlichkeit eins) in einen geordneten Zustand übergeht. Dazu wird die Quantisierung (Diskretisierung) als nichtlineare Funktion $D(\cdot)$ mit

$$D(x) = iq \text{ mit } (i - \frac{1}{2})q \leq x < (i + \frac{1}{2})q \tag{1.27}$$

$$q = \frac{X_{max} - X_{min}}{2^w - 1} \tag{1.28}$$

mit der Anzahl der Binärstellen w dargestellt und die Auswirkungen auf die Gewichte, repräsentiert durch Markovketten, analysiert. Der Beweis hält für die folgenden Bedingungen, die zusätzlich zu den in [41] getroffenen Annahmen gelten:

- **Auswahl des Erregungszentrums:** Im diskreten kann der Fall auftreten, dass zwei Neuronen mit gleichem Abstand zum Eingabevektor existieren[3]. Es muss eine Regelung gefunden werden, nach der das Erregungszentrum gefunden wird.

- **Nachbarschaftsfunktion:** In [41] wurde gezeigt, dass eine räumlich fallende Nachbarschaftsfunktion die Konvergenzgeschwindigkeit erhöht, für den diskreten Fall ist dies nun zwingend notwendig. Der Grund liegt in der Quantisierung $D(\cdot)$, die in bestimmten Fällen dafür sorgt, dass das Erregungszentrum nicht angepasst wird (Rundungsfehler), während die Nachbarn des Erregungszentrums, die einen größeren Abstand zum Eingabevektor haben, noch adaptiert werden können.

  Zur analytischen Beschreibung wird der Parameter $\delta(j)$ eingeführt, der den maximalen Abstand zwischen Erregungszentrum und zu adaptierenden Neuron beschreibt, bei dem das Gewicht aufgrund der Quantisierung nicht verändert wird. Für diesen muss gelten

  $$\delta(j) < \delta(j+1) \text{ wenn } \delta(j+1) \leq \frac{1}{2^w - 1} - 2 \tag{1.29}$$

  damit die Ordnung der Vektoren erfolgen kann.

Des Weiteren fordert Thiran, dass die Anzahl der möglichen Werte der Referenzvektoren größer sein soll als die Anzahl der Neuronen, was für praxisrelevante Konfiguration immer der Fall sein dürfte. Für den multidimensionalen Fall sind bislang keine theoretischen Ergebnisse bekannt, einige empirisch gewonnene Daten werden in Abschnitt 2 vorgestellt.

[3] Im Allgemeinen kann dieser Fall auch für die kontinuierliche Version der eindimensionalen SOM auftreten, allerdings nicht unter den speziellen Bedingungen, die dem Beweis von Cottrell et. al. zugrunde liegen

1.5 Varianten des SOM-Algorithmus

In der Literatur ist eine Vielzahl von Varianten des SOM-Algorithmus zu finden. Diese Varianten optimieren die Eigenschaften der SOM zumeist hinsichtlich einer bestimmten Anwendung, einige Anpassungen sind aber auch fundamentaler Natur. Im Folgenden werden kurz einige fundamentale Erweiterungen vorgestellt.

1.5.1 Beeinflussung des Vergrößerungsexponents

DeSieno beschreibt in [46] einen abgewandelten Algorithmus zur SOM (engl. Conscienc-SOM), bei dem für den Vergrößerungsexponenten $\rho = 1$ gilt. Dazu wird die Suche nach dem Erregungszentrum modifiziert:

$$||\mathbf{x} - \mathbf{m}_c|| = \min_i\{||\mathbf{x} - \mathbf{m}_i||_p - g_i\} \tag{1.30}$$

Das g_i stellt ein *Gewissen* des Neurons in Form eines Speichers dar. Je häufiger das Neuron i Erregungszentrum war, desto höher wird g_i und umso geringer wird die Wahrscheinlichkeit, das dasselbe Neuron wieder Erregungszentrum wird. Die Berechnung der g_i erfolgt nach folgenden Gleichungen:

$$g_i = \gamma(t)(1/l - F_{C,i}(t+1)) \tag{1.31}$$

$$F_{C,i}(t+1) = F_{C,i}(t) + \beta(t)(y_i - F_{C,i}(t)), \tag{1.32}$$

wobei $y_i = 1$ für das Erregungszentrum und $y_i = 0$ für alle anderen Neuronen gilt. Die Faktoren $0 < \beta(t) < 1$ und $0 < \gamma(t)$ sind wählbar und müssen im Verlauf des Lernvorganges gegen Null gehen. In DeSienos Version des Algorithmus wird nur das Erregungszentrum adaptiert, wodurch die Eigenschaft der topografischen Ordnung verloren geht. Um diese Eigenschaft beizubehalten, wird in [94] die Nachbarschaftsfunktion wieder eingeführt:

$$h(c,i) = \begin{cases} 1 & \text{falls } ||r_c - r_i||_1 \leq 1 \\ 0 & \text{sonst} \end{cases} \tag{1.33}$$

wobei r_c die Position des Erregungszentrums und r_i die Position des Neurons i bezeichnet. Es werden also nur das Erregungszentrum und die direkten Nachbarn adaptiert, alle mit dem gleichen Adaptionsfaktor α. Es gibt keinen Beweis für die Eigenschaft $\rho = 1$ im Mehrdimensionalen, Simulationen [95] zeigen aber, dass auch für $d_m > 1$ $\rho \approx 1$ gilt.

Bauer et al. schlagen in [27] eine weitere Modifikation des SOM-Algorithmus vor (nach [95] im Folgenden als BDH abgekürzt), über die sich der Vergrößerungsexponent ρ wählen lässt. Dazu wird die Adaptionsgleichung 1.2 erweitert:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \epsilon_\rho h_{c,i}(t)[\mathbf{x}(t) - \mathbf{m}_i(t)], \tag{1.34}$$

wobei ϵ_ρ einen adaptiven, knotenabhängigen Adaptionsfaktor darstellt, der nach

$$\langle \epsilon_\rho \rangle = \epsilon_0 p(\mathbf{m}_i)^s \tag{1.35}$$

bestimmt wird. Der Exponent s erlaubt die Beeinflussung des Vergrößerungsexponenten $\rho' = \rho(1+s)$ [27, 45], wobei die unbekannte Dichte der Eingangsverteilung in $p(\mathbf{m}_i)$ unter Verwendung der bisher gelernten Information abgeschätzt wird:

$$\epsilon_\rho(t) = \epsilon_0 \left(\frac{1}{\Delta t_c} \left(\frac{1}{||\mathbf{x}(t) - \mathbf{m}_i(t)||^{d_m}} \right) \right)^s \tag{1.36}$$

Dabei ist Δt_c die Zeit, seit dem das Neuron zum letzten Mal Erregungszentrum war und d_m bezeichnet die Dimension der Eingabevektoren. Über den freien Parameter s wird nun nach

$$cP(w)^\rho = P(w)^{\frac{2}{3}(1+s)} \tag{1.37}$$

der gewünschte Vergrößerungsexponent ρ eingestellt. Da zu große Werte für $\epsilon_\rho(t)$ den Lernvorgang destabilisieren würden, wird eine obere Schranke eingeführt: $\epsilon_\rho(t) \leq \epsilon_{max} = 0,9$. Diese Beschränkung zeigt sich als sehr günstig für eine Implementierung in Hardware, da sich der Wertebereich von $\epsilon\alpha$ damit in demselben Wertebereich befindet, wie die Adaptionswerte α. Eine Vergrößerung der Wortbreite bei der Multiplikation ist also nicht notwendig. Simulationen zeigen, dass sowohl die Steuerung des Vergrößerungsexponenten als auch die damit assoziierten Optimierungseigenschaften im realen Experiment den analytisch gewonnenen Erkenntnissen folgen. In Abbildung 1.6 ist das Lernergebnis eines eindimensionalen Datensatzes mit $p(\nu) = 2\nu$ gezeigt, bei dem der Vergrößerungsexponent mit dem BDH Algorithmus variiert wurde. Die gestrichelten Linien zeigen das erwartete Ergebnis für $\rho = 0$ und $\rho = 1$. Diese werden annähernd erreicht, der Einfluss des BDH Algorithmus ist deutlich zu erkennen.

Entgegen den strikten Einschränkungen für den Gültigkeitsbereich der in [27] vorgestellten Algorithmen zeigen u. A. Merényi et al. in [95], dass im Experiment auch wesentlich allgemeinere Probleme von der *magnification control* profitieren.

Sowohl der Algorithmus nach DeSieno, als auch der Algorithmus nach Bauer et al. werden in den nachfolgenden Kapiteln zur Evaluierung des Einflusses von eingeschränkter Präzision auf die Berechnung von SOM in Hardware herangezogen.

1.5.2 Behandlung zeitbehafteter Daten

Für die Repräsentation zeitbehafteter Daten in einer SOM sind im Wesentlichen drei Ansätze bekannt:

1. Zeitverzögerungen in den Eingängen: Hierbei werden der SOM gleichzeitig der aktuelle Datenvektor $\mathbf{x}(t)$ sowie vergangene Vektoren $\mathbf{x}(t-1)$ usw. präsentiert. [73, 139]

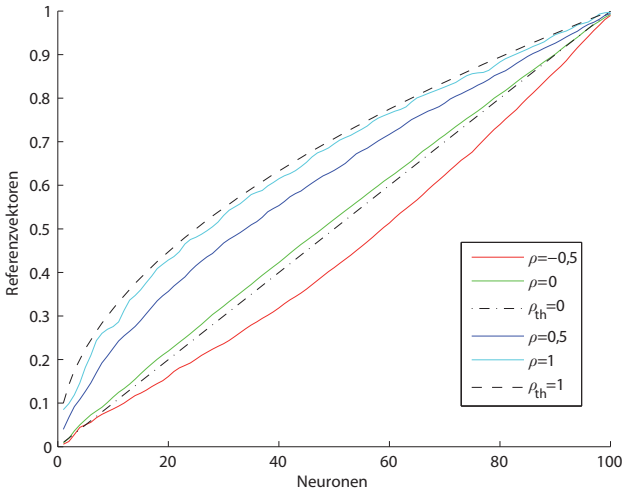


Abbildung 1.6: BDH Algorithmus: Verteilung der Gewichtsvektoren bei einem Datensatz mit $p(\nu) = 2\nu$

2. Rekursion, dabei wird zusätzlich zum Eingabevektor eine Repräsentation des letzten Lernvorganges gelernt, etwa die Koordinaten des letzten Erregungszentrums. Auf diese Weise wird die Trajektorie aufeinanderfolgender Erregungszentren zur Repräsentation der temporalen Daten. [50, 65, 79, 148]

3. Durch den Einsatz sog. *leaky integrator* Neuronen wird bei der Bestimmung des Erregungszentrums zusätzlich zur aktuellen Distanz die Distanz vorangegangener Abstandsberechnungen mit einbezogen. Über einen Verfallsterm in dieser Abstandsberechnung wird der Einfluss vorangegangener Berechnungen begrenzt. [37]

Diese Ansätze, oder Kombinationen daraus, werden vor allem bei der Erkennung natürlicher Sprache verwendet. Zusätzlich gibt es Anwendungen im Bereich der Zeitreihenvorhersage [80].

1.5.3 Dynamische Gitter

Die Wahl geeigneter Parameter für den SOM-Lernvorgang ist unter Umständen, insbesondere bei großen Datensätzen unbekannter Struktur, problematisch. Vor allem die Größe der Karte hat einen wesentlichen Einfluss auf das Lernergebnis, weshalb verschiedene Algorithmen präsentiert wurden, bei der sich die Größe der Karte zur Zeit des Lernvorganges dynamisch ändern kann. Hierzu werden in Abhängigkeit eines lokalen Fehlermaßes Neuronen hinzugefügt, die den lokalen Fehler verringern. Zwei Vertreter dieser Art von selbst-organisierenden neuronalen Netzen sind die wachsenden Gitter [57] und die wachsenden Zellstrukturen [56]. Beide

vereinen die Eigenschaften der SOM (Topografieerhaltung, Vektorquantisierung) auf sich und ermöglichen es, eine Karte ohne a-priori Festlegung der Größe zu erstellen.

1.6 SOM-Implementierungen

Selbst-organisierende Karten wurden seit ihrer Beschreibung durch Teuvo Kohonen auf Basis verschiedenster Technologien implementiert. Die Gründe dafür sind, neben der praktischen Relevanz, vor allem in der einfachen, inhärent parallelen Struktur des Algorithmus zu suchen. Neben den praktischen Einsatzgebieten dient der SOM-Algorithmus oft als Referenz, um verschiedene Systeme zu vergleichen, auch wenn dabei oft Maße verwendet werden, die sich nicht direkt vergleichen lassen (siehe dazu auch Abschnitt 1.3). Im Folgenden werden einige dieser Implementierungen gegliedert nach der zugrundeliegenden Technologie aufgeführt.

1.6.1 Software

Die Anzahl verfügbarer, oft Domänen-spezifischer Softwaresysteme für die Berechnung des SOM-Algorithmus ist aus den oben genannten Gründen unüberschaubar. Daher werden im Folgenden nur drei Vertreter exemplarisch aufgeführt.

Eines der am weitesten verbreiteten Systeme zur SOM-basierten Datenanalyse ist die MATLAB-basierte SOM-Toolbox von Juha Vesanto [141], die am Fachgebiet von Prof. Teuvo Kohonen entwickelt wurde. Es handelt sich um eine komplette Analyseumgebung inklusive vieler Vor- und Nachverarbeitungsschritte sowie zahlreicher Visualisierungsmöglichkeiten. Die SOM-bezogenen Grafiken in der vorliegenden Arbeit wurden ausschließlich auf Basis der o.g. Software erzeugt.

Bei der Implementierung selbst-organisierender Karten werden aufgrund der dem Algorithmus inhärenten Parallelität immer auch massiv parallele Rechnersysteme in Betracht gezogen. In [118] wird ein Überblick über bekannte Implementierungen und deren Vor- und Nachteile gegeben. Eine wichtige Erkenntnis ist, dass die maximal erreichbare Geschwindigkeit nicht alleine von der Prozessoranzahl und deren Leistungsfähigkeit, sondern auch sehr stark vom Verbindungsnetzwerk abhängt. Je enger (hohe Bandbreite und geringe Latenz) die Prozessoren gekoppelt sind, desto besser kann die Parallelität durch mehrere Prozessoren ausgenutzt werden. In den vorliegenden Untersuchungen werden vier verschiedene Rechencluster untersucht; das Optimum der Ausführungszeit für das gegebene Problem liegt hier zwischen zwei und neun Prozessoren. Während Seiffert in seiner Publikation den Standard SOM-Algorithmus verwendet, werden in [83] verschiedene Implementierungsvarianten untersucht, die den Algorithmus selber verändern. Kwiatkowski kommt ebenfalls zu dem Ergebnis, dass die Kommunikation zwischen den Rechenknoten der begrenzende Parameter ist und schlägt daher eine Variante [83] des Batch Algorithmus [78] vor. Mit dieser erzielt er nahezu linea-

re Beschleunigung gegenüber Einzelprozessor Systemen, allerdings zeigt Fort in [54], dass die Qualität der gelernten Karten hinsichtlich der topografischen Ordnung nicht vergleichbar ist. Darüber hinaus gelten alle von Fort beschriebenen Einschränkungen.

Im Gegensatz zu den in [118, 119] präsentierten Ansätzen, die spezialisierte Rechencluster benötigen, hat eine Veränderung in der Architektur von Graphikprozessoren (engl. GPGPUs, General Purpose Graphic Processor Units) eine weitere mögliche Plattform hervorgebracht. Der bis heute anhaltende Trend, viele kleine und einfache Prozessoren zur Bildberechnung zu verwenden, hat verschiedene Autoren in [127, 34, 88] inspiriert, aktuelle GPUs zur SOM-Berechnung zu verwenden.

1.6.2 Analoge und Hybride Hardware

Eine große Herausforderung bei der Implementierung analoger neuronaler Netze ist im Allgemeinen die Speicherung der Gewichts- bzw. Referenzvektoren. In [67] und darauf aufbauend in [63] beschreiben die Autoren Implementierungen, in denen PLLs (engl. Phase Locked Loops) als Gewichtsspeicher verwendet werden. Dabei wird die PLL so modifiziert, dass die Kompensation der Verstimmung von außen kontrolliert werden kann. Der Wert von Gewichten und Daten wird dann in der Phase der Signale kodiert. In [157] wird ein ähnliches System vorgestellt, in dem die Information allerdings in Form von Pulsbreiten kodiert wird.

In [32] wird ein hybrider Ansatz vorgestellt, bei dem die Gewichte über Kondensatoren gespeichert werden. Besonders hervorzuheben ist hier eine Multi-Gate Thyristor Struktur, die sowohl für die Suche nach dem Erregungszentrum als auch zur Kommunikation zwischen den Neuronen verwendet wird. Das System wurde prototypisch mit fünf analogen Neuronen und einer digitalen Kontroll-Logik implementiert.

Die hier vorgestellten analogen Systeme sind in der Regel einfacher aufgebaut und damit wesentlich kleiner als ihre digitalen Pendants. Diese Aussage findet sich auch in [101], wo auch das Rauschverhalten und die damit einhergehende sehr niedrige Präzision analysiert werden.

1.6.3 Digitale Hardware

In [25] beschreiben die Autoren einen SOM-Beschleuniger auf FPGA Basis, der ausschließlich die Manhattan Distanz (p_0-Norm) und Zweierpotenzen bei der Adaption erlaubt. Diese sehr einfache Struktur führt zu einem sehr kleinen und schnellen Aufbau (jedes Neuron belegt 129 Slices auf einem Virtex II FPGA, und läuft mit maximal 140 MHz). Die Autoren weisen darauf hin, dass die eigentliche Konvergenzgeschwindigkeit mit von der verwendeten Rechenpräzision abhängt, die für die vorliegende Arbeit aber nicht angegeben wird. Bei der Suche nach dem

Erregungszentrum wird ein teils paralleler teils sequentieller Ansatz gewählt. Die Neuronen werden zu mehreren Zeilen zusammengefasst, in denen innerhalb einer Zeile parallel über einen Komparatorbaum geprüft wird, während alle Z Zeilen unabhängig voneinander arbeiten. Bei dieser Vorgehensweise werden zusätzlich zu den l Neuronen auch $l-1$ Komparatoren benötigt.

Pena et al. präsentieren in [102] eine vollständig parallele Implementierung der SOM, die sowohl die Manhattan- (p_0) als auch die Unendlich-Norm (p_∞) unterstützen. Besonders hervorzuheben ist hier, dass auch alle Komponenten gleichzeitig verarbeitet werden, allerdings nur maximal zwei, was eine praktische Nutzung des Systems für den größten Teil des Anwendungsspektrums ausschließt. Jedes Neuron belegt bei 8 Bit Präzision 45 Slices auf einem Spartan II FPGA und läuft mit maximal 27 MHz.

Kurdthongmee [82] beschreibt einen SOM-Hardwarebeschleuniger im Bereich der Bildverarbeitung, der zur Erzeugung einer Farbpalette dient. Dabei ist die Größe der Karte auf 8×8 Neuronen bei einer Präzision von 8 Bit mit drei Komponenten eingestellt. Der Autor verwendet einen komponenten- und neuronenparallelen Ansatz, um Bilder der Größe 640×480 Bildpunkte in Echtzeit, d.h. bei 25 Bilder pro Sekunde zu kodieren. Ähnlich stellt Sudha in [123] eine Implementierung im gleichen Anwendungsgebiet vor, allerdings verwendet Sudha als Zieltechnologie einen 0,35 μm CMOS ASIC. Bei diesem Ansatz handelt es sich um eine massiv parallele Implementierung, bei der eine minimale Latenz erreicht wird, gleichzeitig aber auch eine minimale Flexibilität. Der Ansatz lässt sich praktisch nicht in Richtung größerer Implementierungen skalieren und bietet auch nicht die Möglichkeit, die Kartenparameter zu variieren. Im Vergleich mit den hier vorgestellten Implementierungen zeigt sich somit der Mehraufwand, den eine skalierbare und flexible Lösung mit sich bringt.

Im Fachgebiet Schaltungstechnik der Universität Paderborn existieren eine Reihe von Hardwareimplementierungen für selbst-organisierende Karten für die Bereiche Hardwarebeschleunigung und eingebettete Systeme. Rüping [114] und Porrmann [107] beschreiben massiv parallele Beschleunigersysteme mit 8 Bit Auflösung, die keine Multiplizierer verwenden. Porrmann geht in [106] insbesondere auf die Leistungsbewertung für Hardwarebeschleuniger ein, die eine Host Umgebung berücksichtigt. Witkowski [111, 153] entwickelt eine ähnliche SOM-Hardware, die für die Integration in autonome Miniroboter optimiert sind; dabei wird insbesondere die Problematik des Lernens eines sich dynamisch verändernden Datensatzes betrachtet. Aufbauend auf die Implementierung von Porrmann wurde im Rahmen der Studien- und Diplomarbeit des Autors einige Erweiterungen und Vereinfachungen vorgenommen [1, 9, 10, 11].

In Tabelle 1.1 sind die Hardware Implementierungen aus dem Zeitraum zwischen 1999 und 2008 und ihre wichtigsten Eigenschaften aufgelistet. Während die implementierten Größen (l und d_m) stark variieren, sind die verwendeten Wort-

Tabelle 1.1: SOM-Implementierung aus den letzten zehn Jahren. Unter Adaption wird das Verfahren angegeben, mit dem $\alpha(t)h_{c,i}(\mathbf{m}_i - \mathbf{x})$ in der Hardware berechnet wird. Parallelität bezeichnet den Aufbau (neuronen-, oder komponentenparallel). Die Auswirkungen dieser Charakteristika werden in Kapitel 2 näher erläutert. Werte mit einem '*' wurden aus den Angaben in der Publikation errechnet.

Jahr	w_m [Bit]	$\#PE$	d_m max	Norm p	Adapt.	$P_{C,l}$ [MCUPS]	F [MHz]	Ress.	Architektur	Techn. [μm]	Par.	WTA
1999 [103]	1..n	128	k.A.	1	shift	k.A.	k.A.	7,68 mm^2	ASIC	0,25	N	seriell
1999 [125]	8	4×4	3	1	shift	7	11,36	628 Sl.	xc4020e $\times 5$	0,35	N	
2003 [61]	12	16×16	16 v	1	shift	5500	200	21,16 mm^2	ASIC	0,18	N	seriell
2003 [107]	8	2×8	128 v	1	shift	578	45	28,58 mm^2	ASIC	0,8	N	seriell
2004 [11]	16	8×8	1024 v	1,2	mult	1184	50	24384 Sl.	xc2v4000	0,12	N	seriell
2004 [123]	8	$4 \times 4 \times 4$	3	1	mult	1576 *	41	23,52 mm^2	ASIC	0,35	N+C	Baum
2004 [128]	8	8×8	4	1	shift	40 *	33	k. A.	xc2v6000	0,12	N	seriell
2006 [102]	8	5×5	2	1,∞	shift	28,38	27	3009 Sl.	xc2s400	0,15	N+C	Baum
2008 [82]	8	256×3	3	1	mult	1651 *	24,2	37312 Sl.	xc2vp100	0,12	N+C	seriell
2008 [109]	8	16×16	16	1	shift	124	71,43	21961 Sl.	xc2v6000	0,12	N+C	Baum

breiten überwiegend gleich (8 Bit) und die Adaption erfolgt meistens durch Schieben (siehe Kapitel 2. Im folgenden Kapitel wird deutlich werden, dass diese beiden Charakteristika wesentlichen Einfluss auf den Lernvorgang haben.

1.7 Zusammenfassung

Selbst-organisierende Karten nach Teuvo Kohonen sind heute sowohl ein aktives Forschungsgebiet als auch wichtiges Werkzeug im Bereich der Datenanalyse. Im vorliegenden Kapitel wurde der SOM-Algorithmus vorgestellt und es wurden theoretische Erkenntnisse und deren Grenzen zusammengefasst. Einige Anwendungsbeispiele geben Einblick in die Möglichkeiten der SOM im praktischen Einsatz, und zeigen sowohl die Notwendigkeit für effiziente Implementierungen als auch für Modifikationen auf. Daher wurden die für diese Arbeit wichtigsten Modifikationen beschrieben und einige Implementierungen des Algorithmus charakterisiert.

Bei der Aufarbeitung der publizierten SOM-Implementierungen wurden Arbeiten aus den letzten zehn Jahren, speziell aus dem Bereich digitaler Hardware, berücksichtigt, deren Einsatzgebiete spezifisch für eine bestimmte Aufgabe zu sehen sind. Dies steht im Gegensatz zu früheren Arbeiten, die häufig als universelle Plattformen zur schnellen Berechnung von Künstlichen Neuronalen Netzen im Allgemeinen und Selbst-organisierenden Karten im Besonderen dienen. Dieser Trend zur Spezialisierung zeigt einerseits den Bedarf an effizienten Implementierungen, andererseits wird deutlich, dass die SOM immer häufiger aus dem reinen Forschungsumfeld in den praktischen Einsatz übergeht.

Tabelle 1.1, in der einige Implementierungen charakterisiert werden, zeigt deutlich, dass für eine Umsetzung in digitaler Hardware einige Veränderungen am ursprünglichen Algorithmus vorgenommen wurden. Die Auswirkungen dieser Veränderungen wurden bislang nur eingeschränkt analysiert. Daher werden im folgenden Kapitel zunächst die einzelnen Veränderungen beschrieben und dann genauer analysiert.

Kapitel 2

Effekte hardwarespezifischer Anpassungen

Der SOM-Algorithmus kann im weitesten Sinne als rückgekoppeltes System, ähnlich dem Newtonschen Näherungsverfahren oder einem Regler in der Automatisierungstechnik betrachtet werden. Eine Grundvoraussetzung für die Nutzbarkeit solcher Systeme ist die Feststellung der Konvergenz bzw. der Stabilität für alle möglichen Eingaben. Wie in Kapitel 1.2 beschrieben wurde, ist diese Frage für den SOM-Algorithmus (noch) nicht abschließend beantwortet, so dass sich die Bewertung der Konvergenzeigenschaften der SOM für praxisrelevante Konfigurationen nur empirisch zeigen lässt. Während dies für die unmodifizierte Kohonen-SOM anhand vieler praktischer Beispiele gezeigt wurde, sollen die Auswirkungen hardwarespezifischer Anpassungen im Folgenden untersucht werden.

Die in Abschnitt 1.2.2 vorgestellten Anwendungsgebiete stellen eine repräsentative Auswahl dar und zeigen deutlich, dass das Haupteinsatzgebiet der SOM die Klassifizierung von Daten ist. Das Klassifizieren von Daten beinhaltet hier sowohl die Strukturierung einer ungelernten Karte auf Basis eines Datensatzes (Training) als auch das Klassifizieren von Daten auf Basis einer gelernten Karte (Abruf). Zur Beurteilung beider Vorgänge stehen die in Abschnitt 1.3.1 vorgestellten Kriterien zur Verfügung, von denen nicht alle von praktischer Bedeutung sind. Im Folgenden werden daher zunächst die hardwarespezifischen Anpassungen definiert, um dann die Qualitätskriterien auf ihre Aussagekraft hinsichtlich dieser Anpassungen zu überprüfen. Im Anschluss daran erfolgt auf Basis der ausgewählten Kriterien eine empirische Untersuchung der beschriebenen Anpassungen.

2.1 Hardwarespezifische Anpassungen

Für die Berechnung des SOM-Algorithmus kommen aus dem gesamten Spektrum der Hardwarearchitekturen von den ein- oder mehrkernigen Prozessoren bis hin zu spezialisierten massiv-parallelen Prozessorfeldern im Prinzip alle Architekturen in Frage. Da mit einer Spezialisierung auf die zu berechnende Aufgabe und gleichzeitig mit der Parallelisierung gemeinhin ein besseres Verhältnis zwischen Rechenleistung und Energieaufnahme bzw. Rechenleistung und Flächenbedarf erwartet wird, werden im Folgenden hardwarespezifische Anpassungen diskutiert, die für eine massiv-parallele Implementierung der SOM-Berechnung in Frage kommen. Dabei werden Implementierungsdetails, die sich nicht auf das Verhalten des Algorithmus auswirken, hier nicht beschrieben, sondern nur in Kapitel 3 anhand einer konkreten Implementierung gezeigt. Stattdessen werden hier funktionale Anpassungen des Algorithmus diskutiert, die auf das verwendete Zahlenformat und auf die Abstraktion der Multiplikationen zurück gehen. Weitere Modifikationen, wie etwa das Batch-Update Verfahren werden hier nicht untersucht, da sie sehr schlechte Ergebnisse bezüglich der topografischen Ordnung hervorbringen und damit für den praktischen Einsatz ungeeignet sind [54].

2.1.1 Zahldarstellung

Für die Repräsentation der Referenz- und Datenvektoren sowie der Adaptionswerte kommen sowohl Festkomma- als auch Gleitkomma-Datentypen in Frage. Im Rahmen von Simulationen, die auf Allzweck-Prozessoren durchgeführt werden, kommen in der Regel Gleitkomma-Formate zum Einsatz. Bei diesen können große Wertebereiche mit vergleichsweise wenigen Speicherstellen abgedeckt werden, wobei die Genauigkeit (repräsentiert durch den kleinstmöglichen Abstand zweier darstellbarer Werte) von dem darzustellenden Wert abhängt. Bei einer Festkommadarstellung dagegen ist der Wertebereich bei der gleichen Anzahl Speicherstellen vergleichsweise klein, aber die Genauigkeit ist über den gesamten Wertebereich konstant. Um diese Eigenschaften für den praktischen Fall beurteilen zu können, müssen die Vorverarbeitungsschritte berücksichtigt werden, die bei der Berechnung einer SOM in der Regel vorgenommen werden: die Komponenten der Datenvektoren liegen als Rohdaten (etwa Messwerte) innerhalb eines jeweils eigenen Intervalls vor. Eine direkte Verwendung dieser Rohdaten in der SOM käme einer willkürlichen Gewichtung einzelner Komponenten gleich, die in der Regel nicht wünschenswert ist. Daher werden Datensätze vor dem Lernvorgang komponentenweise normalisiert, gewöhnlich in das Intervall $[0..1]$ oder $[-1..1]$.

Bei einer Speicherung von Zahlen x, y in einem Gleitkommaformat

$$x = z_{s,x} \cdot z_{m,x} \cdot z_{b,x}^{z_{e,x}} \tag{2.1}$$

$$y = z_{s,y} \cdot z_{m,y} \cdot z_{b,y}^{z_{e,y}} \tag{2.2}$$

wobei z_s das Vorzeichen, z_m die Mantisse, z_b die Basis und z_e den Exponenten bezeichnen, werden aufgrund der Normalisierung (vgl. [40]) Zahlen mit $z_{e,x} \neq z_{e,y}$ u.U. mit unterschiedlicher Genauigkeit dargestellt. In einer binären Darstellung ($z_b = 2$) mit w_g Bits für die Mantisse gilt für den kleinstmöglichen Abstand dist_{min} zwischen zwei darstellbaren Zahlen $\text{dist}_{min,gleit} = 2^{z_e}$ mit $z_e \in [-2^{w_g-1}, -2^{w_g-1}+1, ..., 2^{w_g-1}-1]$. Für Festkommadarstellung im Wertebereich [0..1] mit w_{fest} Bits gilt entsprechend $\text{dist}_{min,fest} = z_b^{-w_{fest}}$. Im Vergleich zwischen Festkomma und Gleitkommadarstellung existiert also ein Wert z_q, für den gilt

$$\text{dist}_{min,gleit} \leq z_q \leq \text{dist}_{min,fest} \tag{2.3}$$

Für Werte $x > z_q$ ist die Genauigkeit der Gleitkommdarstellung also geringer als die Genauigkeit der Festkommadarstellung und umgekehrt. Beim Vergleich einer Gleitkommazahl mit einfacher Präzision nach IEEE754 (32 Bit) und einer 32 Bit Festkommazahl ist $z_q = 2^{-32} \approx 0,233 \cdot 10^{-9}$. Es muss experimentell überprüft werden, ob eine variable Genauigkeit Vorteile gegenüber einer festen Genauigkeit hat. Neben der Speicherung der Daten ist auch die eigentliche Berechnung des SOM-Algorithmus zu betrachten: Sowohl bei der Distanzberechnung (Gleichung 1.1) als auch bei der Adaption (Gleichung 1.2) tritt der oben beschriebene Effekt auf, und muss experimentell überprüft werden.

Für eine Hardwareimplementierung ist die Wahl der Zahldarstellung von entscheidender Bedeutung, da sich der Ressourcenbedarf der entsprechenden Einheiten stark unterscheidet: die für die Berechnung der SOM benötigten Addierer/Subtrahierer Bausteine sind bei gleicher Wortbreite für Gleitkommawerte um ein Vielfaches größer als die entsprechenden Festkomma-Komponenten. Die Untersuchungen in diesem Kapitel müssen zeigen, ob dieser Mehraufwand für eine Hardwareimplementierung gerechtfertigt ist.

2.1.2 Multiplikation

Multiplikationen werden an drei Stellen bei der Berechnung des SOM-Algorithmus verwendet: bei der Distanzberechnung zwischen Referenz- und Datenvektoren (Abstandsmaße), bei der Distanzberechnung zwischen Neuronen im Gitter (Nachbarschaftsfunktion) und bei der Adaption. Da Multiplikatoren in Hardware verhältnismäßig groß sind, werden verschiedene andere Verfahren eingesetzt, um die Multiplikation zu ersetzen. Die Auswirkung einer solchen Ersetzung hat Auswirkungen auf die folgenden Bereiche:

- **Abstandsmaße:** Der von Kohonen beschriebene Algorithmus spezifiziert kein konkretes Abstandsmaß, in der Regel wird allerdings die p_2-Norm, also die euklidische Distanz, verwendet. In Hardware-Implementierungen kommen häufig verschiedene andere Normen zum Einsatz: p_0 oder Maximum Norm, p_1 oder Manhattan-Norm und $(p_2)^2$ oder quadratisch-euklidische

Norm, die sich jeweils im Hardwareaufwand unterscheiden. Die p_0 Norm lässt sich sehr einfach durch einen Absolutwertbildner, einen Komparator und ein Register realisieren, und für p_1 wird statt dem Komparator ein Summationsglied verwendet. Für p_2 kommt zusätzlich ein Multiplizierer zum Einsatz, auf die Berechnung der Quadratwurzel kann ohne Auswirkungen auf die Größenverhältnisse verzichtet werden. Offensichtlich steigt der Hardwareaufwand mit der Komplexität der verwendeten Norm; da die entsprechende Schaltung in paralleler Hardware vielfach instanziiert wird, muss geklärt werden, welche Auswirkungen die verschiedenen Normen haben.

- **Nachbarschaftsfunktion:** Bei der Adaption spielt die Form der Nachbarschaftsfunktion eine wichtige Rolle; in [49, 48] wurde nachgewiesen, dass die Konvergenzeigenschaften der SOM wesentlich von der Form der Nachbarschaftsfunktion abhängen. Während Hardwareimplementierungen in der Regel so ausgelegt sind, dass beliebige Nachbarschaftsfunktionen emuliert werden können, beschränkt sich die effiziente Berechnung meist auf rautenförmige Nachbarschaftsfunktionen. Kreisförmige Nachbarschaftsfunktionen werden seltener verwendet, da für eine effiziente Realisierung Multiplizierer in jedem Rechenelement notwendig sind. Daher muss geklärt werden, inwiefern sich rautenförmige Nachbarschaftsfunktionen auf Quantisierungsfehler, topografischen Fehler etc. auswirken. Medrano et al. zeigen in [92], dass die topografische Ordnung bei der Verwendung der rautenförmigen Nachbarschaftsfunktion schlechter ist, als bei der runden.

- **Adaption:** Um die Komplexität einer Hardwareimplementierung zu verringern, kann die Multiplikation im Adaptionsschritt (Gleichung 1.2) durch einen Schiebeblock emuliert werden. Dadurch wird die Menge der möglichen Adaptionswerte von dem gesamten Werteumfang der verwendeten Zahldarstellung auf negative Zweierpotenzen, also 2^{-w} mit $w \in \mathbb{N}$ eingeschränkt. Daraus ergeben sich zwei Effekte: zum Einen können die Forderungen an die Adaptionswerte aus [41, 48] nach einer konkaven Nachbarschaftsfunktion, die als notwendige aber nicht hinreichende Kriterien für die Konvergenz der Referenzvektoren gelten, nicht für alle Konfigurationen erfüllt werden. Zum Anderen steigt der minimale Abstand zwischen Referenz- und Datenvektoren, der zu einer Adaption führt [132]. Dadurch kommt es schneller zu dem Effekt des *Einfrierens* der Referenzvektoren, die Konvergenzphase kommt mit allen negativen Folgen frühzeitig zum Stillstand.

2.2 Durchführung der Messungen

Damit ergeben sich drei prinzipielle Varianten, die bezüglich ihrer Funktionalität bewertet werden müssen:

- Die **Gleitkommadarstellung** mit doppelter Genauigkeit (IEEE754: double), sowohl für die Berechnung als auch für die Speicherung der Daten, dient als Referenzimplementierung, weil sie in den bekannten Beweisführungen und Softwareumgebungen für die Berechnung von SOM benutzt wird. Eine entsprechende Implementierung mit einfacher Genauigkeit dient dem Vergleich.

- Bei einer Implementierung basierend auf der **Festkommadarstellung** werden alle Werte (Referenz- und Datenvektoren sowie Adaptionswerte) in der jeweils angegebenen Wortbreite gespeichert. Da Festkommazahlen aber keine dynamische Anpassung des Wertebereichs unterstützen, müssen die Berechnungen (siehe Gleichungen 1.1 und 1.2) entsprechend mit einer höheren Wortbreite vorgenommen werden. Es werden unterschiedliche Wortbreiten untersucht, um ggf. eine minimale Wortbreite für die Implementierung einer SOM-Hardware zu finden.

- Bei der **Festkommadarstellung ohne Multiplizierer** werden zusätzlich die verwendeten Normen angepasst und die Adaptionswerte auf negative Zweierpotenzen (0,5 0,25 ...) eingeschränkt.

Der Vergleich dieser Varianten wird mit Hilfe von Testdatensätzen durchgeführt deren Eigenschaften im Anhang beschrieben werden. Die Parameter der SOM (Kartenform, Nachbarschaftsfunktion, Lerndauer) werden nach den Empfehlungen von Kohonen in [77] für jeden Datensatz individuell, aber für alle Varianten nach dem gleichen Muster festgelegt. Die Kartengröße wird wie von Vesanto [142] vorgeschlagen zu $5\sqrt{k}$ gewählt, in Ausnahmefällen kleiner. Jeder Lernvorgang wird 20-mal wiederholt und die Qualitätskriterien aus Abschnitt 1.3.1 werden berechnet. Es folgt ein Vergleich der gemittelten Werte für die Qualitätskriterien, der Aufschluss über die Auswirkungen der Veränderungen des SOM-Algorithmus gibt.

Als zusätzliches Maß dafür, inwieweit der SOM-Algorithmus die verwendeten Datentypen und Wortbreiten kompensiert, wird die folgende Messung für alle Qualitätskriterien, Datentypen und Wortbreiten durchgeführt: die Referenzvektoren $\mathbf{m}_i$ einer mit doppelter Präzision gelernter Karte werden entsprechend dem Datentypen und der Wortbreite w nach Gleichung 1.27 quantisiert, $\tilde{\mathbf{m}}_i = Q(\mathbf{m}_i, w)$. Anschließend wird die so erzeugte Karte $\tilde{M}_{ref} = Q(M_{ref})$ analog zu den anderen Implementierungen bewertet, und dient so als Referenz für die Qualitätskriterien. Liegt der Wert eines Kriteriums unter dem von $\tilde{M}_{ref}$, so wurden die Effekte der Quantisierung für dieses Kriterium wenigstens teilweise kompensiert.

Liegt er darüber, war eine Kompensation nicht möglich. Diese Messungen können aufzeigen, inwiefern sich das Gleichgewicht der unterschiedlichen Kriterien gegenüber der Referenzimplementierung verändert, bzw. inwiefern die Kompensation von Quantisierungseffekten hinsichtlich einiger Kriterien zu Lasten anderer Kriterien geht.

2.2.1 Bedeutung der Qualitätskriterien

Wie oben angedeutet haben die Qualitätskriterien bezogen auf die Implementierungsvarianten und bezogen auf den praktischen Einsatz unterschiedliche Aussagekraft. Diese soll im Folgenden erläutert werden:

Klassifizierungsfehler

Der Klassifizierungsfehler dient als primäres Merkmal bei der Bewertung unterschiedlicher Implementierungen, da dieser direkt die gewünschten Eigenschaften einer gelernten Karte widerspiegelt. Bei einem Datensatz, dessen Klassen bereits bekannt sind, kann dieser direkt berechnet werde. Bei Datensätzen mit unbekannter Anzahl und Zuordnung von Klassen wird eine Klassifizierung auf Basis der Referenzimplementierung erstellt und als Basis verwendet. Das Verfahren zur automatischen Klassifizierung wird im Anhang A.1 erläutert.

Die Aussage, die durch den Klassifizierungsfehler über eine Implementierung getroffen werden kann, ist insofern nicht vollständig, als dieser Fehler nur für eine willkürlich bestimmte Stichprobe berechnet werden kann, die beim Lernvorgang nicht berücksichtigt wurde. Um darüber hinaus eine Aussage über die Güte der Klassifizierung zu erhalten, wird der Quantisierungsfehler verwendet.

Quantisierungsfehler

Der Quantisierungsfehler als Maß für die Ähnlichkeit zwischen Referenzvektoren und Datenvektoren ergänzt die Aussage des Klassifizierungsfehlers um die Güte der Klassifizierung.

Topografische Funktion

Die Topografie-Erhaltung stellt die Basis für die Klassifikationseigenschaften der SOM dar und ist gleichzeitig die Ursache für eine Reihe sekundärer Effekte im Rahmen des Lernvorganges. Maße für die Topografieerhaltung stellen daher für die Beurteilung der Qualität einer SOM im Allgemeinen, als auch für die Erfassung des Einflusses quantisierter Referenzvektoren eine wichtige Rolle dar und werden im Folgenden untersucht. Die Ursache für den topografischen Fehler liegt im Dimensionsunterschied zwischen dem SOM-Gitter und dem Datensatz X: Ist die *inhärente* Dimension des Datensatzes X größer als die der SOM-Gitterstruktur, so muss der Datensatz *gefaltet* werden. Diese Faltungen, deren Stärke von

der Topografischen Funktion gemessen wird, verursachen topografische Defekte, da Vektoren, die in X nächste Nachbarn sind, im SOM-Gitter unter Umständen nicht auf benachbarte Neuronen abgebildet werden. Die *inhärente* Dimension des Datensatzes entspricht dabei nicht notwendigerweise der Dimension der Datenvektoren d_m, zum Beispiel wenn Komponenten von X linear abhängig sind.

Als Fehlermaße für die Topografieerhaltung kommen die Topografische Funktion und der Topografische Fehler als Spezialfall der Topografischen Funktion in Frage. Da sich die Topografische Funktion $\Phi(\kappa)$ nicht direkt für den Vergleich zweier Implementierungen eignet, wird im Folgenden ein skalares ϕ eingeführt, dass auch die Effekte für $-l < \kappa < l$ (siehe Gleichung 1.13) einbezieht:

$$\phi(\gamma_t) = \sum_{\kappa=-l}^{l} \Phi(\kappa)|\kappa|^{\gamma_t} \tag{2.4}$$

Dabei beschreibt γ_t die Art des Fehlermaßes:

- $\gamma_t = 0$: Summe aller $\Phi(\kappa)$ ohne Gewichtung
- $\gamma_t > 0$: Ein Maß für den Zusammenhang der Gebiete innerhalb der Karte. Ein hoher Wert bedeutet, dass nahe beieinanderliegende Strukturen auf der Karte weit voneinander entfernt sind. Eine Verkleinerung der Anzahl der Neuronen l verbessert in der Regel $\phi(\gamma)$.
- $\gamma_t < 0$: Ein Maß für die Topologieerhaltung, das die Abbildungen $\Psi_{X\to M}$ und $\Psi_{M\to X}$ gleichermaßen berücksichtigt.

Der Betrag von γ_t bestimmt die Gewichtung der κ, im Folgenden wird von $\gamma_t = -1$ ausgegangen. Der Vergleich zweier Implementierungen anhand des Fehlermaßes $\phi(\gamma_t)$ gibt letztlich, wie alle anderen bekannten Maße für die Topografieerhaltung, keine Auskunft über die Art der Faltung des Eingabedatenraumes in den Kartenraum. Stattdessen wird das Ausmaß der topografischen Fehler vermessen und somit eine indirekte Aussage über die Qualität der Topografieerhaltung gegeben.

Entropie

Die Entropie H als Maß für den Informationsgehalt der Karte ermöglicht den direkten Vergleich zweier Implementierungen unabhängig von Quantisierung und Topografieerhaltung. Unterschiede zwischen zwei Implementierungen bei der Entropie deuten auf eine unterschiedliche Verteilung der Datenvektoren auf Neuronen hin. Damit ergänzt die Entropie den Klassifizierungsfehler um Informationen über den Aufbau der Gebiete in der Karte. Bei der Bestimmung der Entropie werden hier ausschließlich aktive Neuronen betrachtet, d.h. Neuronen, die für den Datensatz als Erregungszentren dienen, siehe [94]. Alle inaktiven Neuronen, die in der Regel als *Separatoren* der Kartengebiete dienen, werden somit nicht berücksichtigt, und Effekte, die die Anzahl der inaktiven Neuronen erhöhen, werden von

diesem Maß nicht bewertet. Stattdessen kann hierfür der Vergrößerungsexponent, bzw. die Größe der Kartengebiete (s.u.) verwendet werden.

Vergrößerungsexponent

Der Vergrößerungsexponent als eine wichtige Abbildungseigenschaft der SOM sollte sich, insbesondere hinsichtlich einer Implementierung des Conscious- oder des BDH-Algorithmus nicht verändern. Er lässt sich nicht direkt bestimmen, stattdessen lassen sich die Größen der einzelnen Kartengebiete vergleichen.

2.3 Messergebnisse für SOM

Die oben beschriebenen Qualitätskriterien werden zunächst einzeln anhand ausgewählter Beispiele dargestellt und diskutiert. Zusätzlich werden die Messergebnisse aller Messungen eines Qualitätskriteriums normiert und in einem gemeinsamen Diagramm aufgetragen. Anhand der Ergebnisse wird untersucht, inwiefern die einzelnen Maße eine sinnvolle Aussage über den Lernvorgang erlauben und ob ggf. ein neues Maß definiert werden muss.

Bei der Darstellung der individuellen Beispiele kommen zwei Diagrammtypen zum Einsatz, die zunächst erläutert werden müssen. **Diagramm (a):** auf der Abszisse werden verschiedene Festkomma- und Gleitkommadatentypen aufgetragen, auf der Ordinate der Wert des jeweiligen Qualitätskriteriums. **Diagramm (b):** hier wird für alle untersuchten Datensätze der Anteil der Experimente mit einer Abweichung von 5% und mehr gegenüber der Referenzimplementierung aufgetragen. Dabei werden die Werte für alle Wortbreiten getrennt aufgetragen, es wird ausschließlich die Variante ohne Multiplizierer untersucht. In Diagramm (a) werden für die Qualitätskriterien (hier stellvertretend als K bezeichnet) die folgenden Werte aufgetragen:

- Mittelwert $\bar{K}$ und Standardabweichung $\sigma(K)$ (in Form einer Einhüllenden um den Mittelwert) des Qualitätskriterium werden jeweils für Gleitkommadatentypen, Festkommadatentypen mit und ohne Multiplikation aufgetragen. Dabei werden die Messwerte der Ganzzahldatentypen der bessern Lesbarkeit wegen miteinander verbunden.

- Um den Einfluss der Quantisierung visuell genauer beziffern zu können, werden die Punkte im Graphen gekennzeichnet, bei denen die Abweichung zwischen Festkommadarstellung und Referenz (*double*-Präzision) jeweils maximal 1% (□) bzw. max 5% (○) beträgt. Liegen diese Symbole außerhalb des bezeichneten Bereiches, so liegt keine Festkommadarstellung innerhalb der geforderten Umgebung. Die Qualitätskriterien unterscheiden sich in ihren Bezugssystemen; So ist der Topologische Fehler als prozentuale Angabe zu sehen, während der Quantisierungsfehler einen Absolutwert darstellt. Dies

wirkt sich auf die Berechnung der oben beschriebenen Grenzen insofern aus, als der Quantisierungsfehler zunächst über den Referenzfehler normiert werden muss, während die Grenzen bei den anderen Kriterien direkt berechnet werden können.

- Diese Werte werden jeweils für Implementierungen mit und ohne Multiplizierer (∅Mul.) aufgetragen.

- Die am Ende der Einleitung diese Kapitels beschriebene, zusätzliche Betrachtung einer nachträglich Quantisierten, mit doppelter Genauigkeit gelerneten Karte wird als $\bar{K}(\tilde{M}_{ref})$ aufgetragen.

Als Beispiele dient hier der Datensatz *Brustkrebs*, da diese ein stark unterschiedliches Verhalten aufweisen, welches für viele andere Datensätze repräsentativ ist. Im Rahmen dieser Arbeit wurden auch zahlreiche andere Datensätze untersucht, von denen eine Auswahl in Abschnitt 2.5 gezeigt wird.

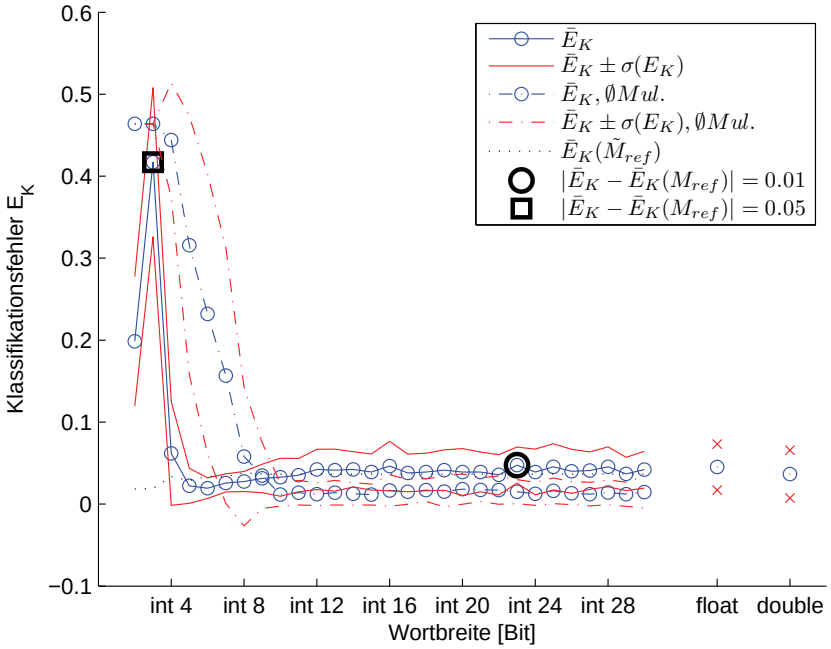
2.3.1 Klassifizierungsfehler

Die Kurven für den Klassifizierungsfehler in Abbildung 2.1 (a) zeigen sehr typische Verläufe mit einem charakteristischen Zacken bei sehr geringen Wortbreiten und einem nahezu konstanten Verlauf für Wortbreiten ab ca. 8 Bit. Die Spitze des Zackens stellt das Maximum des Klassifikationsfehlers von etwa $1/|C|$ mit der Anzahl der Cluster $|C|$ dar, verursacht durch einen nahezu vollständig degenerierten Lernvorgang bei dem die Zuordnung der Datenvektoren zu Clustern rein zufällig erfolgt. Für Wortbreiten kleiner als die des Maximums kann der Fehler erneut sinken, weitere Untersuchungen (siehe Abschnitt 2.3.4) werden jedoch zeigen, dass die Ausdehnung der Cluster sich hier auf sehr wenige Neuronen beschränkt. Mit steigender Wortbreite fällt der Fehler auf einen minimalen Wert und bleibt dann konstant, wobei dieser Wert sowohl kleiner als auch größer als der der Referenzimplementierung sein kann.

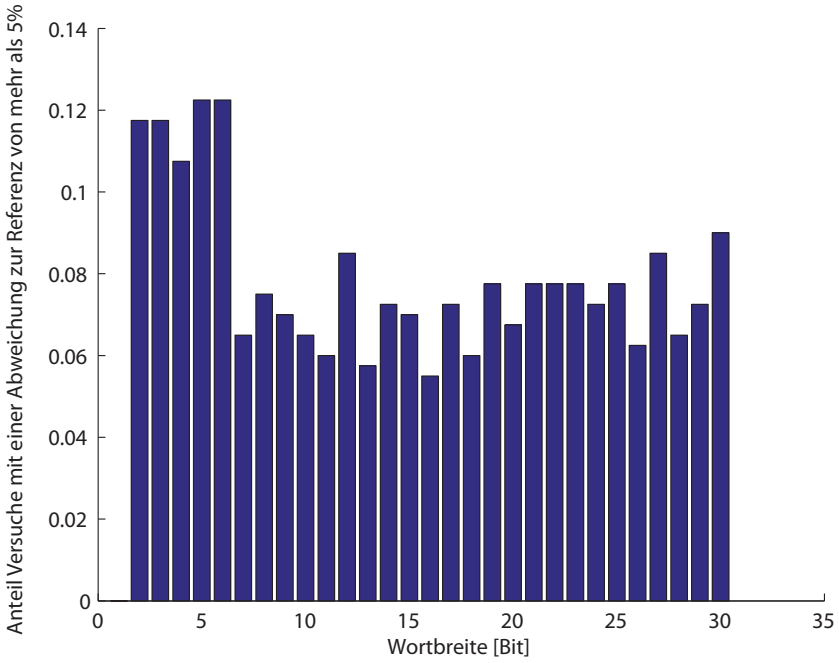
Der Verlauf von $\bar{E}_K(\tilde{M}_{ref})$ zeigt für den Brustkrebsdatensatz für Wortbreiten größer 10 Bit keine relevanten Differenzen zum Verlauf von $\bar{E}_K$.

Die gestrichelten Linien beschreiben den Klassifikationsfehler für die Implementierungen mit Ganzzahl-Arithmetik und ohne Multiplikationen, siehe Abschnitt 2.1.2. Auch hier können die Verläufe sowohl über, als auch unter der Referenzimplementierung liegen; weitere Ergebnisse zeigen aber, dass sowohl die topologische Ordnung, die Quantisierung als auch der Informationsgehalt der Karte (Entropie) für diesen Fall wesentlich schlechter sind, als für eine Implementierung mit Multiplizierern.

Die Positionen der Marker für das Durchschreiten der 1%- (□) bzw. der 5%-Grenze (○) zeigen deutlich, dass der Einfluss der hardwarespezifischen Änderungen wesentlich vom Datensatz abhängen. Dies wird ebenfalls in Abbildung 2.1



(a) Datensatz Brustkrebs



(b) Anteil der Versuche mit $|E_K - \bar{E}_{K,ref}| \geq 0.05$ für alle Datensätze

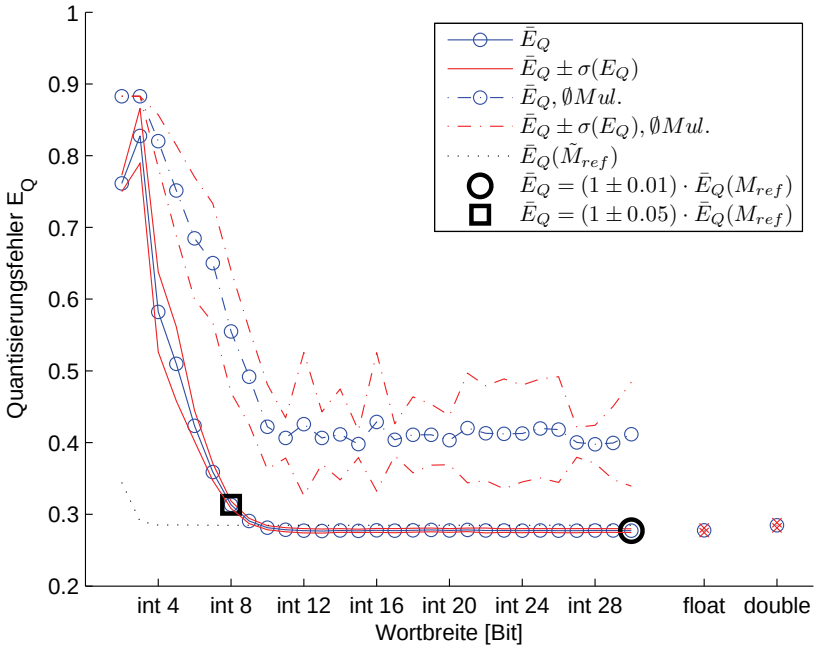
Abbildung 2.1: Klassifikationsfehler bei verschiedenen Datentypen

(b) deutlich. Das Balkendiagramm in Abbildung 2.1 (b) zeigt, wie groß der Anteil der Versuche mit einem *Unterschied* größer 5% von der Referenzimplementierung liegt. Wichtig ist zu bemerken, dass es sich ausschließlich um einen Unterschied, nicht unbedingt aber um einen Fehler handelt. Es kann nicht davon ausgegangen werden, dass die ursprüngliche Klassifikation des Datensatzes mit Hilfe einer Gleitkommaimplementierung korrekt ist, darüber hinaus sind auch in vorgegebenen Klassifikationen (etwa in [108]) eindeutige Fehler zu erkennen. Für diese fehlerhaften Referenzen gibt es im Wesentlichen zwei mögliche Ursachen: zum einen können in einem nicht klassifizierten Datensatz Datenpunkte sehr nach an der Grenze zwischen zwei Clustern liegen, die dann in unterschiedlichen Lernvorgängen unterschiedlichen Clustern zugeordnet werden. Dies tritt vor allem dann auf, wenn die Daten keine deutlich trennbaren Strukturen enthalten. Zum anderen können bei aus der Literatur bekannten Klassifikationen sehr leicht Punkte enthalten sein, die von einem automatischen Klassifikationsverfahren anders erkannt werden, denn solche Klassifikationen stammen häufig aus menschlichen Beobachtungen. So sind beispielsweise im Brustkrebsdatensatz einige Punkte enthalten, die nach einer Klassifikation durch SOM eindeutig dem jeweils anderen Cluster zugeordnet werden müssen. Entweder wurde hier ein Fehler durch den behandelnden Arzt begangen, oder aber dem Algorithmus stehen nicht alle Parameter zur Verfügung, die eine korrekte Klassifizierung ermöglichen würde.

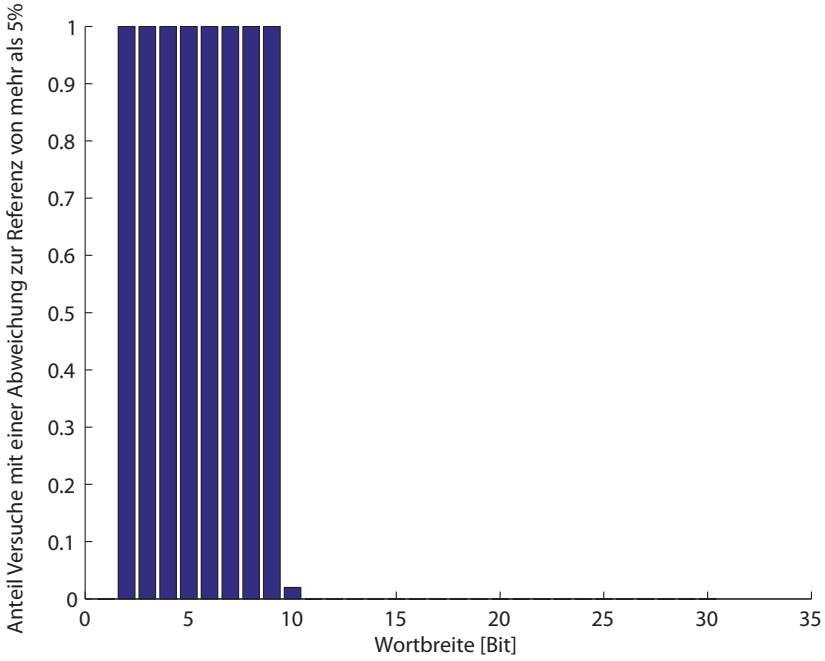
2.3.2 Quantisierungsfehler

Der Einfluss der verwendeten Datentypen auf den Quantisierungsfehler ist in Abbildung 2.2 (a) dargestellt. Der Verlauf ähnelt mit dem lokalen Maximum bei niedrigen und dem konstanten Verlauf bei größeren Wortbreiten prinzipiell dem des Klassifizierungsfehlers, auch das lokale Minimum bei der niedrigsten Wortbreite ist vorhanden. Interessant ist der Verlauf der Referenzwerte $\bar{E}_Q(\tilde{M}_{ref})$, die ab einer Wortbreite von 10 Bit (Brustkrebs) praktisch nicht vom Verlauf der gemessenen Werte zu unterscheiden ist. Unter diesen Wortbreiten steigen die gemessenen Fehler wesentlich stärker an als die der Referenzen, ein weiterer Hinweis darauf, dass sich der Lernvorgang gegenüber der Referenzimplementierung verändert. Im Gegensatz zum Klassifikationsfehler unterscheiden sich die Implementierungen mit und ohne Multiplizierer hier eindeutig durch einen zwei- bis dreimal größeren Fehler und einer deutlich erhöhten Volatilität bei der Variante ohne Multiplizierer.

Abbildung 2.2(b) bestätigt die Vermutung, dass ab einer bestimmten Wortbreite keine Änderung des Quantisierungsfehlers zu erwarten ist, denn erst bei Wortbreiten von unter 11 Bit können Abweichungen von über 5% festgestellt werden.



(a) Datensatz Brustkrebs



(b) $|E_Q/\bar{E}_{Q,ref} - 1| \geq 0.05$ für alle Datensätze

Abbildung 2.2: Quantisierungsfehler bei verschiedenen Datentypen

2.3.3 Topografischer Fehler

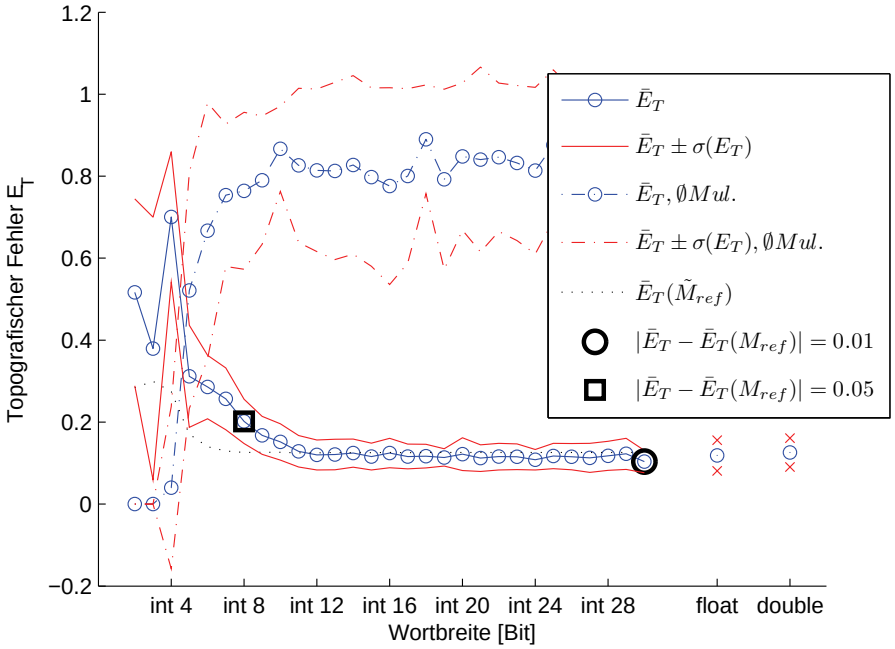
Bei der Darstellung des topografischen Fehlers in Abbildung 2.3 (a) zeigt sich erneut ein ähnliches Verhalten wie für den Klassifikationsfehler in Abbildung 2.1, auch hier ergeben sich für Wortbreiten größer 10 Bit keine signifikanten Veränderungen gegenüber der Referenzimplementierung. Gleichzeitig ist zu bemerken, dass die Implementierung ohne Multiplizierer praktisch keine topografische Ordnung erzeugt, zu erkennen an dem nahezu maximalen Wert für den topografischen Fehler ($E_T \approx 1$).

Abbildung 2.3 (b) zeigt, dass weniger als 10 Prozent der Versuche mit Wortbreiten größer 8 Bit einen Fehler von mehr als 5% gegenüber der Referenzimplementierung aufweisen.

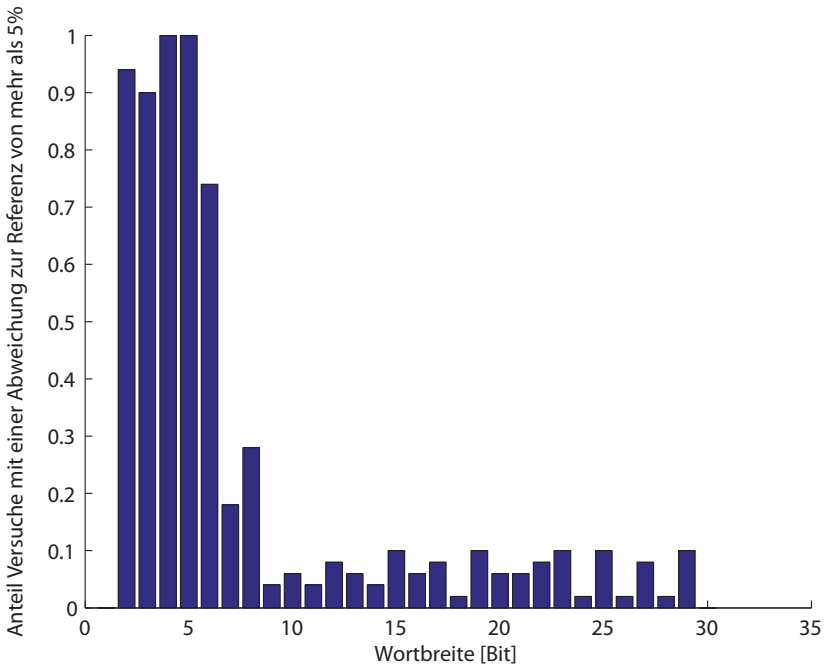
Ein Vergleich der Abbildungen 2.3 und 2.2, besonders im Bereich geringer Wortbreiten, zeigt deutlich die Gegensätzlichkeit der Kriterien für Topografieerhaltung und Quantisierung. Während der Quantisierungsfehler mit geringerer Wortbreite monoton zunimmt, kommt es bei dem topografischen Fehler unter Umständen zu einem lokalen Minimum. Der Grund dafür liegt in der Anzahl der aktiven Neuronen in der Karte: da die Voronoigebiete der Referenzvektoren mit geringer die Wortbreite ansteigen, fallen immer mehr Datenvektoren auf die gleichen Erregungszentren. Die Anzahl der aktiven Neuronen geht also zurück und topologische Fehler, die bei einer größeren Wortbreite und damit verbunden einer größeren Streuung der Referenzvektoren auftreten können, werden ausgeglichen gemacht. Anders ausgedrückt schrumpft die Anzahl der aktiven Elemente auf ein Minimum, und praktisch alle Datenvektoren haben benachbarte Referenzvektoren. In der umgekehrten Richtung, also der Abbildung von den Referenzvektoren zu den Datenvektoren ergibt sich ein ähnliches Bild dadurch, dass mehr und mehr Datenvektoren, bedingt durch die geringe Wortbreite, zusammenfallen und somit ebenfalls zu nächsten Nachbarn werden. Es zeigt sich also, dass die alleinige Betrachtung des topografischen Fehlers zumindest in Untersuchungen mit variierender Wortbreite keinesfalls für die Beurteilung der Qualität einer gelernten Karte ausreicht.

2.3.4 Entropie

Zur Entropie ist grundsätzlich zu bemerken, dass diese bei den untersuchten Datensätzen für die Kohonen-SOM niemals maximal (= 1) wird, d.h. es existiert immer Redundanz innerhalb der gelernten Karte. Stattdessen werden für Gleitkomma- und für Festkommadatentypen mit einer Wortbreite größer 12 Bit (siehe Abbildung 2.4 (a)) nahezu konstante Werte von etwa $H = 0.95$ festgestellt. In diesem Bereich liegen auch die gemessenen Werte für die Festkommadatentypen nur minimal unterhalb der von $H(\tilde{M})$ vorhergesagten Entropie, $\Delta(H) \leq 0.01$. Für die Implementierungen ohne Multiplizierer ergeben sich Werte, die deutlich

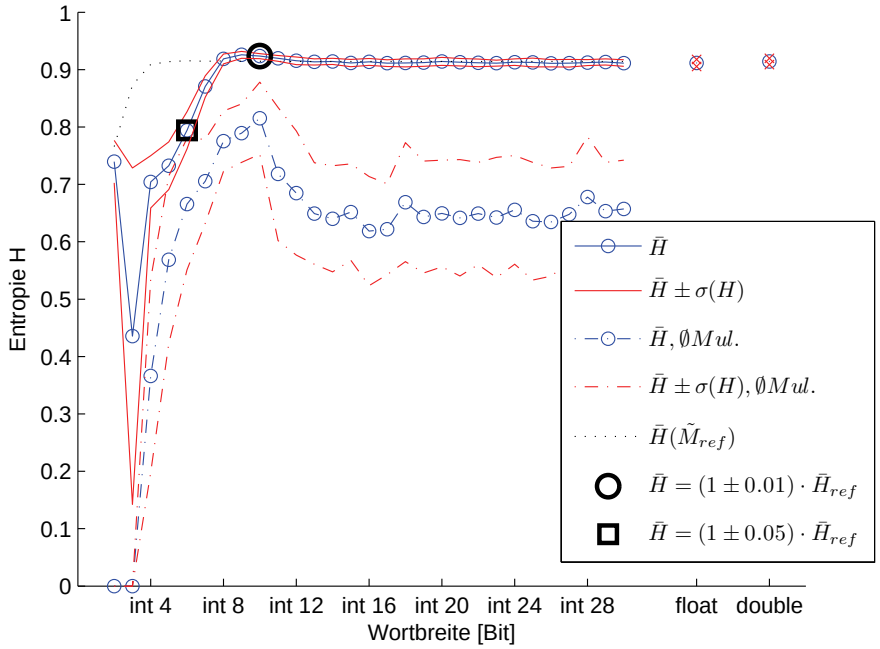


(a) Datensatz Brustkrebs

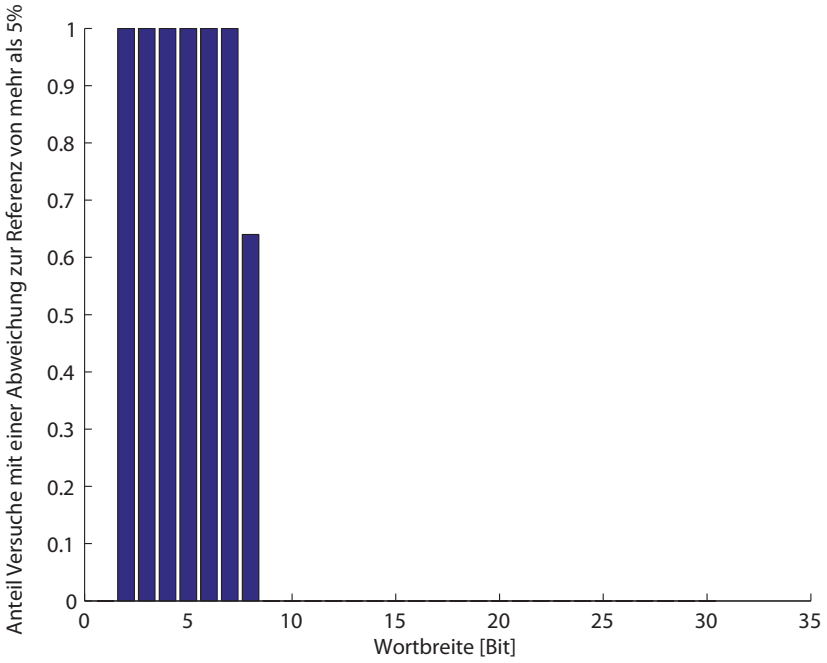


(b) $|E_T - \bar{E}_{T,ref}| \geq 0.05$ für alle Datensätze

Abbildung 2.3: Topografischer Fehler bei verschiedenen Datentypen



(a) Datensatz Brustkrebs



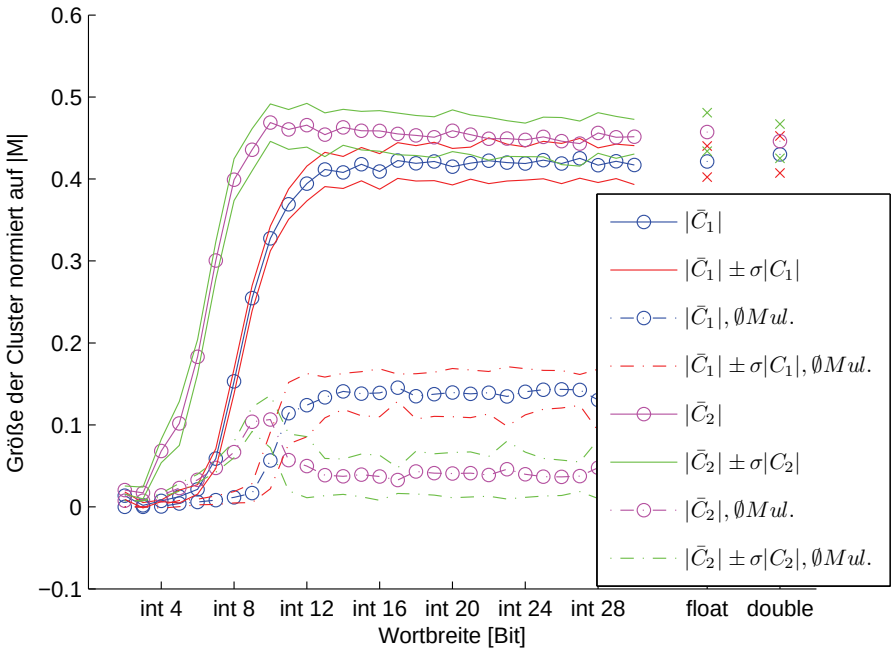
(b) $|H - \bar{H}_{ref}| \geq 0.05$ für alle Datensätze

Abbildung 2.4: Entropie bei verschiedenen Datentypen

unter der Referenz liegen, Abweichungen von $\Delta(H) \geq 0.2$ treten auf; zusätzlich ergeben sich im Verlauf der Entropie, auch bei großen Wortbreiten, sehr große Varianzen auf, die zuverlässige Aussagen über Lernergebnisse praktisch unmöglich machen.

Abbildung 2.4 (b)zeigt, dass sich auch hier die Abweichungen von der Referenzimplementierung bei Wortbreiten größer 9 Bit innerhalb der Toleranz von 5% befinden.

2.3.5 Vergrößerungsexponent



(a) Datensatz Brustkrebs

Abbildung 2.5: Clustergrößen bei verschiedenen Datentypen

Die Bestimmung des Vergößerungsexponenten ist nur indirekt (siehe [113]) und nur für sehr einfache Konfigurationen möglich. Da der Vergößerungsexponent das Verhältnis der Kartengebiete bestimmt, die den Clustern im Eingabedatensatz X zugeordnet werden, können zur Quantifizierung eventueller Variationen näherungsweise die Größen der gefundenen Kartengebiete verwendet werden. Diese sind in Abbildung 2.5 relativ zur Gesamtzahl der Neuronen aufgetragen. Dabei stellt $|\bar{C}_i|$ die durchschnittliche Größe des Clusters i bei einer Festkommaimplementierung mit Multiplizierer dar, $|\bar{C}_i|\emptyset Mul.$ entsprechend bei einer Festkommaimplementierung ohne Multiplizierer. Zusätzlich wurde jeweils die Standardabweichung als Einhüllende eingezeichnet.

Grundsätzlich ist zu bemerken, dass die Summe der Clustergrößen kleiner eins ist, was auf die Existenz inaktiver Neuronen zurückzuführen ist. Inaktive Neuronen sind Neuronen, die keinem Datenvektor als Erregungszentrum zugeordnet sind; sie finden sich gewöhnlich in den Grenzbereichen zwischen Clustern. Für die Implementierungen mit Multiplizierer wird sofort deutlich, dass es in Bezug auf die Referenzimplementierung jeweils eine geringe Abweichung der absoluten Clustergrößen und deren Verhältnis zueinander gibt. Darüber hinaus geht mit einer sinkenden Wortbreite eine Veränderung des Vergrößerungsexponenten einher: Bis zu einer Wortbreite von 12 Bit sinkt die Größe des kleineren Clusters und die Größe des größeren Clusters steigt an, was auf einen Anstieg von $\rho \to 1$ für sinkende Wortbreiten hinweist. Unterhalb der kritischen Grenze von 12 Bit verändern sich sowohl die absoluten Größen als auch deren Verhältnis drastisch, effektiv wird nur noch ein kleiner Bereich der Karte ausgenutzt. Während andere Datensätze ein ähnliches Verhalten zeigen, sollte diese Aussage, insbesondere auch wegen der sehr geringen Ausprägung des Effekts nicht ohne weiteres generalisiert werden.

Für die Implementierungen ohne Multiplizierer ist festzustellen, dass die absoluten Größen der Kartengebiete stark von der Referenz abweichen, und dass das Verhältnis der Clustergrößen untereinander sich umkehrt. Offenbar gilt für diese Implementierung $\rho < 0$, was im Allgemeinen mit *Perceptual Magnet Effect* bezeichnet wird. Die anderen untersuchten Datensätze zeigen ein ganz ähnliches Verhalten, es bleibt aber zu klären, ob sich dieser Effekt kontrollieren und ausnutzen lässt. Unterhalb einer kritischen Wortbreite, die zwischen den Beispielen variiert, kommt es erneut zu einer Umkehr der Größenverhältnisse in Richtung der Referenzimplementierung, allerdings weiterhin bei einer sehr geringen Ausnutzung der Karte.

2.4 Interpretation der Ergebnisse

Im vorangegangenen Abschnitt wurden Messungen auf Basis von Benchmarkdatensätzen durchgeführt und einzeln analysiert. Dabei zeigt sich vor allem, dass die Beurteilung einer SOM-Implementierung keinesfalls auf Basis eines einzelnen Qualitätskriteriums erfolgen darf. Auch die in der Literatur häufig vorzufindende Kombination aus Quantisierungs- und topografischem Fehler sind nicht ausreichend, um Implementierungen miteinander zu vergleichen. Stattdessen zeigt sich in vielen Beispielen, dass die bekannten Fehlermaße unter Umständen gegensätzliche und scheinbar widersprüchliche Tendenzen aufzeigen, wie z.B. Topografischer Fehler und Quantisierungsfehler, siehe Abschnitt 2.3.3. In Konsequenz muss für den Vergleich von SOM-Implementierungen ein Maß gefunden werden, welches die Aussagen der oben verwendeten Maße vereint; ein skalares Maß kann den Entscheidungsprozess vereinfachen. Ein mögliches Maß wird in Abschnitt 2.5 definiert und auf die untersuchten Datensätze angewandt.

Zusätzlich zeigt sich, dass keine allgemeingültige Aussage zu den Mindestanforderungen an eine Hardwareimplementierung, die der Referenzimplementierung hinsichtlich der Qualitätskriterien bis auf einen tolerierbaren Fehler gleicht, getroffen werden kann. Zwar sind für alle untersuchten Datensätze bei einer Wortbreite von 16 Bit keine Abweichungen größer 5% festzustellen. Aufgrund der kleinen Auswahl an Datensätzen kann dies nicht auf die Allgemeinheit übertragen werden. Dies wird insbesondere vor dem Hintergrund der bekannten Abbildungsgesetze der SOM deutlich, die besagen, dass die SOM eine Approximation an $p(X)$ darstellt. Alleine die Quantisierung von X verändert das i.d.R. ohnehin unbekannte $p(X)$, so dass keine Aussage über den Einfluss der Quantisierung getroffen werden kann. Unter der Hypothese, dass den Datensätzen aus demselben Bereich, etwa des *Hyperspectral Imaging*, ein ähnliches[1] $p(X)$ zugrundeliegt, muss für die Bestimmung der Hardware-Mindestanforderungen eine exemplarische Untersuchung wie in Abschnitt 2.2 beschrieben erfolgen.

2.5 Vergleich der Implementierungsvarianten

In den vorangegangenen Abschnitten zeigt sich, dass keines der präsentierten Qualitätskriterien alleine für eine qualifizierte Aussage bezüglich der Unterschiede zwischen zwei Implementierungen herangezogen werden kann. Die teilweise gegenläufigen Aussagen müssen in einem einzelnen Maß zusammengefasst und somit vergleichbar gemacht werden. Ein solches Maß wird im Folgenden definiert und auf die gemessenen Werte angewandt.

Das Maß Q basiert auf den korrekt normierten Werten der weiter oben definierten Qualitätskriterien, und ist damit $Q(E_K, E_T, E_Q, H, |C_1|, ..., |C_u|))$, wobei $u = |C|$ die Anzahl der Cluster und $|C_i|$ deren Größe definiert:

$$Q = (1 + \bar{E}_{K,ref} - E_K)^{\gamma_K} (1 + E_{T,ref} - E_T)^{\gamma_T} (1 + \bar{H}_{ref} - H)^{\gamma_H} \cdot \left(\frac{E_{Q,ref}}{E_Q}\right)^{\gamma_Q} \left(\prod_{i=1}^{|C|} \left[1 - \left|\frac{|C_{i,ref}| - |C_i|}{|M|}\right|\right]\right)^{\frac{\gamma_C}{|C|}} \tag{2.5}$$

Q ist im mathematischen Sinne eine Metrik im $n + 4$ dimensionalen Raum der Qualitätsmerkmale, die die einzelnen Kriterien über die Exponenten γ_K, γ_T, γ_Q, γ_H und γ_C gewichtet. Während alle Teilmaße bis auf den Vergrößerungsexponenten direkt in die Metrik eingehen, kann die Veränderung des Vergrößerungsexponenten nur indirekt über die Veränderung der Clustergrößen angegeben werden. Da die Veränderungen der Clustergrößen bei Verschiebungen zwischen den Clustern in der u-ten Potenz in das Produkt eingehen, wurde der Korrekturfaktor $u^{-1} = |C|^{-1}$ eingeführt. Da über diesen Wert nicht bestimmt werden kann,

[1] Eine mathematisch exakte Definition der Ähnlichkeit zwischen zwei WDF muss an dieser Stelle fehlen

in welche Richtung sich der Wert für ρ verschiebt, muss der Gewichtung dieses Faktors besondere Aufmerksamkeit gewidmet werden.

Die Metrik Q kann als Abstandsmaß zwischen zwei Implementierungen Werte zwischen null und unendlich annehmen. Hier wird immer der Abstand zwischen einer beliebigen Implementierung und einer Referenz erzeugt, wobei die Referenz durch die Implementierung mit *double*-Präzision repräsentiert wird. Insofern können die Werte von Q in drei Bereiche eingeteilt werden:

- $Q \lessapprox 1$: In diesem Fall weist die zu bewertende Implementierung eine geringere Qualität als die Referenz auf.
- $Q \approx 1$: Die Qualität der zu bewertenden Implementierung gleicht denen der Referenz. Dies bedeutet im Umkehrschluss aber nicht, dass die Ergebnisse der Implementierungen gleich sind, da sich einzelne Faktoren unter Umständen ausgleichen können. Dieses Verhalten kann über die Gewichtung der Faktoren gesteuert werden.
- $Q \gtrapprox 1$: Für diesen Fall übersteigt die Qualität der zu bewertenden Implementierung die der Referenz, wobei auch hier die Gewichtung zu beachten ist.

Die Gewichte γ können beliebige Werte annehmen. Im Folgenden werden alle Gewichte mit Ausnahme von γ_K gleich eins gesetzt. Aufgrund der besonderen Bedeutung der Klassifikationseigenschaften gilt im Folgenden $\gamma_K = 2$. In den folgenden Abschnitten wird die soeben definierte Metrik auf die in Abschnitt 2.1 beschriebenen und in den Abschnitten 2.3.1 bis 2.3.5 vermessenen Implementierungen angewandt.

2.5.1 Vergleich der Kohonen-SOM-Implementierungen

In Abbildung 2.6 wurden die Ergebnisse für die Metrik Q bei den in Abschnitt A.2 beschriebenen Datensätzen aufgetragen. Zusätzlich wurde zu dem Referenzwert eins eine Umgebung von $\pm 5\%$ eingezeichnet, die als Beispiel für eine tolerierbare Qualitätsveränderung dienen soll.

1. Für jeden Datensatz gibt es eine kritische Wortbreite, unterhalb derer das Lernergebnis stark von der Referenzimplementierung abweicht. Diese Grenze variiert für die betrachteten Datensätze zwischen 7 und 18 Bit.
2. Oberhalb der kritischen Wortbreite variieren die Werte für Q in einem Bereich von maximal $\Delta Q = 0,05$, ohne dabei einem erkennbaren Muster zu folgen. Abhängig von den Ansprüchen an die Vergleichbarkeit von Referenz und Hardwareimplementierung kann also für jeden Datensatz eine kritische Grenze festgelegt werden, oberhalb derer kein Zugewinn an Qualität bzw. Vergleichbarkeit mehr sichergestellt werden kann.

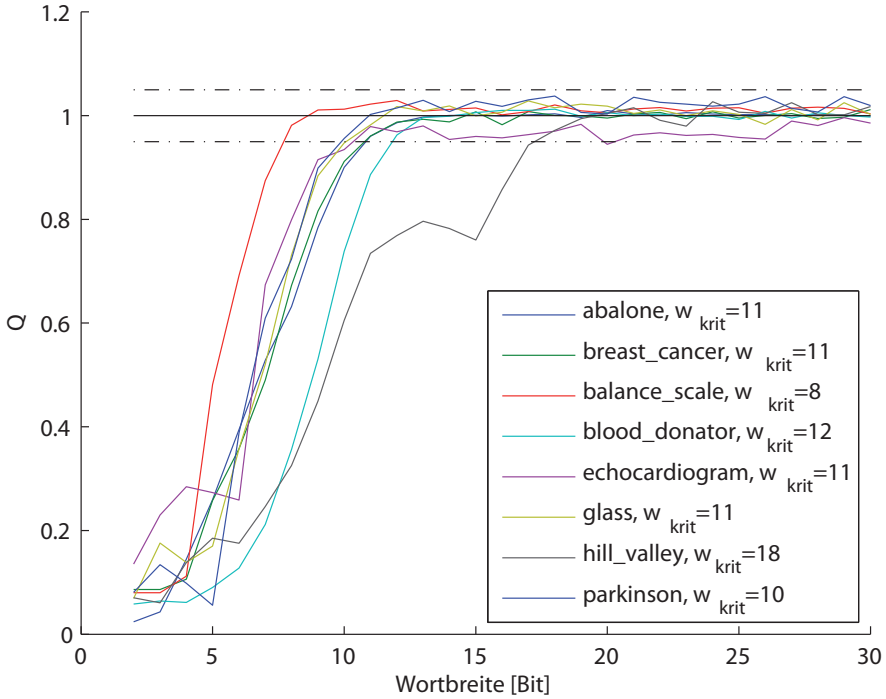


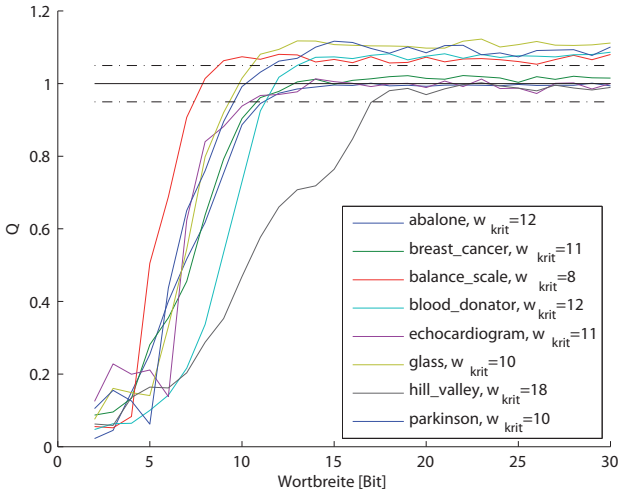
Abbildung 2.6: Metrik Q für die Kohonen-SOM

Die Messungen zeigen, dass die hardwarespezifischen Anpassungen des SOM-Algorithmus stets einen Einfluss auf das Lernergebnis haben, dieser ist aber nicht a-priori vorhersagbar. Es zeigt sich, dass eine kritische Wortbreite existiert, oberhalb derer sich die Veränderungen von Q in einem sehr eng begrenzten Bereich abspielen. Insbesondere ist oberhalb der kritischen Wortbreite keine konstante Qualitätssteigerung zu beobachten; die Werte für Q variieren ohne erkennbares Muster abhängig von der Wortbreite.

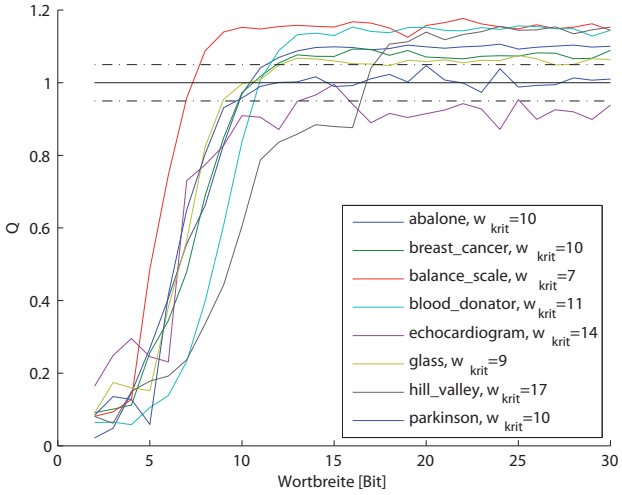
Für eine potentielle Hardwareimplementierung sind daraus zwei Schlüsse zu ziehen. Erstens muss für jeden Einsatzbereich im Vorhinein bestimmt werden, welche Wortbreite für den Lernvorgang benötigt wird, bzw. ob der Lernvorgang auf der gegebenen Hardware sinnvoll durchgeführt werden kann. Zweitens ist festzustellen, dass eine Vergrößerung der Wortbreite oberhalb der so festgestellten Grenze keinen Qualitätsgewinn sicherstellt, und somit überflüssig ist. Dies ist insbesondere deshalb wichtig, weil mit einer Vergrößerung der Wortbreite auch eine Steigerung der Ressourcenbelegung der Implementierung einhergeht.

Variation der Lernparameter

In der Literatur ist häufig zu finden, dass die Qualitätsverluste, die durch hardwarespezifische Anpassungen des SOM-Algorithmus hervorgerufen werden, durch



(a) Q bei doppelter Anzahl an Epochen



(b) Q bei 125% Kartengröße (125% Neuronen)

Abbildung 2.7: Variation der Lernparameter für die Kohonen-SOM

eine Adaption der Lernparameter (Dauer des Lernvorganges, Größe der Karte, Verlauf von α etc.) ausgeglichen werden können. Diese Variationen und Ihre Auswirkungen sollen im Folgenden untersucht werden.

Die Abbildungen 2.7 (a) und (b) zeigen die Auswirkung (a) einer Verdoppelung der Lerndauer und (b) einer Vergrößerung der Karte um 25%. Es ist sofort ersichtlich, dass eine Erhöhung der Anzahl von Lernschritten zu einem qualitativ besseren Lernergebnis führt, was aber in Anbetracht des typischen Verlaufes von Quantisierungs- und topografischem Fehler (Abbildung 1.5) nicht anders zu erwarten ist: Solange sich die Karte in einem Zustand befindet, in dem sich

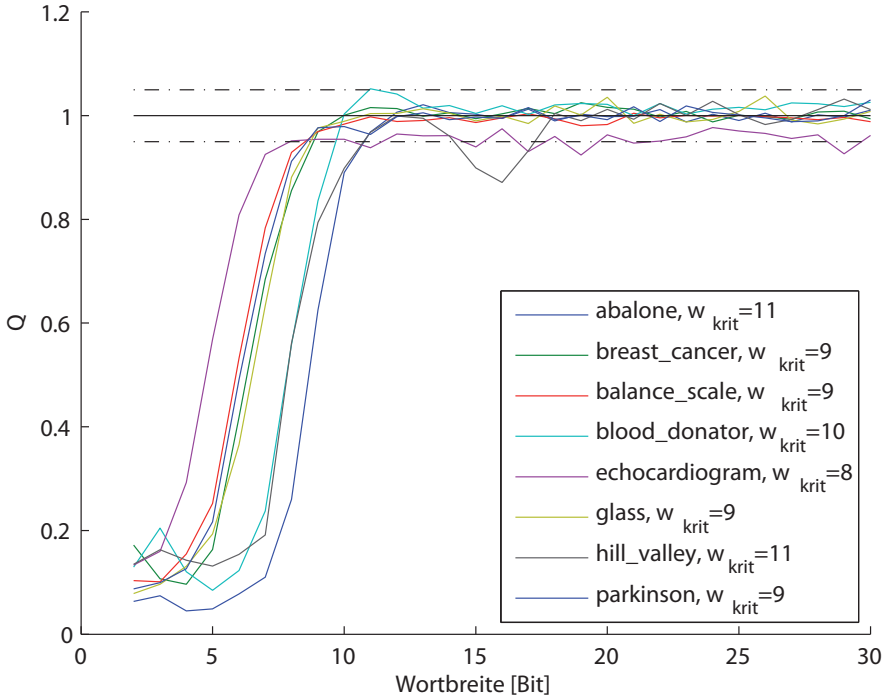


Abbildung 2.8: Metrik Q für die Conscious-SOM

Quantisierungs- und topografischer Fehler durch weitere Lernschritte minimieren lassen, ist dieser Effekt für alle funktionierenden Implementierungen gleichermaßen zu erwarten. Wird als Referenz eine Implementierung mit *double*-Präzision verwendet, so ergibt sich kein Unterschied zum originalen Vergleich zwischen den Implementierungen mit einfacher Lerndauer. Gleichermaßen ist zu erwarten, dass die Verlängerung der Lerndauer keinen ausgleichenden Effekt auf die Auswirkungen der hardwarespezifischen Anpassungen hat, falls die Karte sich bereits in einem konvergierten Zustand befindet. Da die Anzahl an Lernschritten bis zum Erreichen dieses Zustandes nicht a-priori bekannt ist, kann daher nicht vorausgesetzt werden, dass eine Erhöhung der Lerndauer den gewünschten Ausgleich erbringt.

Gleichzeitig zeigt Abbildung 2.7 (b), dass auch eine Vergrößerung der Karte die Auswirkungen der hardwarespezifischen Anpassungen ausgleichen kann, wobei weder hier, noch bei der Erhöhung der Lerndauer bekannt ist, um welchen Betrag vergrößert bzw. verlängert werden muss.

2.5.2 Vergleich der Conscious-SOM-Implementierungen

Für die Conscious-SOM wurden dieselben Experimente wie in Abschnitt 2.2 beschrieben durchgeführt. Die Ergebnisse werden hier nicht im Einzelnen sondern nur in Form der Metrik Q diskutiert. In Abbildung 2.8 zeigen sich nur wenige Unterschiede zur Kohonen-SOM-Implementierung: auch hier zeigt sich eine kritische Wortbreite w_{krit} unterhalb derer die Werte für Q stark absinken, unterschiedlich sind dagegen die Werte von w_{krit}. Diese liegen im Durchschnitt etwas unter denen für die Kohonen-SOM, auffällig ist aber, dass die Reihenfolge der w_{krit} für die einzelnen Datensätze eine andere ist, als für die Kohonen SOM. Ein weiterer Unterschied liegt in den absoluten Werten für Q oberhalb der kritischen Wortbreite: während die Werte für den Datensatz *Abalone* und *Balance Scale* bei der Kohonen-SOM stets über der Referenzimplementierung liegen, variieren sie für die Conscious-SOM um die Referenz. Insgesamt sind hier die Unterschiede zwischen Referenz- und Hardwareimplementierung also kleiner als bei der Kohonen-SOM und die kritische Wortbreite w_{krit} liegt im Durchschnitt niedriger als die der Kohonen-SOM.

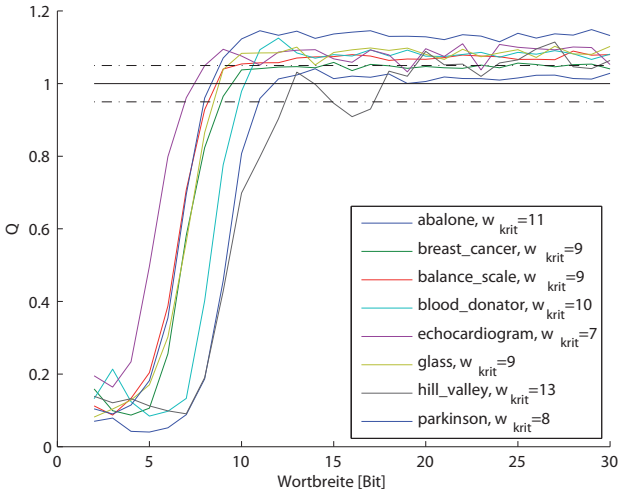
Daraus lässt sich schließen, dass die Einführung des *Gewissens* bei der Conscious-SOM die Effekte der hardwarespezifischen Anpassungen teilweise ausgleicht. Entsprechend ist diese Variante der SOM besser für eine Hardware-Implementierung geeignet, als die originale Implementierung von Kohonen.

Variation der Lernparameter

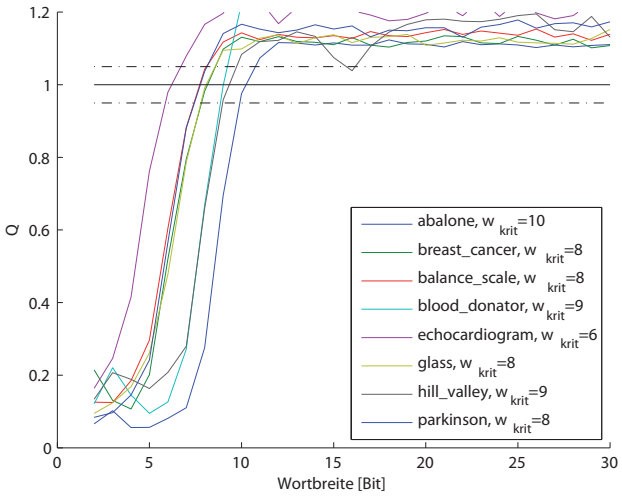
Bei dieser Variante der SOM kommen zusätzlich zu den weiter oben beschriebenen eine Reihe zusätzlicher Parameter für die Variation in Frage, insbesondere der Verlauf der Parameter β und γ. Da diese aber im Zusammenhang mit dem Verlauf von α zu sehen sind, würde dies eine exponentielle Vergrößerung des Parameterraumes ergeben und damit den Rahmen dieser Arbeit sprengen. Stattdessen werden hier ausschließlich die Ergebnisse für die Variationen der Kartengröße und der Lerndauer präsentiert. Abbildung 2.9 zeigt deutlich, dass auch für die Conscious-SOM eine Verlängerung der Lerndauer oder eine Vergrößerung der Karte die Effekte hardwarespezifischer Anpassungen ausgleichen können. Dabei bleibt zu beachten, dass dieser Ausgleich nicht in allen Bereichen des SOM-Lernvorganges erfolgen kann (s.o.). Zusätzlich kann nicht a-priori bestimmt werden, um welchen Betrag erhöht bzw. vergrößert werden muss.

2.6 Weitere hardwarespezifische Anpassungen

Zusätzlich zu den in Abschnitt 2.1 beschriebenen Anpassungen hinsichtlich Quantisierung und Multiplikation werden in Hardware häufig noch weitere Anpassungen vorgenommen, die im Folgenden diskutiert werden sollen.



(a) Q bei doppelter Anzahl an Epochen



(b) Q bei 125% Kartengröße

Abbildung 2.9: Variation der Lernparameter für die Conscious-SOM

- **Abstandsberechnung mit $p1$-Norm:** Bei der Abstandsberechnung zwischen Referenz- und Datenvektoren mit der quadratisch-euklidischen Norm wird eine Summationseinheit mit maximal $w_{sum,p2} = 2w + \lceil \log_2(d_m) \rceil$ Bit Wortbreite benötigt, um einen Überlauf in jedem Fall zu verhindern. Bei der Manhattan Norm ($p1$) werden nur $w_{sum,p1} = w + \lceil \log_2(d_m) \rceil$ Bit benötigt, also w Bit weniger. Ausgehend von einer quadratischen Abhängigkeit von Chip-Fläche und Wortbreite (bei gleicher maximaler Taktfrequenz) wird bei Verwendung der Manhattan-Norm entsprechend $100 \cdot w_{sum,p1}/w_{sum,p2}$ Prozent Fläche pro Neuron eingespart.

- **Auswahl der Vektoren:** Der SOM-Algorithmus schreibt eine zufällige Auswahl der pro Lernschritt zu erlernenden Vektoren vor, und auch alle theoretischen Ergebnisse (siehe Abschnitt 1.4) setzen diese Auswahl voraus. Für eine praktische Implementierung, bei der die Daten in einem Speicher abgelegt werden, ist diese Zugriffsmethode die schlechtestmögliche. Dies wird sofort deutlich, wenn Cache-Strategien betrachtet werden: durch die zufällige Auswahl der Vektoren kann die Speicherzugriffsreihenfolge offensichtlich nicht vorhergesagt werden. Sobald der Datensatz also nicht mehr komplett im Cache gespeichert werden kann, kommt es im schlimmsten Fall in jedem Lernschritt zu einem Cache-Miss, d.h. die Daten müssen aus dem langsamen Hauptspeicher nachgeladen werden. Stattdessen wird in Hardwareimplementierungen häufig eine andere Strategie verwendet: Die Zugriffsreihenfolge wird vor dem Lernvorgang für $|X|$ Lernschritte festgelegt und die Vektoren entsprechend im Speicher abgelegt. Bei dem eigentlichen Lernvorgang wird diese Reihenfolge so oft durchlaufen, bis die gewünschte Anzahl an Lernschritten erreicht wurde. Durch diese Vereinfachung wird die Zeit für Speicherzugriffe und bei einem eventuellen Cache die Anzahl der Nachladezyklen minimiert. Gleichzeitig hält für diesen Fall aber keiner der auf Quantisierung oder Topologieerhaltung bezogenen Beweise in Abschnitt 2.1.

  Die Ergebnisse für eine entsprechende Kohonen-SOM bzw. Conscious-SOM-Implementierung finden sich in Abbildung A.1 (c) und (d). Es zeigt sich, dass sich für die betrachteten Datensätze ausschließlich positive Veränderungen hinsichtlich der Qualität ergeben.

- **Quadratische statt runde Nachbarschaftsfunktion:** Auch bei Verwendung eines Multiplizierers für die eigentliche Adaption, und der daraus resultierenden Möglichkeit zu *runden* Nachbarschaftsfunktionen, kann es sinnvoll sein, *quadratische* Nachbarschaftsfunktionen zu verwenden: Beim Adaptionsvorgang müssen die Adaptionswerte $\alpha h_{c,i}$, sofern sie nicht lokal in jedem Prozessorelement berechnet werden, über einen globalen Mechanismus verteilt werden. Unter der Voraussetzung, dass innerhalb eines Radius r um das Erregungszentrum[2] relevante Adaptionswerte ($\alpha h_{c,i} \neq 0$) existieren, müssen bei der rechteckigen Form mindestens $r+1$ und bei der runden Form mindestens r^2+1 Werte verteilt werden. Wenn alle Prozessorelemente an einem gemeinsamen Bus angeschlossen sind werden also pro Adaptionsschritt mindestens $r+1$ bzw. r^2+1 Übertragungen notwendig, was speziell für breite Nachbarschaftsfunktionen einen großen Geschwindigkeitsvorteil von r^2-r Übertragungen für $p1$ ergibt. Ein Spezialfall ergibt sich bei der Berechnung der Conscious-SOM, wie in Abschnitt 1.5.1 beschrieben. Hier kommen nur zwei Varianten der Nachbarschaftsfunktionen in Frage: Bei der

[2] $r = 0$ bezieht sich ausschließlich auf das Erregungszentrum

quadratischen Funktion werden alle acht direkten Nachbarn adaptiert, bei der rautenförmigen Funktion nur die horizontalen und vertikalen. Für alle praktischen Belange bezüglich der Latenz der Verteilung der Adaptionswerte fällt diese Unterscheidung nicht ins Gewicht; für alle CSOM Messungen wurde, falls nicht anders angegeben, die rautenförmige Nachbarschaftsfunktion verwendet, da diese in den Experimenten eine feinere Strukturierung der Karte gezeigt hat.

Diese Veränderungen bieten Vorteile bezüglich der Ressourcen Fläche, Energie und Zeit für eine spezialisierte Hardware Implementierung, gleichzeitig bedeuten sie aber auch Veränderungen gegenüber dem ursprünglichen SOM-Algorithmus, deren Auswirkungen überprüft werden müssen. Für diese Überprüfung kommen die gleichen Verfahren wie in Abschnitt 2.5 zum Einsatz. Die Ergebnisse dieser Untersuchungen sind im Anhang A.3 grafisch dargestellt, die Werte der w_{krit} sind in Tabelle 2.1 aufgetragen.

2.7 Zusammenfassung

In diesem Kapitel wurden hardwarespezifische Anpassungen des SOM-Algorithmus motiviert und auf funktionaler Ebene empirisch analysiert. Bei der Analyse wurden zusätzlich zu den aus der Literatur bekannten Qualitätskriterien ein neues Maß verwendet, das die Veränderungen des sog. Vergrößerungsexponenten anzeigt. Da die Qualitätskriterien z.T. widersprüchliche Aussagen bzgl. der Qualität einer Implementierung ergeben, wurde eine Metrik entwickelt, die auf Basis eines gewichteten Produktes der zuvor beschriebenen Kriterien ein skalares Maß für die Qualität einer SOM-Implementierung ergibt. Auf Basis dieser Metrik wurden die Anfangs beschriebenen Anpassungen anhand von Testdatensätzen überprüft und erläutert.

Bezüglich der Zahldarstellung zeigt sich, dass für jeden Datensatz eine spezifische Konstante w_{krit} existiert, die die Mindestanforderungen an die Wortbreite einer Hardwareimplementierung mit Festkommadarstellung beschreibt. Die absoluten Werte für w_{krit} sowie die durchschnittlichen Werte für Q oberhalb dieser Grenze sind in Tabelle 2.1 aufgeführt. Daraus ergeben sich die folgenden Schlüsse:

- **Eine Vergrößerung der Karte** wirkt sich sowohl bei der Conscious- als auch bei der Kohonen-SOM positiv auf $\bar{w}_{krit}$ aus, d.h. eine zu geringe Wortbreite bei einer Hardware-Implementierung kann unter Umständen durch eine Vergrößerung der Karte ausgeglichen werden. Gleichzeitig erhöht sich die Qualität der Karte um bis zu 20%. Dieses Ergebnis bestätigt Lightowler et al. in [85], der allerdings grundsätzlich davon ausgeht, dass 8 Bit Wortbreite bei einer entsprechend großen Karte ausreichend sind. Die Ergebnisse

in den hier gezeigten Untersuchungen zeigen, dass davon nicht ohne weiteres ausgegangen werden kann.

- **Eine Verlängerung der Lerndauer** zeigt während dessen nur wenig Auswirkung auf die Qualität der Karte und $\bar{w}_{krit}$ wird nicht beeinflusst.

- **Die Verwendung der $p1$-Norm** bringt in den vorliegenden Beispielen nur sehr geringe Abweichungen zur Referenz, was die Angaben aus [92] bestätigt. Auffällig ist der Trend bei dem sich besonders für Datensätze mit einer höheren Dimensionalität ein kleineres w_{krit} ergibt. In Verbindung mit den Ergebnissen aus [21] lässt sich vermuten, dass für Datensätzen mit einer sehr hohen Dimensionalität ($\gg 100$) die $p1$-Norm vorteilhaft sein kann.

- **Die Einführung einer wiederkehrenden Reihenfolge** für die Auswahl der Vektoren im Lernvorgang hat in den Experimenten keine negativen Auswirkungen auf den Lernvorgang, es wird im Gegenteil $\bar{w}_{krit}$ leicht erniedrigt und die Qualität $\bar{Q}$ für die Conscious-SOM leicht erhöht. Dies ist vermutlich darauf zurückzuführen, dass auf jeden Fall alle Vektoren gelernt werden, bei einer zufälligen Auswahl der Vektoren ist hierfür ein *guter* Zufallsgenerator von Nöten. Es bleibt zu bemerken, dass die Kardinalität des Datensatzes, also die Anzahl der Vektoren oder Datenpunkte vermutlich nicht irrelevant sind. Aus Sicht der Anwendung ist die Wiederholung eines kleinen Datensatzes in immer derselben Reihenfolge weniger als eine zufällige Auswahl zu sehen als die Wiederholung eines sehr großen Datensatzes. Für eine endgültige Beurteilung sind Ergebnisse mit sehr großen Datensätzen nötig.

- **Die Verwendung einer *rechteckigen* Nachbarschaftsfunktion** wirkt sich für beide SOM-Varianten unterschiedlich aus. Im Falle der Standard-Kohonen SOM ergeben sich nur leichte Unterschiede zur Referenz, die sich im Rahmen der statistischen Schwankungen bewegen. Für die Conscious-SOM ergeben sich deutlich negative Folgen, was über die sich ergebende Form der Nachbarschaftsfunktion erklärbar ist. Während standardmäßig eine rautenförmige Nachbarschaftsfunktion, also die kleinstmögliche symmetrische Nachbarschaftsfunktion verwendet wird, werden bei der quadratischen Nachbarschaftsfunktion alle Nachbarn des Erregungszentrums adaptiert. Dadurch ergibt sich zwischen benachbarten Erregungszentren eine größere Überlappung bei der Adaption. Während dieser Effekt bei der Kohonen-SOM unbedingt erwünscht ist, führt er bei der Conscious-SOM, bei der die Weite der Nachbarschaftsfunktion immer konstant bleibt, insbesondere bei der Quantisierung teilweise zu schlechteren Ergebnissen.

- **Die Approximation der Nachbarschaftsfunktion durch negative Zweierpotenzen** wird in Tabelle 2.1 nicht weiter diskutiert, da sich bereits in Abschnitt 2.3.3 zeigt, dass die Ergebnisse kaum mit der Referenz

vergleichbar sind. Sobald die topografieerhaltenden Eigenschaften der SOM im konkreten Fall wichtig sind, schließt sich die Verwendung dieser Variante ohne Multiplizierer aus. Dieses Ergebnis steht im Gegensatz zu [85], der davon ausgeht, dass ein Multiplizierer nicht notwendig ist.

Die Untersuchung der zusätzlichen hardwarespezifischen Anpassungen aus Abschnitt 2.6 zeigt also, dass die Implementierung eines Hardwarebeschleunigers auf Basis der genannten Anpassungen nur sehr begrenzte Nachteile gegenüber der Referenzimplementierung aufweist, sofern die Wortbreite w_{krit} für den zu lernenden Datensatz nicht unterschritten werden.

Bei der Betrachtung individueller Datensätze auf Ebene der einzelnen Qualitätskriterien zeigt sich, dass die hardwarespezifischen Anpassungen in den meisten Fällen eine *Veränderung* des Lernvorganges hervorrufen, die sich in einer Variation der Qualitätskriterien niederschlagen. Gleichzeitig muss festgestellt werden, dass die Richtung der Veränderung zu einer besseren oder schlechteren Qualität der Lernergebnisse mit den derzeit bekannten Methoden nicht vorhergesagt werden kann.

Variation	Original		125% Neuronen		doppelte Lerndauer		$p1$-Norm für $\lvert\vec{w}-\vec{m}\rvert$		wiederkehrende Reihenfolge		quadratische Nachbarschaftsfunktion	
SOM-Typ	K	C	K	C	K	C	K	C	K	C	K	C
Datensatz	w_{krit} für eine maximale Abweichung von 5%											
abalone	11	11	10	10	12	11	11	11	11	11	11	11
breast_cancer	11	9	10	8	11	9	11	10	11	9	12	9
balance_scale	8	9	7	8	8	9	8	9	8	8	8	9
blood_donator	12	10	11	9	12	10	12	10	12	10	12	10
echocardiogramm	11	8	14	6	11	7	10	7	8	6	8	12
glass	11	9	9	8	10	9	10	9	10	8	10	9
hill_valley	18	11	17	9	18	13	17	10	17	12	18	12
parkinson	10	9	10	8	10	8	10	9	12	8	11	9
∅	11,5	9,5	11	8,25	11,5	9,5	11,13	9,38	11,1	9	11,25	10,13
Datensatz	$\bar{Q}(w)$ mit $w > w_{krit}$ für eine maximale Abweichung von 5%											
abalone	1	1	1,1	1,1	0,99	1	1	0,97	1	1	1	0,97
breast_cancer	1	1	1,1	1,1	1	1	0,99	0,95	1	1	0.99	0,96
balance_scale	1	0,99	1,1	1,1	1,1	1,1	1,03	0,97	1,1	1	1,06	0,96
blood_donator	1	1	1,1	1,3	1,1	1,1	1	0,96	1	0,99	1	0,97
echocardiogramm	0,97	0,95	0,93	1,2	0,99	1,1	1,02	0,99	1,2	1,2	1,15	0,9
glass	1	1	1	1,1	1,1	1,1	1,02	1,04	1,1	1,1	1,04	1,03
hill_valley	1	0,99	1,1	1,1	0,99	1	1	0,98	1	1	1,97	0,95
parkinson	1	1	1	1,2	1,1	1,1	1	0,98	1	1,2	1,04	0,95
∅	1	0,99	1,1	1,2	1	1,1	1	0,98	1	1,1	1,03	0,96

Tabelle 2.1: Ergebnisse für Q bei der Variation der Lernparameter

Kapitel 3

Eingebettete SOM-Hardware

Die Ergebnisse aus Kapitel 2 zeigen, dass eine Implementierung des SOM-Algorithmus als massiv paralleler Hardware-Beschleuniger mit bestimmten Hardware-spezifischen Anpassungen möglich ist. Insbesondere ist hervorzuheben, dass nicht auf eine Gleitkomma-Zahldarstellung zurückgegriffen werden muss, solange die experimentell gezeigten Toleranzen bei einer Festkomma-Zahldarstellung im Einzelfall akzeptabel sind.

Im Folgenden wird untersucht, welche prinzipiellen Implementierungsvarianten für ein massiv paralleles SOM-Hardware System existieren, und welche sich hinsichtlich ihrer Ressourceneffizienz am besten für eine Implementierung in Hardware eignen. Dazu wird ein Modell zur Abschätzung des Ressourcenbedarfs der prinzipiellen Varianten auf Basis von prototypischen Implementierungen erstellt, das als Grundlage für die Bestimmung der Ressourceneffizienz dient. Um im Folgenden konkrete Implementierungen miteinander vergleichen zu können, wird ein realistisches Anwendungsszenario definiert, das vergleichsweise hohe Ansprüche an einen Beschleuniger stellt. Anhand dessen werden dann Bewertungen der in diesem Kapitel eingeführten, prinzipiellen Varianten durchgeführt. Abschließend werden weitere Vergleiche mit eingebetteten Prozessoren mit speziellen Hardware-Beschleunigern bzw. Instruktionssatzerweiterungen [19] durchgeführt.

Im Folgenden wird häufig der Begriff Prozessorelement (PE) verwendet. Dabei ist es wichtig, diesen nicht mit einem Neuron zu verwechseln, auch wenn in dem gleichen Zusammenhang die Begriffe neuronen- und komponentenparallel verwendet werden. Ein Prozessorelement ist eine elementare Recheneinheit, die die Funktionalität eines Neurons abbilden kann. Da einzelne Prozessorelemente auf unterschiedliche Arten zu funktionalen Gruppen zusammengefasst werden können, ist diese Abbildung nicht notwendigerweise $1:1$, d.h. ein Prozessorelement *kann* ein Neuron abbilden, genauso gut können aber auch mehrere Prozessorele-

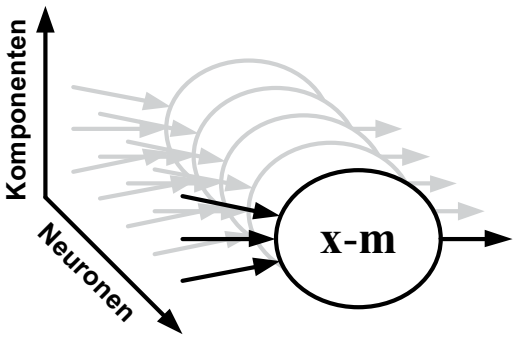


Abbildung 3.1: Bei der SOM sind zwei voneinander unabhängige Möglichkeiten zur Parallelisierung auf funktionaler Ebene zu berücksichtigen

mente zusammen ein Neuron bzw. ein Prozessorelement mehrere Neuronen abbilden. Um diese Fälle deutlich voneinander zu unterscheiden wird im Folgenden zunächst ausschließlich der physikalische Aufbau der Rechenelemente und deren Verschaltung angegeben. Dieser Aufbau definiert, welche Funktionen *gleichzeitig* bzw. *parallel* ausgeführt werden *können*, dadurch wird aber nicht eingeschränkt, wie die zu berechnende Karte aussehen soll. Dieser Aspekt wird im Abschnitt 3.2.3 behandelt.

3.1 Untersuchung der Realisierungsalternativen

Neben den Überlegungen zu den grundlegenden mathematischen Anforderungen an eine Realisierung des SOM-Algorithmus (siehe Kapitel 2) sind für eine massiv parallele Hardwareimplementierung grundsätzliche Überlegungen zum Aufbau der Prozessorelemente und deren Verschaltung notwendig. Da die Leistungsfähigkeit einer Implementierung im Wesentlichen davon abhängt, inwieweit sie die dem Algorithmus inhärente Parallelität abbildet, stellt die Untersuchung der gleichzeitig und unabhängig voneinander durchführbaren Operationen die Grundlage für die Entwicklung entsprechender Varianten dar. Die Untersuchung erfolgt hierarchisch, d.h. zunächst wird die Verschaltung potentieller Rechenelemente analysiert, anschließend folgt der interne Aufbau entsprechender Elemente.

Die Analyse der Verschaltung ergibt zwei prinzipielle Ansätze, die in Abbildung 3.1 in Anlehnung an das biologische Vorbild dargestellt werden. Da sich beide Ansätze (Neuronen- und Komponentenparallel) auch kombinieren lassen, ergeben sich insgesamt drei mögliche Varianten:

- **Neuronenparallel:** Die Berechnungen der l Neuronen einer SOM sind in weiten Teilen unabhängig voneinander und können parallel abgearbeitet werden. Der bei weitem größte Teil der Berechnungsdauer wird für *Abstandsberechnung* und *Suche des Erregungszentrums* (Gleichung 1.1) einer-

seits und *Adaption* (Gleichung 1.2) andererseits verwandt; der einzige Synchronisationspunkt zwischen diesen Funktionen liegt in der Suche und der Kommunikation des Erregungszentrums. Entsprechend können bei diesem Ansatz *bis zu* l Prozessorelemente parallel an der Berechnung der l Neuronen arbeiten. Die zu erwartende Steigerung der Berechnungsgeschwindigkeit wird dabei durch die nicht vollständig parallelisierbare Suche nach dem Erregungszentrum begrenzt. Abbildung 3.2 (a) zeigt den prinzipiellen Aufbau dieser Variante.

- **Komponentenparallel:** Genau wie die Berechnungen der Neuronen parallel ausgeführt werden können, eignen sich auch die Aufgaben bezogen auf die Komponenten für eine Parallelisierung. Dabei werden die Vektoroperationen in Gleichung 1.1 und 1.2, die beim neuronenparallelen Ansatz sequentiell berechnet werden, durch *bis zu* d_m Prozessorelemente parallel abgearbeitet. Entsprechend werden hier die Berechnungen pro Neuron sequentialisiert. Da in diesem Fall das Erregungszentrum sofort nach Abschluss der Berechnungen für alle Neuronen vorliegt, ist ein linearer Anstieg der Bearbeitungsgeschwindigkeit zu erwarten. Der schematische Aufbau dieser Implementierung wird in Abbildung 3.2(b) gezeigt.

- **Neuronen- und komponentenparallel:** Bei der Kombination beider Ansätze wird entlang beider *Achsen* parallelisiert um die maximale Beschleunigung zu erzielen. Es werden also *bis zu* d_m Prozessorelemente für die Berechnung eines Neurons zusammengefasst, die anschließend bis zu l-mal instantiiert werden, um auch alle Neuronen gleichzeitig berechnen zu können. Dadurch können also bis zu $l \cot d_m$ Prozessorelemente gleichzeitig an der Berechnung der Karte beteiligt werden. Der schematische Aufbau dieser Variante wird in Abbildung 3.2(c) dargestellt.

Ein entsprechendes Prozessorelement lässt sich auf verschiedene Arten realisieren, dabei sind insbesondere die Anforderungen an den Durchsatz und die verfügbaren Ressourcen zu berücksichtigen. In Abbildung 3.2(d) ist ein Prozessorelement dargestellt, das für alle vorgestellten Architekturvarianten verwendet werden kann, und sowohl für die Abstandsberechnung als auch für die Adaption jeweils ein Eingabedatum (eine Vektorkomponente) pro Takt verarbeiten kann. Dabei wurde die unter dieser Voraussetzung minimale Anzahl an arithmetischen Elementen verwendet, die Latenz des Prozessorelementes wird durch die Anzahl der Registerstufen im Datenpfad L_d bestimmt. Die verwendeten Speicherelemente dienen der Speicherung von Adaptionswerten (α-REG), Abstandswerten (Σ-REG) und Referenzvektorkomponenten (Cmp DP-RAM/REG).

Für eine konkrete Implementierung des Beschleunigersystems sind zwei wesentliche Punkte zu betrachten:

Steuereinheiten:

Neben dem in Abbildung 3.2 dargestellten Datenpfad, der in einer parallelen Implementierung n-fach instanziiert wird, ist für die Berechnung des SOM-Algorithmus ein entsprechender Kontrollpfad notwendig. Dieser Kontrollpfad beinhaltet sowohl lokale Komponenten, also solche, die für jedes einzelne Prozessorelement instanziiert werden müssen, als auch globale Komponenten, die nur einmal für das gesamte Prozessorfeld benötigt werden. Abhängig von der Verschaltungsvariante verschiebt sich der Anteil der lokalen und globalen Komponenten sehr deutlich und beeinflusst damit wesentlich den Ressourcenbedarf: während bei einer komponentenparallelen Implementierung der Datenpfad praktisch vollständig global gesteuert werden kann, da sich die Berechnungen der einzelnen Prozessorelemente ausschließlich durch die Daten- und Referenzvektorkomponenten unterscheiden, ist bei dem neuronenparallelen Ansatz ein hoher Anteil an lokalen Steuerungsmechanismen erforderlich. Der Grund dafür ist in den Parametern des Algorithmus zu suchen: während alle Parameter für die Komponenten jeweils eines Referenzvektors (insbesondere der Adaptionswert $h_{c,i}$) gleich sind, benötigen unterschiedliche Neuronen auch unterschiedliche Parameter. Entsprechend müssen für den neuronenparallelen Ansatz Mechanismen gefunden werden, die die Zuweisung individueller Parameter für jedes Neuron zulassen. Abbildung 3.2(d) zeigt einen Teil dieser Mechanismen in Form des α-Registers und des nachfolgenden Multiplexers, beide Elemente werden in dieser Form nur vom neuronenparallelen Ansatz benötigt. Zusätzlich werden alleine für die Realisierung dieser Funktionalität noch weitere Register, Komparatoren, Multiplexer und Zähler benötigt, die bei den komponentenparallelen Varianten entfallen können.

WTA Schaltung:

Die WTA Schaltung (engl. Winner Take All, der Gewinner bekommt alles) wird zur Bestimmung des Erregungszentrums bei der Abstandsberechnung benötigt. Dabei kommen je nach Implementierungsvariante verschiedene Verfahren in Frage, die sich ebenfalls auf den Ressourcenbedarf auswirken: für die neuronenparallelen Implementierungen kommen sowohl Komparatorbäume als auch ein Bitserieller Suchalgorithmus ([76], eine erweiterte Version wird in Abschnitt B.1 erläutert) in Frage. Bei den komponentenparallelen Implementierungen ist die Frage nach der Bestimmung des Minimums trivial, da auf jeden Fall eine Addiererschaltung benötigt wird, die die Distanzen der einzelnen Komponenten aufaddiert. Am Ausgang der Addiererschaltung wird in jedem Takt ein Abstandswert ausgegeben, von denen dann das Minimum bestimmt werden muss. Im Folgenden wird kurz zusammenfasst, hinsichtlich welcher Systemkomponenten sich die Implementierung der Prozessorelemente für die Varianten des Gesamtsystems unterscheiden:

- **Neuronenparallel mit Bitserieller Suche:** Bei dieser Variante wird eine

Schaltung wie in Abbildung B.1 benötigt, für die das Summenregister jedes einzelnen Prozessorelementes um eine Funktion zum Schieben der Daten erweitert werden muss. Zusätzlich werden einige Register und kombinatorische Logik für jedes einzelne Prozessorelement nötig.

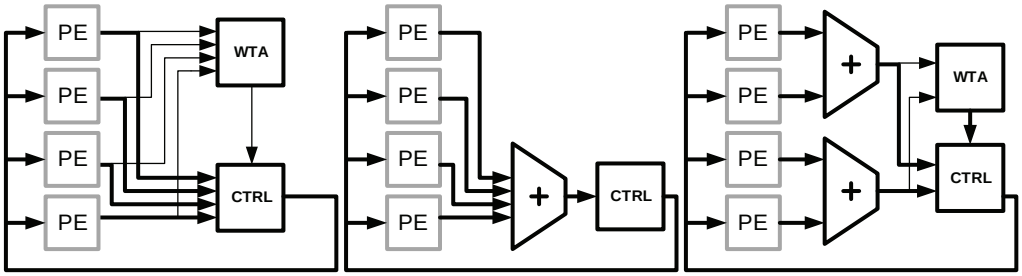
- **Neuronenparallel mit Komparatorbaum:** In diesem Fall entfällt die oben beschriebene Logik, stattdessen werden die PE_n Summenregister über einen Komparatorbaum mit $PE-1$ Komparatoren verknüpft. Der Komparatorbaum muss derart konstruiert werden, dass der Ursprung des Minimums identifiziert werden kann. Zusätzlich kommen ggf. Register zum Einsatz, um die maximale Taktfrequenz des Baumes zu erhöhen.

- **Komponentenparallel:** Da die Ergebnisse der PE_k Prozessorelemente aufsummiert werden müssen, um den Abstand zwischen Referenz- und Datenvektor zu bestimmen, kommt hier ein Addiererbaum mit $PE_k - 1$ Addierern zum Einsatz. Die Suche nach dem Minimum reduziert sich hier auf eine Vergleichsoperation; wie oben kommen ggf. zusätzliche Register zum Einsatz, um die maximale Taktfrequenz des Baumes zu erhöhen.

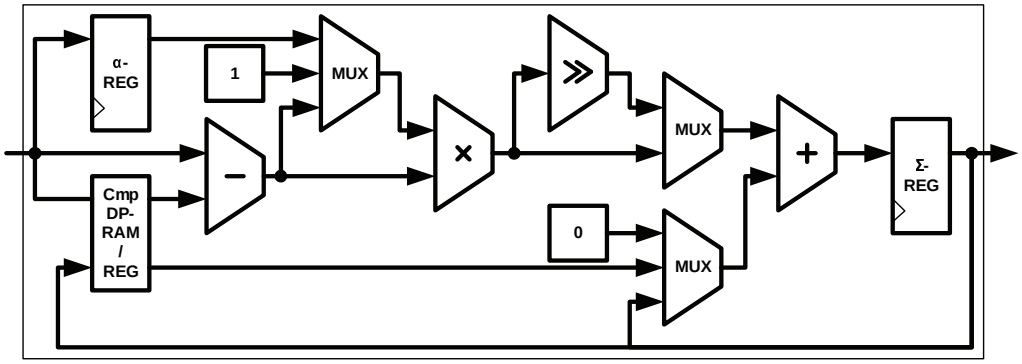
Der Ressourcenbedarf der oben beschriebenen WTA-Schaltungen und Steuereinheiten ist für die Implementierungsvarianten gegenläufig: während beim neuronenparallelen Ansatz eine sehr flächensparende Suche nach dem Erregungszentrum möglich ist, werden größere Anteile an lokalen Steuereinheiten benötigt. Für den komponentenparallelen Ansatz ist dieses Verhältnis umgekehrt, womit sich ein nicht trivialer Entwurfsraum aufspannt, der im Folgenden untersucht werden soll.

3.1.1 Bestimmung von Flächen, Leistung und Latenz

Unabhängig von der Definition der Ressourceneffizienz müssen die Größen Fläche (A), Leistungsaufnahme (P) und Latenz (T) in Abhängigkeit der Implementierungsvariante, der Anzahl der Prozessorelemente und der Lernparameter (Dimension des Datensatzes, Anzahl der Wiederholungen, Weite der Nachbarschaftsfunktion) bestimmt werden. Während die Latenz (in Form der Anzahl der benötigten Taktzyklen) für jede Implementierung direkt bestimmt werden kann, müssen sowohl für die maximale Taktfrequenz, die Leistungsaufnahme und für die Fläche konkrete Implementierungen untersucht werden. Dazu werden die Varianten als generische Entwürfe (die Anzahl der Prozessorelemente, Wortbreite und Speichertiefe können über sog. *Generics* angegeben werden) in VHDL beschrieben und synthetisiert. Da der Synthesevorgang sehr zeitaufwändig ist, wird dieser nur für eine bestimmte Anzahl von Punkten im Entwurfsraum durchgeführt, um dann zu interpolieren. Auf Basis dieser Interpolationen entsteht ein sehr präzises Modell für die Ressourceneffizienz.



(a) neuronenparallel (b) komponentenparallel (c) komp.- und neuronenpar.



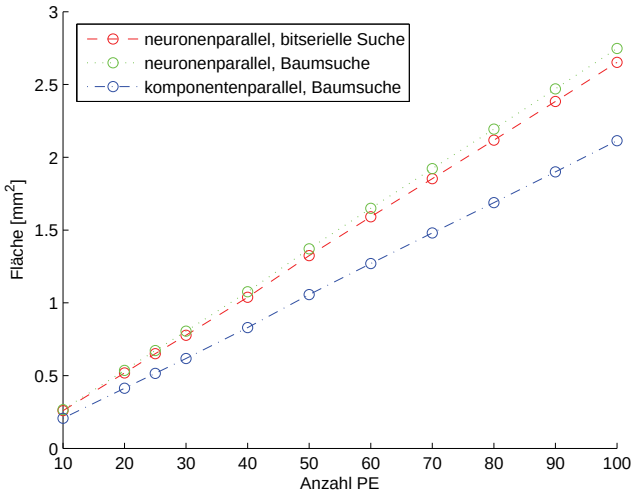
(d) Architektur eines universellen Prozessorelementes für SOM, dass ein Datum pro Takt verarbeiten kann; die komplette Steuerlogik wurde nicht dargestellt.

Abbildung 3.2: Mögliche Architekturen für eine massiv parallele SOM-Implementierung unter der Voraussetzung, dass pro Takt ein Datenwort pro paralleler Einheit verarbeitet werden muss. Abbildung (a) zeigt die neuronenparallele Implementierung, bei der jedes Prozessorelement die Berechnung eines oder mehrerer Neuronen übernimmt. Abbildung (b) zeigt den komponentenparallelen Ansatz, bei dem jedes Prozessorelement die Berechnung einer oder mehrerer Komponenten genau eines Neurons übernimmt. In Abbildung (c) wird der gemischt neuronen- und komponentenparallele Ansatz gezeigt, bei dem jeweils zwei komponentenparallel arbeitende Prozessorelemente zu einem Neuron zusammengefasst werden.

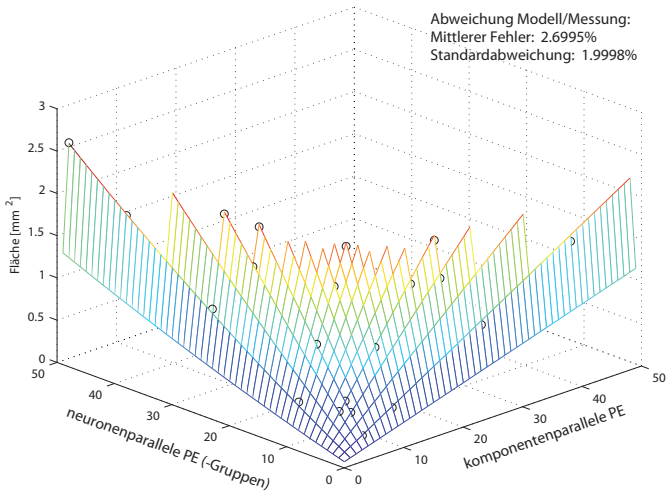
Flächenbedarf: A

Bei der Bestimmung des Flächenbedarfes wurden die oben beschriebenen Architekturvarianten in einer 90 nm Technologie von UMC synthetisiert. In Abbildung 3.3 wird dieser Flächenbedarf *ohne* die benötigten RAM-Speicherelemente dargestellt, um die Unterschiede zwischen den Varianten deutlicher hervorzuheben.

Es zeigt sich, dass sich bei den rein komponenten- *oder* neuronenparallelen Implementierungen für den Flächenbedarf ein linearer Zusammenhang der Form $a_1 PE_{nk} + a_0$ mit der Anzahl der Elemente PE_{nk} ergibt, a_0 und a_1 sind abhängig von der jeweiligen Architektur. Für den komponenten- *und* neuronenparallelen Ansatz ergibt sich ein Zusammenhang der Form $PE_k(a_0 + a_1 PE_n + a_2 PE_n^2)$, es



(a) Komponenten- **oder** neuronenparallel, siehe Abbildung 3.2 (a) und (b)



(b) Neuronen- **und** komponentenparallel, siehe Abbildung 3.2 (c)

Abbildung 3.3: Flächenbedarf der Implementierungsvarianten bezogen auf die Anzahl der Prozessorelemente ohne Speicher. Die Kreise stellen die Syntheseergebnisse dar. Zum besseren Verständnis dieser 3D-Darstellung werden in Tabelle 3.1 einige Werte aufgetragen.

ergibt sich also ein zusätzlicher quadratischer Term für die Neuronen PE_n in der Steigung für die Komponenten PE_k. Im Vergleich benötigen der rein komponentenparallele und der gemischt-parallele Ansatz die geringste Fläche. Bei den neuronenparallelen Ansätzen ist der bitserielle Ansatz wie erwartet kleiner als der Komparatorbaum.

$PE_n \times PE_k$	$A\ [mm^2]$	$PE_n \times PE_k$	$A\ [mm^2]$	$PE_n \times PE_k$	$A\ [mm^2]$
2×50	2,174	2×25	1,087	2×10	0,435
4×25	2,145	5×10	1,073	4×5	0,429
25×4	2,307	10×5	1,087	5×4	0,429
50×2	2,543	25×2	1,153	10×2	0,435

Tabelle 3.1: Ausgewählte Werte aus Abbildung 3.3(b)

$PE_n \times PE_k$	$P\ [W]$	$PE_n \times PE_k$	$P\ [W]$	$PE_n \times PE_k$	$P\ [W]$
2×50	0,290	2×25	0,145	2×10	0,058
4×25	0,288	5×10	0,144	4×5	0,058
25×4	0,311	10×5	0,147	5×4	0,058
50×2	0,344	25×2	0,156	10×2	0,059

Tabelle 3.2: Ausgewählte Werte aus Abbildung 3.4(b)

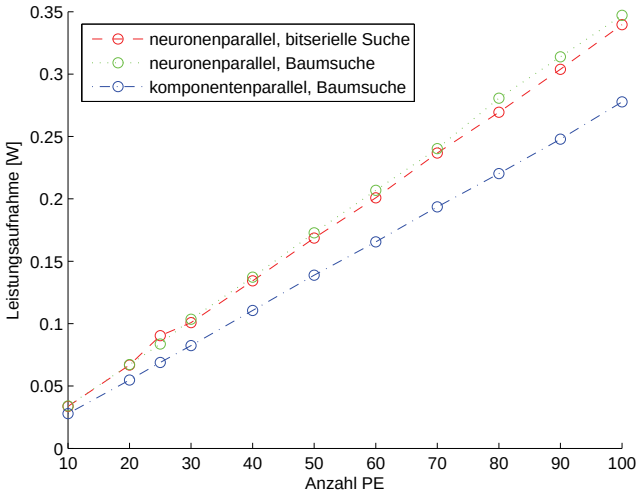
Leistungsaufnahme: P

Auch die Leistungsaufnahme der Implementierungsvarianten steigt nahezu linear mit der Anzahl der implementierten Elemente an, die Verhältnisse untereinander entsprechenden denen des Flächenbedarfs. Da diese Darstellung ebenfalls auf Synthesewerten beruht, muss bei der Interpretation beachtet werden, dass hier Leitungslängen nur sehr ungenau modelliert wurden, und dass die Werte auf Basis von fixen Schaltwahrscheinlichkeiten (50%) der Eingänge angenommen wurden. Da die Verwendung von Schaltwahrscheinlichkeiten den tatsächlichen Einsatz nur sehr ungenau abbilden, sind entsprechend auch die absoluten Werte sehr ungenau. Für einen Vergleich zwischen verschiedenen auf diese Weise abgeschätzten Implementierungen sind die Werte allerdings gut geeignet, da die durchzuführenden Berechnungen verschiedener Implementierungen bis auf wenige Ausnahmen[1] gleich sind.

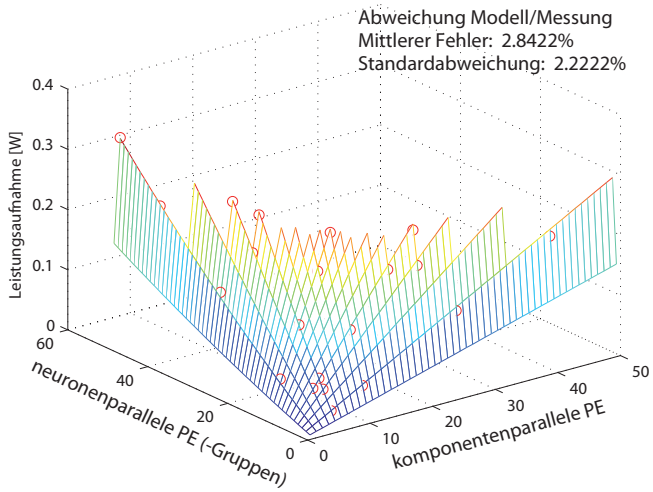
Die Approximation der Leistungsaufnahme für das Modell ergibt sich analog zum Flächenbedarf: die rein komponenten- *oder* neuronenparallelen Ansätze werden über $P = p_1 PE_{nk} + p_0$ abgeschätzt, bei dem gemischtparallelen Ansatz ergibt sich $P = PE_k(p_0 + p_1 PE_n + p_2 PE_n^2)$.

Für sich betrachtet ergibt die Leistungsaufnahme weitere Informationen für die Realisierung als ASIC, da hier abgeschätzt werden kann, wie die Spannungsversorgung des eigentlichen Rechenkerns ausgelegt werden muss. Bei der Bewertung einer konkreten Implementierung hinsichtlich ihrer Ressourceneffizienz ist zusätz-

[1] Bei dem neuronenparallelen Ansatz wird die Position des Neurons im Gitter lokal gespeichert, bei den komponentenparallelen nicht. Entsprechend werden alle darauf bezogenen Berechnungen, maßgeblich die Abstandsberechnung bei der Adaption, lokal bzw. global durchgeführt. Da diese Berechnungen für relevante Datensätze einen sehr geringen Anteil an der Gesamtberechnung haben, können sie hier vernachlässigt werden.



(a) Komponenten- **oder** neuronenparallel, siehe Abbildung 3.2 (a) und (b)



(b) Neuronen- **und** komponentenparallel, siehe Abbildung 3.2 (c)

Abbildung 3.4: Leistungsaufnahme der Implementierungsvarianten bezogen auf die Anzahl der Prozessorelemente. Die Kreise stellen die Syntheseergebnisse dar. Zum besseren Verständnis dieser 3D-Darstellung werden in Tabelle 3.2 einige Werte aufgetragen.

lich die Berechnungsdauer (Latenz) T von Interesse, da erst in Verbindung mit T der Energiebedarf bezogen auf eine Berechnungsaufgabe abgeschätzt werden kann. Wenn zusätzlich eine Zeitschranke vorgegeben wird, innerhalb derer eine Berechnungsaufgabe komplettiert werden muss, können erweiterte Energiesparmaßnahmen abgeschätzt werden. Im folgenden Abschnitt wird daher zunächst die

$PE_n \times PE_k$	$L\ [s]$	$PE_n \times PE_k$	$L\ [s]$	$PE_n \times PE_k$	$L\ [s]$
2×50	4,16	2×25	4,16	2×10	10,56
4×25	2,56	5×10	4,80	4×5	8,96
25×4	2,92	10×5	4,16	5×4	9,92
50×2	3,04	25×2	4,60	10×2	9,28

Tabelle 3.3: Ausgewählte Werte aus Abbildung 3.5(b)

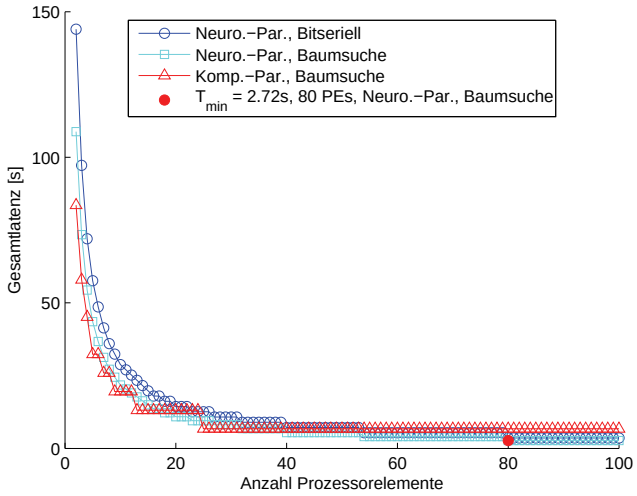
Latenz berechnet, anschließend wird beispielhaft der Energiebedarf mit skalierter Versorgungsspannung bestimmt.

Berechnungsdauer: T

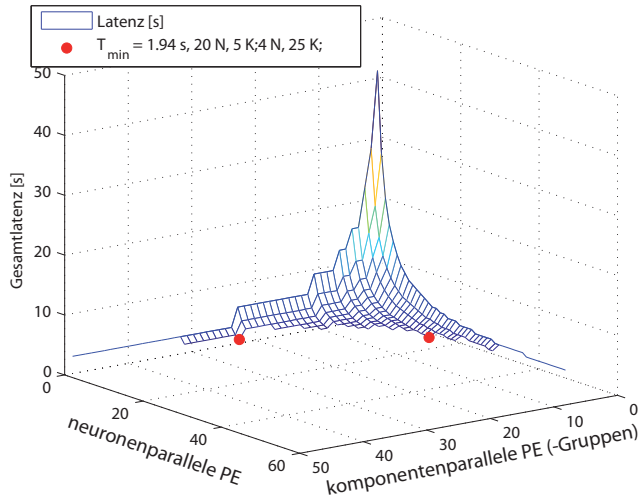
Die minimale Berechnungsdauer T (in diesem Zusammenhang auch als Latenz L bezeichnet) lässt sich für die einzelnen Implementierungen exakt in Form der Anzahl der benötigten Takte bestimmen. Dazu wird der Berechnungsvorgang in die drei Phasen *Initialisierung*, *Berechnung* und *Ausgabe* unterteilt, wobei die Berechnung in der Regel den größten Zeitanteil benötigt und in die Abschnitte *Abstandsberechnung*, *Suche des Erregungszentrums* und *Adaption* unterteilt wird. Tabelle 3.4 zeigt die einzelnen Komponenten für jede Architekturvariante.

Dabei beschreiben die $\lceil \frac{l}{PE_n} \rceil$ und $\lceil \frac{d_m}{PE_k} \rceil$ Terme jeweils den Grad der Virtualisierung, der immer dann einen Wert größer eins einnimmt, wenn die Anzahl der physikalisch verfügbaren Prozessorelemente kleiner ist, als die der berechneten Elemente (Neuronen oder Komponenten). Entsprechend beschreiben PE_N und PE_k die Anzahlen der Prozessorelemente, die für die Berechnung der Neuronen bzw. der Komponenten pro Neuron verwendet werden. Über den Aufbau und die Verschaltung der Elemente ergibt sich L_d als Latenz eines Prozessorelementes und L_a bzw. L_k die Latenz des Addierer- bzw. Komparatorbaumes. r_h entspricht der Anzahl der Elemente der Adaptionsfunktion, die im neuronenparallelen Fall sequentiell an die Prozessorelemente kommuniziert werden müssen. Eine Besonderheit ergibt sich bei der Suche nach dem Erregungszentrum im neuronenparallelen Fall ohne Komparatorbaum: w_s, die Anzahl der Bits im Summenregister ergeben die Latenz der WTA-Komponente, $\lceil \frac{w_s}{w_b} \rceil + 3$ beschreibt den Auslesevorgang, bei dem sowohl die Position des Neurons im Netzwerk als auch der Abstandswert des Neurons über einen w_b Bit breiten Datenbus übertragen werden müssen.

Abbildung 3.5 zeigt konkrete Werte für die Berechnungsdauer; als Taktfrequenz werden hier die in der Synthese vorgegebenen $F = 500$ MHz angenommen. Zunächst wird deutlich, dass der gemischt neuronen- und komponentenparallele Ansatz mit nahezu allen Konfigurationen am schnellsten ist. Ein weiterer Aspekt ist der nicht stetige Verlauf der Berechnungsdauer mit der Anzahl der Prozessorelemente. Dies ist dadurch zu begründen, dass nicht alle Prozessoren zu jeder Zeit an der Berechnung beteiligt werden können, weil die Anzahl an zu berechnenden



(a) Komponenten- **oder** neuronenparallel, siehe Abbildung 3.2 (a) und (b)



(b) Neuronen- **und** komponentenparallel, siehe Abbildung 3.2 (c)

Abbildung 3.5: Gesamtlatenz der Implementierungsvarianten bezogen auf die Anzahl der Prozessorelemente bei einem Datensatz mit 25 Komponenten der auf 160 Neuronen in 10^7 Lernschritten gelernt wird. Zum besseren Verständnis dieser 3D-Darstellung werden in Tabelle 3.3 einige Werte aufgetragen.

Elementen nicht restlos durch die Anzahl der berechnenden Komponenten teilbar ist. Da die Größe dieses Restes mit einer wachsenden Anzahl an Prozessoren zunimmt, sind auch die Stufen in Abbildung 3.5 größer. Dies erklärt auch, warum die Stufen bei dem gemischt-parallelen Ansatz, besonders in der Diagonale der xy-Ebene, verhältnismäßig klein sind: die Anzahl der Elemente entlang der beiden

Architektur	Initialisierung	Auslesen
N-p. BS	$\lceil \frac{l}{PE_n} \rceil (d_m + 2 + PE_n(2 + L_d))$	$PE_n \lceil \frac{l}{PE_n} \rceil (d_m + 2)$
N-p. KB	$\lceil \frac{l}{PE_n} \rceil (d_m + 2 + PE_n(2 + L_d))$	$PE_n \lceil \frac{l}{PE_n} \rceil (d_m + 2)$
K-p.	$\lceil \frac{d_m}{PE_k} \rceil l$	$PE_k \lceil \frac{d_m}{PE_k} \rceil l$
N-K-p.	$\lceil \frac{d_m}{PE_k} \rceil \lceil \frac{l}{PE_n} \rceil$	$\lceil \frac{d_m}{PE_k} \rceil d_m \lceil \frac{l}{PE_n} \rceil l$

Architektur	Abstandsberechnung	Suche Erregungszentrum
N-p. BS	$\lvert X \rvert \lceil \frac{l}{PE_n} \rceil (1 + d_m + L_d)$	$\lvert X \rvert (\lceil \frac{l}{PE_n} \rceil (3 + w_s + \lceil \frac{w_s}{w_b} \rceil))$
N-p. KB	$\lvert X \rvert \lceil \frac{l}{PE_n} \rceil (1 + d_m + L_d)$	$\lvert X \rvert \lceil \frac{l}{PE_n} \rceil (1 + L_k)$
K-p.	$\lvert X \rvert \lceil \frac{d_m}{PE_k} \rceil (1 + PE_n + L_d)$	$\lvert X \rvert \lceil \frac{d_m}{PE_k} \rceil (1 + L_a)$
N-K-p.	$\lvert X \rvert \lceil \frac{d_m}{PE_k} \rceil \lceil \frac{l}{PE_n} \rceil (1 + L_d)$	$\lvert X \rvert \lceil \frac{d_m}{PE_k} \rceil \lceil \frac{l}{PE_n} \rceil (1 + L_k + L_a)$

Architektur	Adaption	
N-p. BS	$\lvert X \rvert \lceil \frac{l}{PE_n} \rceil (1 + d_m + L_d + r_h)$	
N-p. KB	$\lvert X \rvert \lceil \frac{l}{PE_n} \rceil (1 + d_m + L_d + r_h)$	
K-p.	$\lvert X \rvert \lceil \frac{d_m}{PE_k} \rceil l$	
N-K-p.	$\lvert X \rvert \lceil \frac{d_m}{PE_k} \rceil \lceil \frac{l}{PE_n} \rceil + L_d$	

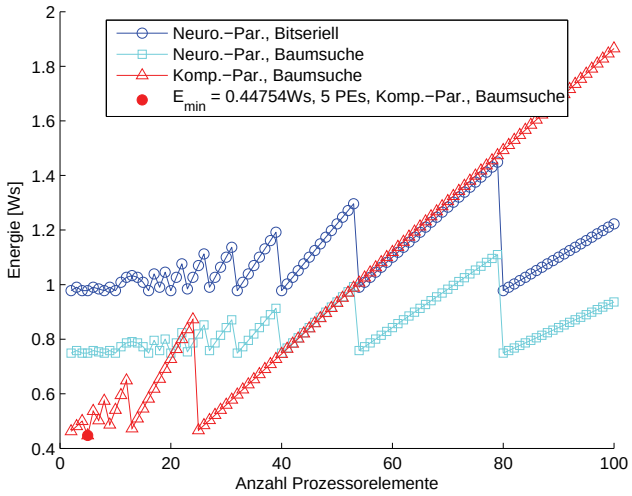
Tabelle 3.4: Berechnung der Gesamtlatenz eines Hardwarebeschleunigers mit verschiedenen Architekturen. **Abkürzungen:** N-p. BS: Neuronenparallel mit bitserieller Suche, N-p. KB: Neuronenparallel mit Komparatorbaum, K-p.: Komponentenparallel, N-K-p. Neuronen- und komponentenparallel.
Die einzelnen Beiträge sind ohne Optimierungen aufgeführt: in [106] wird berechnet, dass im Durchschnitt nur 1/3 aller Neuronen adaptiert werden. Wird dies beim Entwurf der Steuerungslogik berücksichtigt, kann entsprechend bis zu 2/3 der Zeit für die Adaption gespart werden, siehe Abschnitt 3.4.6.

Parallelitätsgrade ist verhältnismäßig geringer.

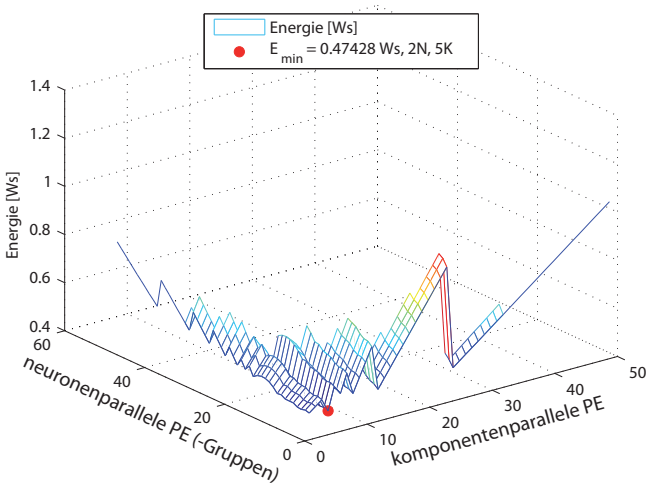
Mit dieser Berechnung der Berechnungsdauer ist es möglich, eine Betrachtung des Energiebedarfes der einzelnen Varianten zu erstellen. Dabei können auch Zeitschranken und dementsprechend Veränderungen der Taktrate und der Versorgungsspannungen untersucht werden.

Energiebedarf mit skalierter Versorgungsspannung

Im Gegensatz zu einem prozessorbasierten System zur Berechnung des SOM-Algorithmus kann der dedizierte Hardwarebeschleuniger ausschließlich für die Berechnung der SOM eingesetzt werden. Daher kann Rechenleistung, die nicht für die Berechnung der SOM benötigt wird, etwa weil die Datenvektoren nicht mit der maximal möglichen Datenrate übertragen werden, für keine anderen Aufgaben zur Verfügung gestellt werden. Um in diesem Fall die geringste mögliche Menge an Energie umzusetzen, kann die Rechenleistung der Hardware über die Versorgungs-



(a) Komponenten- **oder** neuronenparallel



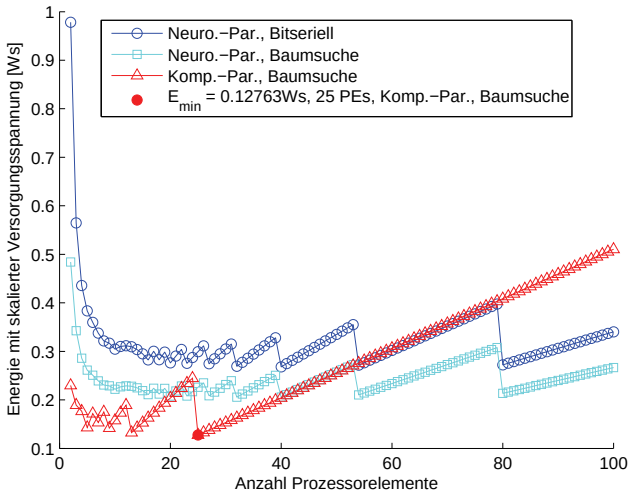
(b) Neuronen- **und** komponentenparallel

Abbildung 3.6: Energieaufnahme bezogen auf die Anzahl der Prozessorelemente bei einem Datensatz mit 25 Komponenten der auf 160 Neuronen in 10^7 Lernschritten gelernt wird.

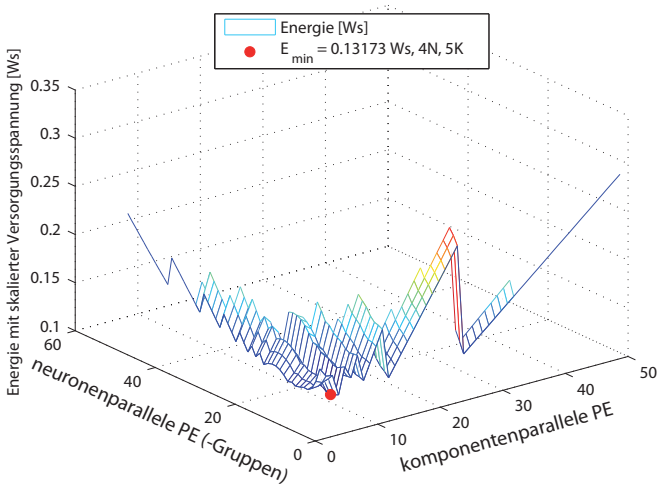
spannung in einem gewissen Rahmen an die Anforderungen angepasst werden. So hängen nach [98, 117] Versorgungsspannung U_{DD} und minimale Periodendauer t_p näherungsweise über

$$t_p \propto \frac{U_{\mathrm{DD}}}{(U_{\mathrm{DD}} - U_{\mathrm{th}})^{\eta}} \tag{3.1}$$

zusammen, wobei die Schwellspannung U_{Th} sowie K und $\eta \approx 1$ technologieabhängige Konstanten und C_L die Lastkapazität am Ausgang des Gatters be-



(a) Komponenten- **oder** neuronenparallel



(b) Neuronen- **und** komponentenparallel

Abbildung 3.7: Energieaufnahme mit skalierter Versorgungsspannung bezogen auf die Anzahl der Prozessorelemente bei einem Datensatz mit 25 Komponenten der auf 160 Neuronen in 10^7 Lernschritten gelernt wird.

schreiben. Wird nun also eine Zeitschranke t_{max} angenommen, innerhalb derer ein Lernvorgang abgeschlossen werden soll, so kann über eine Skalierung der Versorgungsspannung die Energieaufnahme deutlich beeinflusst werden. In Abbildung 3.6 werden die Energieaufnahmen für eine Berechnung ohne, in 3.7 mit Skalierung der Versorgungsspannung dargestellt. Dabei wird ein Datensatz mit 25 Komponenten auf 160 Neuronen mit 10^7 Lernschritten zugrundegelegt, und die Latenz einer neuronenparallelen Implementierung mit bitserieller Suche und 2 Neuronen

(also eine Minimalkonfiguration) als Zeitschranke eingesetzt, so dass für alle anderen Varianten eine Skalierung der Versorgungsspannung erfolgen kann. Deutlich sind die Spitzen im Energieumsatz zu erkennen, die immer dann auftreten, wenn besonders viele Prozessorelemente nicht zur Berechnung des Algorithmus beitragen können. Die Leistungsaufnahme $P_{\varnothing}$, die nicht zur Berechnung des Algorithmus beiträgt bestimmt sich zu:

$$P_{\varnothing} \propto \frac{d_m \bmod PE_k}{d_m} + \frac{l \bmod PE_n}{l} \tag{3.2}$$

Ebenso zeigt sich, dass der Energiebedarf für eine Implementierung mit mehr Prozessorelementen durch den erhöhten Verwaltungsaufwand eher steigt. Erst wenn unter Berücksichtigung der Zeitschranken die Frequenz F und die Versorgungsspannung U_{DD} abgesenkt werden, kann durch eine wachsende Anzahl an Prozessorelementen Energie gespart werden. Abbildung 3.7 zeigt aber auch, dass ein Optimum für die Anzahl der Prozessorelemente existiert, über das hinaus die benötigte Gesamtenergie wieder steigt.

Die Absenkung der Versorgungsspannung stellt also eine wichtige Möglichkeit für die Reduktion der Energieaufnahme dar. Um für das oben beschriebene Beispiel die Skalierung durchführen zu können, muss die Versorgungsspannung auf bis zu ca. 105% der nominalen Schwellspannung reduziert werden. In diesem Bereich ist es notwendig, weitere Techniken wie die dynamische Anpassung der Schwellspannung [96] zu nutzen. Dabei kommt es zu einer Erhöhung der Leckströme, die hier nicht weiter diskutiert wird. Zusätzlich kommt das teilweise Abschalten der Versorgungsspannung in Frage. Bei der Auswertung der Ressourceneffizienz wird die Absenkung der Versorgungsspannung gesondert berücksichtigt werden.

3.1.2 Ressourceneffizienz

Im vorangegangenen Abschnitt wurden zunächst die Basisgrößen A, T und P für verschiedene Implementierungsvarianten eines SOM-Hardwarebeschleunigers auf Basis von Syntheseergebnissen ermittelt. Anhand dieser Messgrößen wurde ein Modell erzeugt, dass im Bereich von $l \leq 100$ Prozessorelemente Interpolationen mit einem Fehler $\leq 2,5\%$ ermöglicht, d.h. für ein gegebenes Szenario kann der komplette Entwurfsraum für Systeme mit bis zu 100 Prozessorelementen aufgespannt und untersucht werden. Für die Auswahl einer der Implementierungen wird ein Kriterium benötigt, das die Punkte im Entwurfsraum bewertet. Hier kommt unter Anderem die sog. Ressourceneffizienz in Frage, die im Folgenden definiert werden soll. Da eine große Anzahl von Punkten im Entwurfsraum in keinerlei Hinsicht optimal sind, werden diese vor der Berechnung der Ressourceneffizienz über die Bestimmung der Paretomenge ausgeschlossen.

Bei der Bestimmung der Paretomenge werden diejenigen Punkte aus dem Explorationsraum bestimmt, die nicht von anderen dominiert werden, d.h. für

jeden Punkt in der Pareto Menge gibt es keinen anderen, der hinsichtlich aller Zielkriterien (A,P,T) *besser* ist. Da die Paretomenge in allen nicht-trivialen Fällen mehr als einen Punkt enthält, gilt es, ein Auswahlkriterium hierfür zu finden. Ein solches Kriterium kann die Ressourceneffizienz RE sein:

$$RE_i^{-1} = \left(\frac{\tilde{A}}{A_i}\right)^{\gamma_A} \left(\frac{\tilde{T}}{T_i}\right)^{\gamma_L} \left(\frac{\tilde{P}}{P_i}\right)^{\gamma_P} \tag{3.3}$$

Dabei stellen die $\tilde{A}, \tilde{P}, \tilde{T}$ mögliche Normierungsfaktoren dar, etwa das jeweilige Minimum aus dem Entwurfsraum, die γ dienen der Gewichtung der Teilprodukte. Durch eine geeignete Wahl der γ lassen sich weitere Maße, wie etwa das sog. *Power-Delay-Produkt* ($\gamma_A = 0$, $\gamma_P = 1$, $\gamma_L = 1$) ableiten. Da die Ressourceneffizienz von der Latenz T abhängig ist, die wiederum von der Architektur *und* von der Dimension des Datensatzes abhängt, muss die Ressourceneffizienz für jeden Datensatz einzeln untersucht werden. In Abschnitt 3.4 wird die Ressourceneffizienz beispielhaft für eine Anwendung aus der Luft- und Raumfahrttechnik bestimmt und ausgewertet. Dabei werden zusätzlich zu den hier beschriebenen Logikeinheiten auch die benötigten Steuer- und Speicherelemente einbezogen.

Ein wichtiger Punkt, der an dieser Stelle nicht berücksichtigt werden kann, ist die Bandbreite, mit der die Daten an das Hardwaremodul herangeführt werden. Diese kann entscheidend für die Auswahl einer der Architekturen sein, da die Anzahl der Ein- und Ausgänge (und damit u. A. die Fläche des ASICs) wesentlich von diesen Datenschnittstellen abhängt. Die Auswahl einer solchen Schnittstelle ist nur im Kontext mit einer konkreten Systemumgebung zu treffen, und kann daher hier nicht erfolgen. An Stelle der Auswahl und der dadurch notwendigen Abschätzungen zu Chipfläche, Latenz und Leistungsaufnahme wird hier der Bandbreitenbedarf der verschiedenen Architekturen angegeben; die maximale Bandbreite B_{max} des Neuronen-parallelen Ansatzes wird als 1 angenommen:

- Neuronenparallel, bitserielle Suche: $B_{max,\text{n-par a}} = 1$
- Neuronenparallel, Komparatorbaum: $B_{max,\text{n-par b}} = 1$
- Komponentenparallel, Addiererbaum:$B_{max,\text{k-par}} = PE_k$
- Komponenten- und neuronenparallel: $B_{max,\text{kn-par}} = PE_k$

Die Anzahl der gleichzeitig benötigten Komponenten hängt von der Anzahl der komponentenparallel arbeitenden Prozessorelemente ab, demnach benötigt bei der gleichen Anzahl an Prozessorelementen der komponentenparallele Ansatz die größte Bandbreite, gefolgt von dem Komponenten- und neuronenparallelen Ansatz. Allein die neuronenparallelen Architekturen haben eine Bandbreite unabhängig von der Anzahl der Prozessorelemente, daher sind diese in vielen Fällen den übrigen Implementierungen vorzuziehen.

Für das Anwendungsbeispiel in Abschnitt 3.4 gilt $B_{max} = 0,487$ Datenworte pro Systemtakt, was bei zwei 16 Bit Wortbreite und einem Systemtakt von 50 MHz einer Bandbreite von 48 MByte/s entspricht. Bei einer komponentenparallelen Implementierung steigt die benötigte Bandbreite auf 9,5 GByte/s an.

3.1.3 Zusammenfassung

In diesem Abschnitt wurden die prinzipiellen Realisierungsalternativen für einen massiv parallelen Beschleuniger für selbst-organisierende Karten untersucht. Dabei wurden anhand von prototypischen Implementierungen des Prozessorfeldes *ohne* Steuerung die Größen Fläche A, Leistungsaufnahme P und Latenz T bestimmt und modelliert. Anhand dieses Modells wurden dann die Auswirkungen von Energiesparmaßnahmen (exemplarisch am Beispiel der adaptiven Versorgungsspannung) und die Ressourceneffizienz ermittelt.

Es zeigt sich, dass in vielen Fällen der gemischt neuronen- *und* komponentenparallele Ansatz die beste Alternative darstellt, da dieser sowohl einen geringen Flächenbedarf als auch eine geringe Leistungsaufnahme aufweist. Gleichzeitig ist dieser Ansatz besonders flexibel, da die Anzahl der nicht genutzten Prozessorelemente im Vergleich zu den anderen Varianten gering ist. Die Nachteile dieser Architektur liegen in der u.U. sehr hohen benötigten Bandbreite und der vergleichsweise komplexen Steuerung. Um diese Punkte zu quantifizieren, wird im nächsten Abschnitt eine vollständige prototypische Implementierung dieser Variante, sowie der neuronenparallelen Variante mit bitserieller Suche durchgeführt.

3.2 Implementierung

Im vorangegangenen Abschnitt wurde gezeigt, dass unter den betrachteten Gesichtspunkten der neuronen- und komponentenparallele Ansatz in weiten Teilen die größte Ressourceneffizienz besitzt. Da in diese Betrachtung noch keine Ergebnisse zur Größe der Kontrolllogik eingegangen sind, werden im Folgenden dieser und der neuronenparallele Ansatz detailliert beschrieben, und anschließend miteinander verglichen. Dabei werden die Prozessorfelder und die Kommunikationslogik plattformunabhängig in VHDL beschrieben und anschließend mit Hilfe von FPGAs realisiert. Diese Implementierungen sind gleichermaßen als Hardwarebeschleuniger für die Berechnung selbst-organisierender Karten wie auch als Prototypen für eine mögliche ASIC Entwicklung zu verstehen.

3.2.1 Partitionierung

Wie in Abschnitt 3.1 dargestellt, kann die Hardwareimplementierung aus vielen gleichförmigen, parallel arbeitenden Verarbeitungseinheiten, den Prozessorelementen (PE) aufgebaut werden. Diese werden zu einem Prozessorfeld mit einer

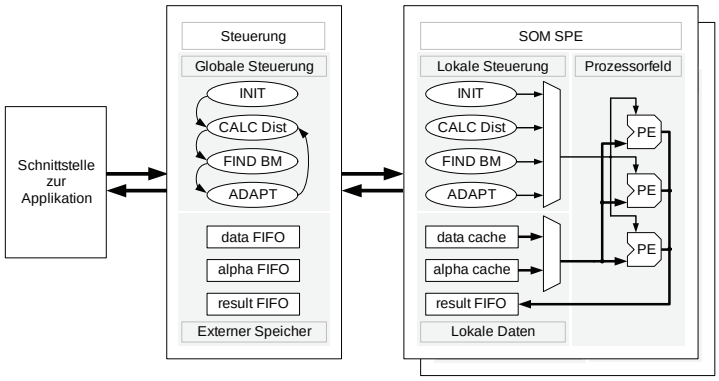


Abbildung 3.8: Partitionierung mit mehreren Hardware-Bausteinen

gemeinsamen Kommunikationsinfrastruktur zusammengefasst, welches von einer dedizierten Steuereinheit kontrolliert wird. Insbesondere hinsichtlich der Performanz und der Skalierbarkeit des Systems ist es wichtig, eine günstige Partitionierung bei der Verteilung auf mehrere ASICs oder FPGAs zu finden. Bild 3.8 zeigt eine Partitionierung, bei der durch die Aufteilung in globale Steuereinheiten und Steuereinheiten am Prozessorfeld im laufenden Betrieb ausschließlich Datenvektoren und Informationen zum Erregungszentrum ausgetauscht werden müssen; während in der Abbildung eine neuronen-parallele Implementierung dargestellt ist, kann dieser Ansatz für alle Architekturen verwendet werden.

Im Folgenden werden die Prozessorelemente für die oben genannten Ansätze im Detail verglichen, anschließend werden die Unterschiede auf der Ebene der Prozessorfelder und der lokalen Steuerung aufgezeigt. Abschließend wird der exakte Ressourcenverbrauch anhand der FPGA Implementierungen verglichen und somit der Einfluss der Steuerlogik beziffert.

3.2.2 SOM-Prozessorelement

Abbildung 3.9 zeigt die für die beiden Architekturen benötigten Datenpfade unter der Annahme, dass pro Takt ein Datum verarbeitet werden soll (siehe Abschnitt 3.1). Dabei wird offensichtlich, dass die benötigten arithmetischen Einheiten und deren Verschaltung dieselben sind, einzig die (im Bild nicht verzeichnete) Wortbreite des Addierers verändert sich: da beim neuronen-parallelen Ansatz eine Summation über bis zu d_m Teilsummanden vorgenommen werden muss (der quadratisch euklidische Abstand zweier Vektoren mit m Komponenten mit einer Wortbreite von $2(w+1)$ Bit), wächst die Eingangswortbreite entsprechend auf $2(w+1)+d_m$ an. Darüber hinaus sind beide Ansätze Spezialfälle des in Abbildung 3.2 dargestellten allgemeinen Prozessorelementes.

In diesem Bild lässt sich deutlich erkennen, warum der Flächenbedarf der beiden Implementierungen so deutlich unterschiedlich ist (siehe Abschnitt 3.1.1).

Im neuronenparallelen Fall

- muss es einen Selektionsmechanismus geben, da individuelle Speicherbereiche beschrieben werden müssen, etwa die Position des Neurons im Gitter (SELECT).

- muss es einen Mechanismus zur individuellen Zuweisung eines Adaptionswertes geben (ALPHA). Dies ließe sich prinzipiell auch über den Selektionsmechanismus erreichen, mit Hilfe einer entsprechenden Erweiterung kann dies aber deutlich zeiteffizienter durchgeführt werden. In der vorliegenden Implementierung werden für die Verteilung über den o.g. Selektionsmechanismus $5l_A$ Takte benötigt, mit Erweiterung sind dies $5+r_h$ Takte, wobei r_h den Radius der Nachbarschaftsfunktion darstellt, l_A bezeichnet die Anzahl der adaptierten Neuronen, es gilt $l_A \propto r_h^2$.

- muss das o.g. Summenregister (SUM) zur Speicherung der Distanz zwischen Eingabe- und Referenzvektor verwendet werden.

Da diese Funktionen für den neuronen- und komponentenparallelen Fall teilweise auf eine höhere Architekturebene (das Prozessorfeld) verschoben werden, ist nicht a priori abzusehen, inwieweit sich der Ressourcenbedarf der beiden Ansätze unterscheiden wird. Im Folgenden wird zunächst kurz die Funktionalität der Datenpfade beschrieben:

- **Initialisierung:** Bei der Initialisierung wird eine beliebige Anzahl an Zellen im lokalen Speicher, die als Referenzvektoren $\mathbf{m}_i$ dienen, mit Pseudo-Zufallswerten beschrieben. Der dazu benötigte Pseudo-Zufallszahlengenerator wird mit einer positionsabhängigen Konstante initialisiert, so dass die Werte für alle Elemente unterschiedlich sind. Im neuronenparallelen Fall werden zunächst die Koordinaten des Prozessorelements (x,y) an die Adressen null bzw. eins des lokalen Speichers geschrieben.

- **Abstandsberechnung:** Bei der Abstandsberechnung zwischen den Referenz- und Datenvektoren $\mathbf{m}_i$ bzw. $\mathbf{x}(t)$ werden im neuronenparallelen Fall die Datenvektoren elementweise über den Datenbus geschickt, während die Referenzvektoren aus dem lokalen Speicher gelesen werden. Es werden sowohl die Manhattan-Distanz $p = 1$ als auch die Euklidische Distanz unterstützt $p = 2$ (eigentlich quadratisch-euklidisch, siehe Abschnitt 2.1.2). Im gemischt neuronen- und komponentenparallelen Fall werden mehrere Komponenten des Datenvektors gleichzeitig übertragen. Die Summation der Teilergebnisse erfolgt auf einer höheren Ebene.

- **Erregungszentrum:** Im neuronenparallelen Fall wird die Suche mit dem in Abschnitt B.1 beschrieben Algorithmus durchgeführt, das dazu benötigte

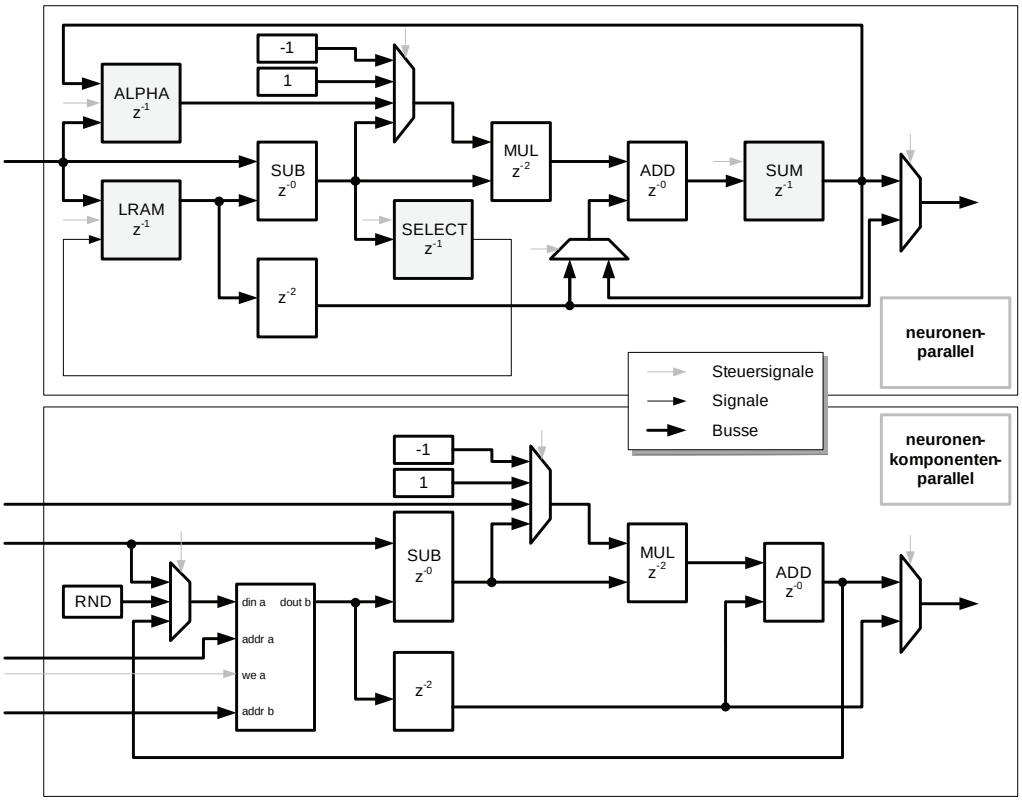


Abbildung 3.9: Datenpfade der neuronen- bzw. neuronen- und komponentenparallelen Architektur

Schieberegister wird im Modul SUM (ROTL in Bild B.2c) instanziiert. Die Suche resultiert in einer automatischen Selektion des Erregungszentrums, so dass die Position des Prozessorelementes im Prozessorfeld ermittelt werden kann. Diese wird für die spätere Adaption benötigt. Bei der neuronen- und komponentenparallelen Version erfolgt die Suche nach dem Erregungszentrum auf einer höheren Ebene.

- **Adaptionswerte:** Während im neuronen- und komponentenparallelen Fall auf Grund der vergleichsweise geringen Anzahl an physikalisch vorhandenen Neuronen die Adaptionswerte von der Steuerung direkt zugewiesen werden, erfolgt die Zuweisung im neuronenparallelen Fall individuell: der für die Adaption benötigte Alpha-Wert wird mit Hilfe des ALPHA Elementes gespeichert. Die Position des globalen Erregungszentrums wird über den Datenbus des Prozessorfeldes an die einzelnen PEs geschickt. Die Berechnung des Abstandes des individuellen Elementes zum Erregungszentrum wird analog zur Abstandsberechnung zwischen Vektoren durchgeführt und dann im Summenregister gespeichert. Im nächsten Schritt werden Adaptionswerte α_i über den Datenbus geschickt, während die Werte im Summenregister in

jedem Takt um c_{dec} (i.d.R. 1) dekrementiert werden. Wenn der Wert im Summenregister die Null erreicht, wird der Adaptionswert vom Datenbus im Alpharegister gespeichert. Auf diese Weise benötigt die Verteilung der Alpha-Werte $r_h + 2$ Takte, wobei r_h die Breite der Alpha-Funktion angibt. Die Form der Adaptionsfunktion wird vom für die Berechnung des Abstandes zum Erregungszentrum verwendeten Abstandsmaß bestimmt. Während sich bei der Verwendung der Manhattan-Distanz rautenförmige Isolinien (Abbildung 3.10a) ergeben, resultieren aus der Euklidischen Distanz kreisförmige Isolinien (Abbildung 3.10b).

- **Adaption:** Analog zur Abstandsberechnung werden in beiden Fällen die Elemente des Datenvektors über den Datenbus geschickt, die Differenz zwischen Daten- und Referenzelementen wird mit α multipliziert und anschließend in den korrekten Wertebereich verschoben (bei einem Wertebereich von $[0..2^n - 1]$ werden von den $2n$ Bits, die nach der Multiplikation auftreten nur die oberen n Bits benötigt). Dieser Wert wird auf das ursprüngliche $m_{i,j}$ addiert und unter der ursprünglichen Adresse gespeichert.

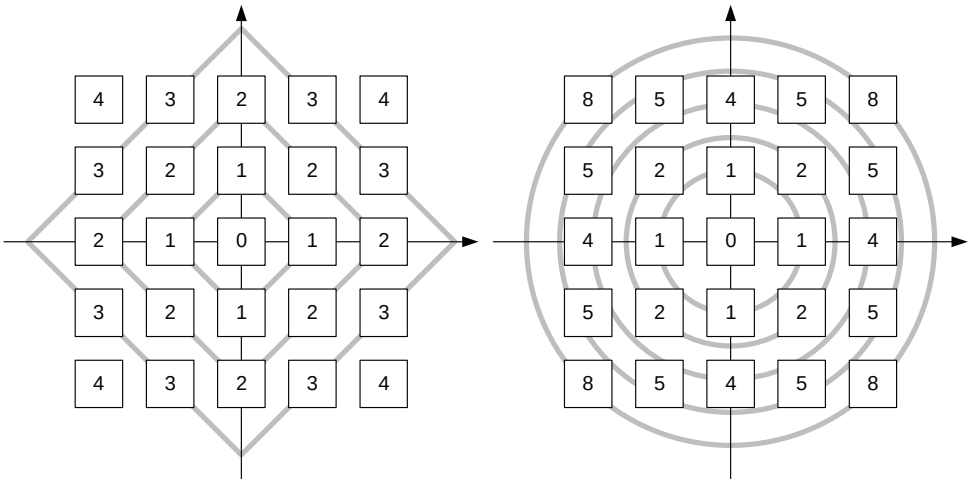
- **Auslesen:** Mit Hilfe der Selektionseinheit kann der Speicherinhalt von einer externen Quelle definiert bzw. ausgelesen werden. Nach einer Selektion kann nur noch das selektierte Element auf entsprechende Schreib- und Lese-Befehle reagieren, wodurch ein Auslesen individueller lokaler Speicher ermöglicht wird. Außerdem kann so auch der Speicherinhalt frei beschrieben werden, was wichtig für das Laden bereits gelernter Karten und für die Virtualisierung der Karte (s.u.) ist. Die beiden Architekturen unterscheiden sich hier nur durch den (oben beschriebenen) Selektionsmechanismus.

3.2.3 Virtualisierung

Ein wichtiger Aspekt für die effiziente Nutzung der hier beschriebenen Architekturen ist das Konzept der Virtualisierung. Dabei werden auf den vorhandenen Prozessorelementen eine größere Anzahl von Neuronen/Komponenten berechnet, als physikalisch vorhanden sind. Dabei werden die entsprechenden Referenzvektoren sequentiell im Speicher der Prozessorelemente abgelegt und auch sequentiell abgerufen. Dies führt zu einer erhöhten Latenz bei der Emulation einer SOM. Andererseits kann der lokale Speicher der Prozessorelemente besser ausgenutzt, und größere Karten berechnet werden. Dabei können abhängig von der Speichertiefe bis zu

$$v_{np} = \left\lfloor \frac{Speichertiefe}{d_m + 2} \right\rfloor \text{ (neuronenparallel)} \tag{3.4}$$

$$v_{nkp} = \left\lfloor \frac{Speichertiefe}{d_m} \right\rfloor \text{ (neuronen- } und \text{ komponentenparallel)} \tag{3.5}$$



(a) Manhattan (b) (quadratisch) euklidisch

Abbildung 3.10: Auswirkung des Distanzmaßes auf die Form der Adaptionsfunktion

virtuelle Elemente pro Prozessorelement emuliert werden. Die Datenpfade wurden so entwickelt, dass die Umschaltung zwischen virtuellen Elementen keine zusätzlichen Takte benötigt, alleine die Speicherinitialisierung im neuronenparallelen Fall wird komplexer: Die virtuellen Positionen aller Neuronen müssen in die lokalen Speicher geschrieben werden. Das ist nur durch eine externe Steuerung möglich und benötigt PE Selektionen und $2(v - \mathrm{PE})$ Schreibzugriffe, wobei PE die Anzahl der Prozessorelemente und v die Anzahl der virtuellen Neuronen angibt.

3.2.4 Architektur Prozessorfeld

Abbildung 3.11 zeigt den schematischen Aufbau der beiden Prozessorfelder. Wie schon in Abschnitt 3.2.2 beschrieben, müssen für den neuronen- und komponentenparallelen Aufbau einige Funktionen auf die Ebene der Prozessorfelder verlagert werden. So wird die Summe der Teilprodukte aus der Distanzberechnung jeweils für eine Gruppe von Prozessorelementen (schraffierte Bereiche in der Abbildung) berechnet, was in der Abbildung durch einen Summationsblock angedeutet wird. Die so gruppierten Prozessorelemente bilden eine Funktionseinheit, die alle Berechnungen eines Neurons (oder mehrerer virtueller Neuronen) durchführen können. Für die Suche nach dem Minimum ergeben sich hier zwei mögliche Verfahren: entweder wird das weiter oben beschriebene, bitserielle Verfahren angewandt, oder es wird ein Komparatorbaum verwendet. Für diese Entscheidung müssen Latenz und Flächenbedarf gegeneinander abgewogen werden (siehe Abschnitt 3.1.1).

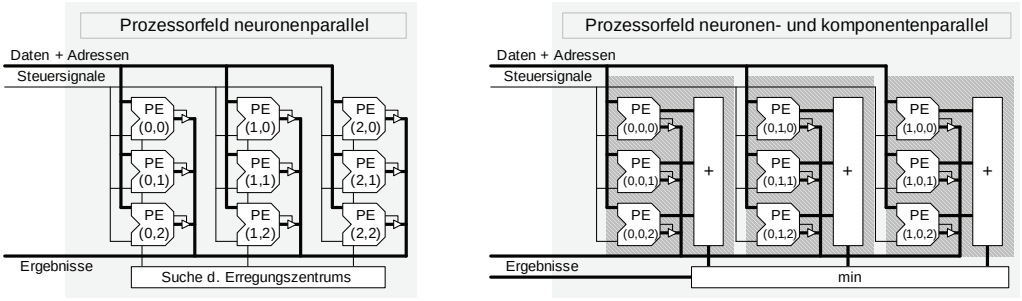


Abbildung 3.11: Prozessorfelder in beiden Architekturen. Die schraffierten Bereiche zeigen Funktionsgruppen, die als Neuronen arbeiten.

Da im neuronenparallelen Fall die Summation innerhalb der Prozessorelemente erfolgt, müssen auf der Prozessorfeldebene keine weiteren Summationsblöcke instanziiert werden. In der Abbildung wird die Suche nach dem Minimum nach dem bitseriellen Verfahren angedeutet. Die weiteren Kommunikationsstrukturen zum Transfer von Eingangsvektoren und Ergebnissen sind äquivalent aufgebaut, allerdings unterscheiden sich die Wortbreiten: während im neuronenparallelen Fall immer nur eine Komponente des Eingangsvektors anliegen muss, werden für die parallele Berechnung mehrerer Komponenten im neuronen- und komponentenparallelen Fall entsprechend mehrere Komponenten gleichzeitig benötigt.

Wie in 3.2.1 beschrieben, wird zu jedem Prozessorfeld eine lokale Steuerung implementiert, die die Kontrolle atomarer Abläufe übernimmt. Jeder dieser Abläufe wird durch einen endlichen Automaten (FSM, engl. Finite State Machine) beschrieben, der Zugriff auf einige Konfigurationsregister und auf die Datenschnittstellen hat. Derzeit sind die folgenden FSMs implementiert:

- **Initialisierung:** Alle Speicherzellen werden mit pseudo-zufälligen Werten beschrieben und die virtuellen Ebenen (Abschnitt 3.2.3) werden initialisiert. Im neuronenparallelen Fall beinhaltet dies das individuelle Setzen der Positionen der Neuronen im Gitter.

- **Laden:** Ein zuvor abgespeicherter Zustand einer Karte kann neu geladen werden, um etwa die Parameter des Lernvorgangs anzupassen.

- **Erregungszentrum:** Es wird das Erregungszentrum zu dem Datenvektor im Datencache gesucht. Die Position des Neurons und der Abstand zum Datenvektor werden im Ergebnis-FIFO gespeichert, so dass ein globales Erregungszentrum von mehreren Prozessorfeldern gefunden werden kann.

- **Erregungszentrum CSOM:** Bei dieser Implementierung wird das *Gewissen* der CSOM in die Berechnung einbezogen.

- **Adaption:** Die Adaption erfolgt nach den Vorgaben des Benutzers; im Adaptionsspeicher werden Länge und Gültigkeit der Adaptionsfunktion, sowie die Adaptionsfunktion selber abgespeichert. Dies ermöglicht eine sehr feingranulare Anpassung der Adaptionsfunktion, da zu jedem Vektor eine neue Funktion geladen werden kann.

- **Adaption CSOM:** Im Gegensatz zur reinen Adaption wird zusätzlich das *Gewissen* der CSOM angepasst.

- **Ergebnisse auslesen:** Um die Komponentenkarten bzw. die U-Matrix verarbeiten zu können, werden alle Referenzvektoren aus den Speichern der Prozessorelemente ausgelesen und in den Ergebnisspeicher geschrieben.

Diese Funktionen ermöglichen die Simulation sowohl einer SOM nach Kohonen, als auch einer Conscious-SOM nach [46], wobei eine übergeordnete Steuerung die Daten zur Verfügung stellen und die einzelnen FSMs starten muss. Des Weiteren ist die lokale Steuerung erweiterbar, Funktionen wie

- eine Adaption abhängig vom Abstand Datenvektor - Referenzvektor [29, 27]

- konstante Adaptionsfunktion (Diamond SOM, [93])

sind durch das Implementieren und Integrieren einer entsprechenden FSM möglich. Eine Vielzahl von Varianten des SOM-Algorithmus können auf diese Weise ermöglicht bzw. weiter beschleunigt werden.

3.2.5 Bestimmung der Adaptionswerte

Ein wesentlicher Unterschied zwischen den Architekturvarianten ergibt sich bei der Adaption: während die Positionen der Neuronen im Gitter immer lokal in den Prozessorelementen vorhanden ist, und damit der Adaptionswert aus einer Folge von Werten bestimmt werden kann (siehe Abschnitt 3.2.2), muss dies für den neuronen- und komponentenparallelen Fall separat erfolgen. Dazu kommen prinzipiell zwei Varianten in Frage:

1. Da die Struktur der Prozessorelemente weiterhin die Berechnung der Distanz zwischen dem aktuellen Neuron und dem Erregungszentrum analog zur Berechnung der Distanz zwischen Eingabe- und Referenzvektor erlaubt, kann das gleiche Verfahren wie im neuronenparallelen Fall angewandt werden. Dazu werden jeweils einem Prozessorelement pro Gruppe ein Speicher mit den Positionsdaten aller virtuellen Neuronen und die Adaptionswerte zur Verfügung gestellt. Die so entstandene Inhomogenität innerhalb der Gruppen hat einen höheren Aufwand für die Steuerung zur Folge. Der zusätzliche Hardwareaufwand, abgesehen von der erweiterten Steuerung, ist

mit jeweils zwei zusätzlichen Speichereinheiten (für Position und Adaptionswerte) pro Neuronengruppe zu beziffern. Die Latenz dieser Schaltung wächst linear mit der der Anzahl virtueller Ebenen, $L_\alpha = v(L_d + 1)$ Takte pro virtueller Ebene.

2. Durch Verwendung einer externen Funktionseinheit (außerhalb des Prozessorfeldes) kann die Berechnung der Adaptionswerte für alle virtuellen Ebenen (nach der ersten) parallel zur eigentlich Adaption erfolgen und somit kann die Adaption insgesamt beschleunigt werden. Zusätzlich kann bei dieser Variante die Berechnung seriell oder parallel durchgeführt werden, wodurch sich unterschiedliche Werte für Latenz und Ressourcenbedarf ergeben.

Welche dieser Varianten für das konkrete Einsatzgebiet implementiert werden soll, muss nach Betrachtung der Anforderungen und der Ressourceneffizienz entschieden werden. Im vorliegenden Fall wurde die erste Variante implementiert.

Das Prozessorfeld und die Lokale Steuerung sind auf verschiedenen Technologien (ASIC/FPGA) implementierbar und die Schnittstellen sind plattformunabhängig. Um diese Module nutzen zu können, müssen die Schnittstellen ggf. an die neue Plattform, etwa als Co-Prozessor in einem eingebetteten System oder als massiv paralleler Hardware-Beschleuniger angepasst werden. Da die beiden Komponenten (lokale Steuerung und Prozessorfeld) zusammen ein abgeschlossenes Modul ergeben, das in einem System beliebig oft instanziiert und verschaltet werden kann, bietet sich einer ASIC Fertigung dieser Baugruppe an. Im Folgenden wird die Leistung der Architekturvarianten wie auch deren Ressourcenverbrauch auf Basis einer prototypischen Implementierung mit dem RAPTOR Rapid Prototyping System ausgewertet.

3.3 RAPTOR

Die RAPTOR-Familie besteht aus modularen, FPGA basierten Rapid Prototyping Systemen, die vorwiegend im Bereich des ASIC Prototyping und der Beschleunigung massiv paralleler Strukturen eingesetzt werden. Das RAPTOR Basissystem stellt verschiedene Kommunikationsinfrastrukturen und Managementfunktionen für die bis zu sechs FPGA Module zur Verfügung: die FPGA Module (aktuell bis Xilinx Virtex 5) können von einem Host PC, der über die PCI/PCI-X/PCI-XPress Schnittstelle angebunden wird, konfiguriert und angesprochen werden. Für die Kommunikation der Module untereinander stehen zwei verschiedene Bussysteme und zusätzliche Punkt-zu-Punkt Verbindungen zur Verfügung. Im Rahmen dieser Arbeit wurde das RAPTOR2000 System eingesetzt, das mit bis zu 6 FPGA Modulen bestückt werden kann.

3.4 Anwendungsszenario

Als Anwendungsszenario für eine beispielhafte Evaluation des Entwurfsraumes dient eine Anwendung aus dem Bereich der hyperspektralen Bilder. Wie in Abschnitt 1.2.2 beschrieben, handelt es sich hier um ein bildgebendes Verfahren aus der Luft- und Raumfahrttechnik, bei dem zweidimensionale Aufnahmen einer Planetenoberfläche erstellt werden. Bei diesen Aufnahmen wird das Licht nicht wie in der digitalen Fotografie in drei sondern in bis zu mehrere hundert Farben aufgeteilt. Darüber hinaus erstreckt sich das beobachtete Spektrum vom nahen infraroten bis zum ultravioletten Licht. Die so gewonnenen *spektralen Signaturen* sind charakteristisch für die auf der Planetenoberfläche vorhandenen Stoffe und Stoffgemische. Aus diesen Informationen können Rückschlüsse, beispielsweise auf Mineralien oder Verunreinigungen im Boden gezogen werden. Dies ist besonders in erdfernen Missionen interessant, da Planetenoberflächen aus dem Orbit auf Erzvorkommen oder Kohlenstoffverbindungen untersucht werden können. Die SOM dient in diesem Fall als Klassifikator, der automatisch ähnliche Signaturen gruppiert. Über eine Rückprojektion der so gewonnenen Gruppen auf das ursprüngliche Bild kann eine Karte des fotografierten Bildes erstellt werden, die mit Informationen über die am Boden vorhandenen Stoffe angereichert ist. Aufgrund der beschränkten Ressourcen an Bord eines entsprechenden Flugkörpers ist es besonders wichtig, eine sehr flächen- und energieeffiziente Implementierung zu erzeugen.

In [93] wird ein Szenario beschrieben, in dem eine Karte mit $40 \times 40 = 1600$ Neuronen mit jeweils 194 Komponenten zur Segmentierung eines hyperspektralen Bildes mit $420 \times 614 = 257880$ Bildpunkten verwendet wird. Daraus ergibt sich, dass ein vollständiges Beschleunigersystem mindestens $1600 \times 194 = 310400$ Komponenten speichern können muss, was einer Speicherfläche von mindestens etwa $13,8 mm^2$ in einer 90 nm UMC Technologie entspricht. Daher wurden die im Folgenden untersuchten Konfigurationen so gewählt, dass unabhängig von Art und Anzahl der verwendeten Prozessorelemente genug Speicher für die Referenzvektoren zur Verfügung steht.

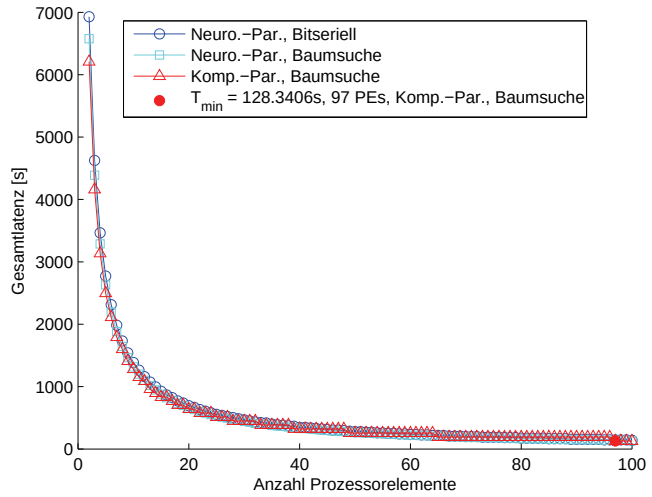
3.4.1 Einfluss der hardwarespezifischen Anpassungen

Mit Hilfe der in Kapitel 2 vorgestellten Methoden wurde untersucht, welche Wortbreite für das Lernverfahren mindestens zur Verfügung gestellt werden muss, um ein mit einer Softwareimplementierung qualitativ vergleichbares Ergebnis zu erreichen.

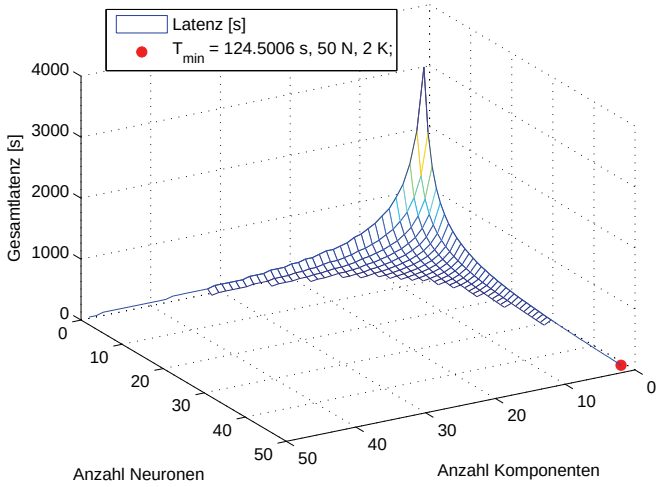
3.4.2 Latenz

In [93] werden keine Angaben zur Anzahl an Lernschritten gemacht, daher wurden zur Bestimmung der Latenz eine Anzahl von 10^7 Lernschritten angenom-

men[2]. Abbildung 3.12 zeigt die Latenz für die Berechnung der oben beschriebenen Konfiguration. In diesem Beispiel zeigt sich durch die *Treppenstruktur* die



(a) Komponenten- *oder* neuronenparallel



(b) Komponenten- *und* neuronenparallel

Abbildung 3.12: Latenz des Anwendungsbeispiels

Abhängigkeit der Latenz von dem Verhältnis zwischen physikalisch vorhandenen Prozessorelementen und den nutzbaren Prozessorelementen. So liegt das Optimum für die Latenz bei den komponenten- *oder* neuronenparallelen Varianten bei 97 Prozessorelementen für den komponentenparallelen Ansatz, weil sich die 194

[2]Die Anzahl der Lernschritte hat für alle relevanten Beispiele nur einen sehr geringen Einfluss auf das Endergebnis, da die sonstigen Verarbeitungsschritte (Initialisierung, Auslesen der Karte) bezogen auf die Latenz um Größenordnungen unter den Lernschritten liegen.

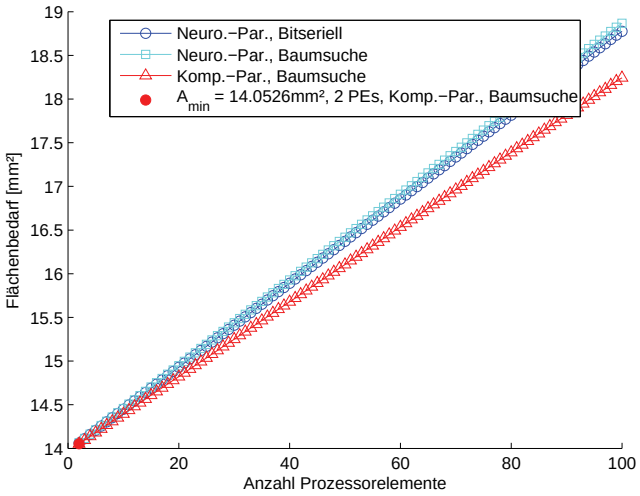
Komponenten des Datensatzes optimal auf 97 Prozessorelemente aufteilen lassen; die anderen Varianten fallen wegen des grundsätzlichen Geschwindigkeitsvorteils, siehe Tabelle 3.4, hinter dieser zurück.

Es zeigt sich ebenfalls, dass der gemischt neuronen- und komponentenparallele Ansatz mehr Prozessorelemente auslasten kann, und dadurch einen weiteren Geschwindigkeitszuwachs verzeichnen kann. Zusätzlich (in der Abbildung nicht explizit eingezeichnet) kann festgestellt werden, dass ausnahmslos alle Konfigurationen mit 100 Prozessorelementen mindestens die gleiche Geschwindigkeit wie der schnellste komponenten- *oder* neuronenparallele Ansatz bieten. Dies deutet darauf hin, dass für große Konfigurationen der gemischte Ansatz vorzuziehen ist.

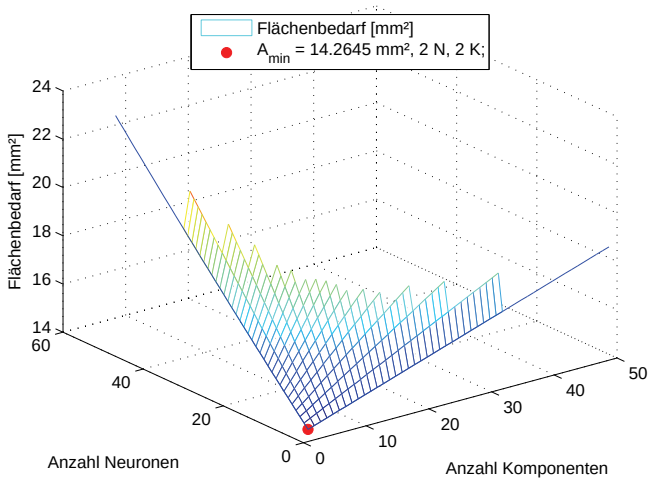
3.4.3 Fläche und Leistungsaufnahme

Die Auswirkungen der Steuerungslogik und insbesondere der lokalen Speicherelemente für Referenzvektoren, Adaptionswerte und Positionen auf die Gesamtfläche sind wie zu erwarten sehr groß, verhalten sich aber im Wesentlichen linear zur bisher betrachteten reinen Logik. Eine Ausnahme bildet die neuronen- und komponentenparallele Implementierung, bei der auf Grund der im vorangegangenen Abschnitt beschriebenen Verfahren zur Bestimmung von Adaptionswerten zusätzliche Speicherelemente pro Neuron vorgesehen werden müssen. Die Fläche für die vier Varianten wird in Abbildung 3.13 dargestellt, dabei wurde der Aufwand für die Steuerlogik der nicht vollständig implementierten Varianten auf Basis der beiden vorhandenen Implementierungen abgeschätzt. Da sich die Leistungsaufnahme, die aus Syntheseergebnissen errechnet wurde, proportional zur Fläche verhält, wurde diese nicht gesondert dargestellt. Wie die Abbildung zeigt, kann ein komplettes Beschleunigersystem für diesen Datensatz auf einer Fläche von $14mm^2$ aufgebaut werden, wobei der wesentliche Flächenanteil für die Speicherelemente ($13,8mm^2$, s.o.) benötigt wird. Für die größte (und, wie im vorangegangenen Abschnitt gezeigt, schnellste) Implementierung müssen $22mm^2$ aufgewendet werden. Werden diese Werte mit den korrespondierenden Latenzen ins Verhältnis gesetzt, so zeigt sich der Effekt der Parallelisierung: eine Vergrößerung der Fläche um den Faktor $1,5$ führt zu einer Beschleunigung um den Faktor 55. Dieser Zusammenhang spannt einen diskreten Entwurfsraum zwischen Latenz, Leistungsaufnahme und Fläche auf, der vom Anwender untersucht werden muss. Der Entwurfsraum wird typischerweise von Forderungen an die Implementierung eingeschränkt, etwa eine maximale Latenz oder Leistungsaufnahme. Die noch verbliebenen Punkte werden dann auf Pareto-Optimalität geprüft und vom Anwender bewertet. Für diese Bewertung und Auswahl kann unter anderem die Ressourceneffizienz RE nach Gleichung 3.3 verwendet werden.

Abbildung 3.14 zeigt die für dieses Anwendungsbeispiel relevante Paretofront bzw. Paretomenge. Dabei stellt sich heraus, dass die neuronenparallele Implementierung mit binärer Suche für dieses Beispiel nicht optimal ist. Stattdessen



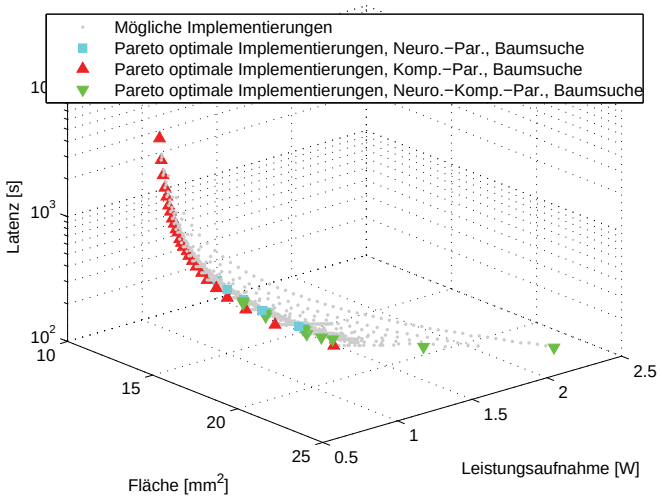
(a) Komponenten- *oder* neuronenparallel



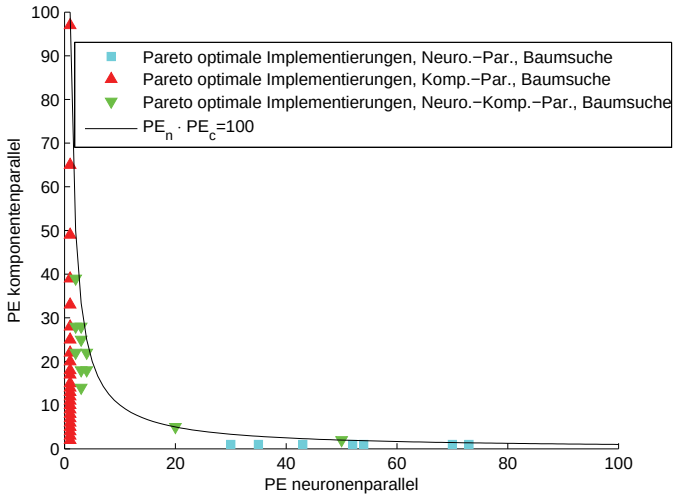
(b) Komponenten- *und* neuronenparallel

Abbildung 3.13: Flächenbedarf des Gesamtsystems inklusive der Referenzspeicher und Caches für Daten und Adaptionswerte

setzt sich die Paretomenge ausschließlich aus Punkten aus den anderen drei Architekturen zusammen, die damit als Basis für die weiteren Betrachtungen gelten können. Problematisch an den hier dargestellten Ergebnissen ist, dass (wie oben beschrieben) die benötigten Eingabe/Ausgabe-Datenschnittstellen nicht berücksichtigt werden. Da die Kosten dieser Datenschnittstelle mit der benötigten Bandbreite (von 48 MByte/s auf 9,5 GByte/s, siehe 3.1.2) steigt, wird die Entscheidung im Zweifelsfall eher für eine Implementierung mit wenigen komponentenparallel arbeitenden Prozessorelementen ausfallen.



(a) Paretofront

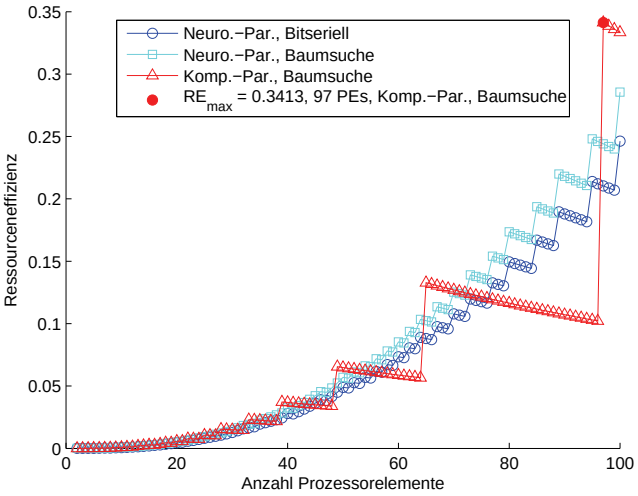


(b) Paretomenge

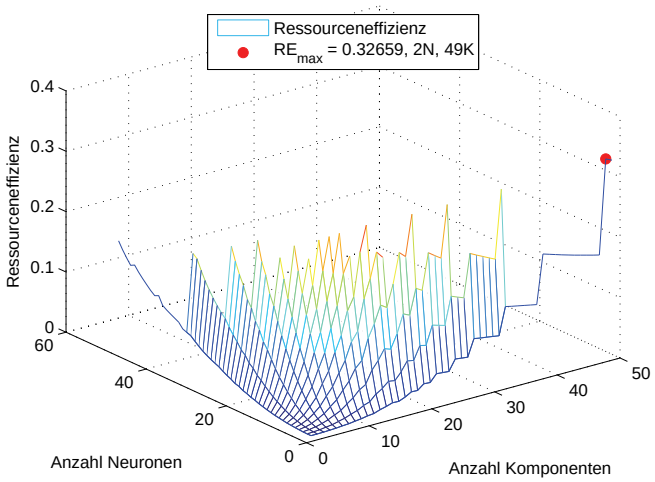
Abbildung 3.14: Paretofront und Paretomenge für Implementierungen gemäß Anwendungsbeispiel

3.4.4 Ressourceneffizienz

Die Auswertung der Ressourceneffizienz in Abbildung 3.15 zeigt wie die Latenz einen Verlauf mit Sprüngen, die durch die unterschiedliche Auslastung des Systems begründet sind. Neben den Sprüngen, die jeweils mit einem lokalen Optimum einhergehen, lässt sich erkennen, dass das globale Optimum für diesen Datensatz jenseits der Grenze von 100 PE liegt, und nur über eine Extrapolation bestimmt werden kann. Dies wurde hier nicht weiter untersucht.



(a) Komponenten- *oder* neuronenparallel



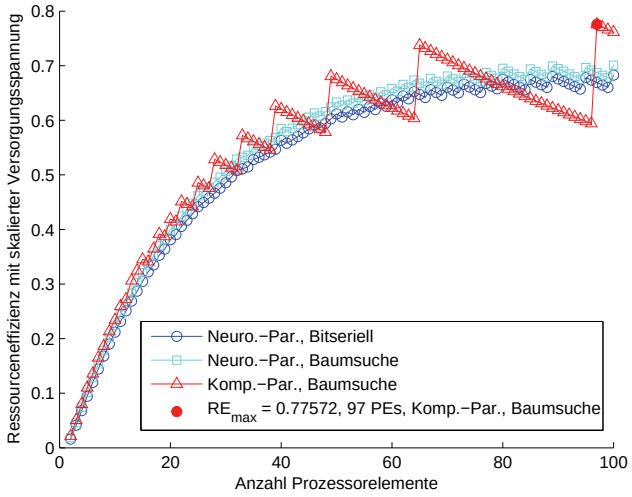
(b) Komponenten- *und* neuronenparallel

Abbildung 3.15: Ressourceneffizienz des Gesamtsystems inklusive der Referenzspeicher und Caches für Daten und Adaptionswerte

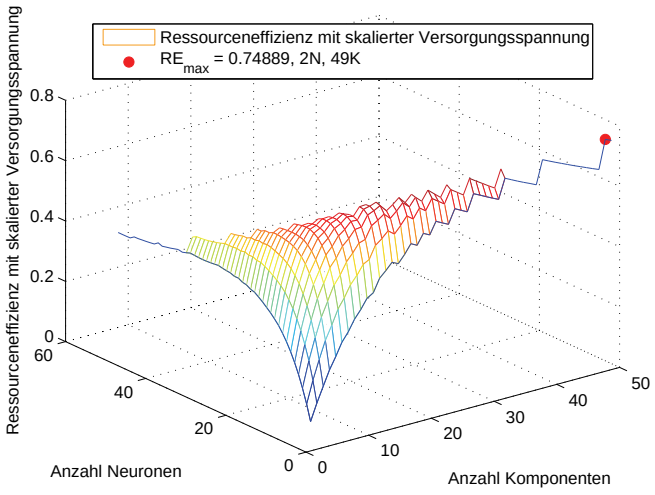
3.4.5 Technologische Optimierungen

Wie oben beschrieben, besteht unabhängig von der gewählten Implementierung, die Möglichkeit, den Energieumsatz über die Versorgungsspannung zu beeinflussen. Sobald die Verarbeitungsgeschwindigkeit höher als benötigt ist, etwa in einer Echtzeitumgebung in der eine bestimmte Menge an Vektoren pro Zeit verarbeitet werden müssen, können die Taktfrequenz und nach Gleichung 3.1 die Versorgungsspannung abgesenkt werden. Da die Versorgungsspannung quadratisch in

die Berechnung der Verlustleistung eingeht, sinkt diese entsprechend stark ab. Abbildung 3.16 zeigt diesen Zusammenhang in Bezug auf die Ressourceneffizienz.



(a) Komponenten- *oder* neuronenparallel



(b) Komponenten- *und* neuronenparallel

Abbildung 3.16: Ressourceneffizienz des Gesamtsystems mit skalierter Versorgungsspannung

Als Zeitlimit wurde die Latenz der langsamsten Implementierung gewählt und alle Implementierungen wurden entsprechend skaliert. Wieder lässt sich die Treppenstruktur erkennen, allerdings liegt hier das globale Optimum im extrapolierten Bereich. Der Verlauf der Ressourceneffizienz mit skalierter Versorgungsspannung unterscheidet sich deutlich von der nicht skalierten Variante.

Diese Methode erweitert den Entwurfsraum durch die Möglichkeit, ungenutzte

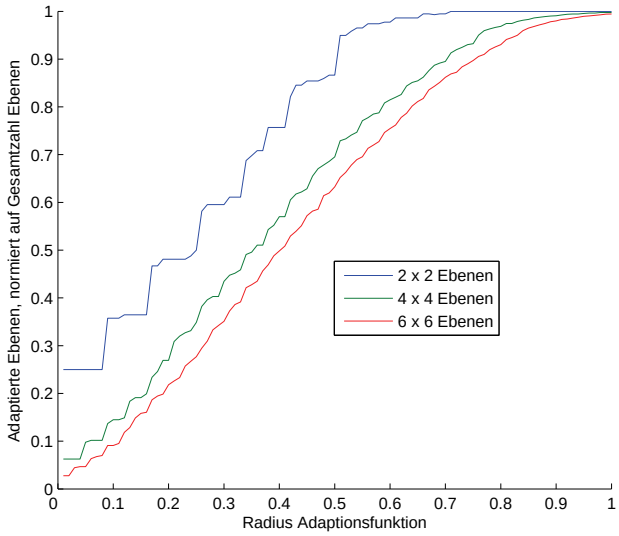


Abbildung 3.17: Verhältnis der adaptierten virtuellen Ebenen zu den vorhandenen virtuellen Ebenen

Ressourcen in Form von Zeit zu nutzen. Wenn unterschiedliche Anforderungen mit demselben System abgearbeitet werden sollen, kann hierdurch eine Abwägung zwischen Latenz und Energieaufnahme zur Laufzeit durchgeführt werden. Dies ermöglicht eine sinnvolle Nutzung des in diesem Anwendungsbeispiel betrachteten Systems auch für kleinere Karten.

3.4.6 Algorithmische Optimierungen

Abhängig von der Adaptionsfunktion und der Position des Erregungszentrums gibt es in der Regel Neuronen, die nicht adaptiert werden, d.h. $h_{c,i} = 0$. Dies kann bei der Berechnung von Karten mit mehreren virtuellen Ebenen ausgenutzt werden, um die Berechnungszeit zu reduzieren: alle Ebenen, bei denen für alle Neuronen $h_{c,i} = 0$ gilt, können bei der Adaption übersprungen werden. Ausgehend von der in [106] erstellten Abschätzung, dass im Durchschnitt 22% aller Neuronen adaptiert werden, kann der reine Adaptionsprozess gegenüber der zuvor gezeigten Variante im Idealfall etwa um den Faktor fünf beschleunigt werden. Tatsächlich hängt die Beschleunigung von der Anzahl der virtuellen Ebenen und damit von der Granularität der Gesamtkarte ab.

In Abbildung 3.17 ist diese Abhängigkeit beispielhaft aufgetragen. Die Kantenlänge der quadratischen Karte wurde dabei auf eins skaliert, und der Radius der Adaptionsfunktion im Intervall $[0,01..1]$ variiert. Mit steigendem Radius der Adaptionsfunktion steigt auch die Anzahl der zu adaptierenden Ebenen an. Gleichzeitig wird die Abhängigkeit von der Anzahl der virtuellen Ebenen deut-

lich: während die Anzahl der Ebenen steigt, die Karte also feiner unterteilt wird, werden insgesamt weniger Ebenen adaptiert. Da der Adaptionsschritt, bestehend aus dem Verteilen der Adaptionswerte und der eigentlichen Adaption einen wesentlichen Anteil an der Gesamtberechnungsdauer ausmacht (vgl. Tabelle 3.4, Adaption), kann die Einführung der oben genannten Optimierung eine hohe Leistungssteigerung mit sich bringen.

Diese Optimierung lässt sich sowohl im neuronen-, als auch im gemischt neuronen- und komponentenparallelen Ansatz anwenden: zunächst werden die Teile der Schaltung, die die Adaptionskoeffizienten berechnen, um ein zusätzliches Register erweitert, das die Anzahl der Adaptionskoeffizienten speichert. Nach der Berechnung des Abstandes zum Erregungszentrum wird überprüft, ob der Abstand größer ist als die Anzahl der Koeffizienten. Ist dies bei allen Prozessorelementen der Fall, so kann die lokale Steuerung den Adaptionsprozess für die aktuelle Ebene abbrechen, da kein Prozessorelement einen Adaptionswert ungleich null erhalten würde.

3.5 Auswertung

In diesem Kapitel wurden die prinzipiellen Architekturvarianten eines SOM-Hardwarebeschleunigers hinsichtlich ihres Ressourcenbedarfes untersucht. Dazu wurde auf Basis von einigen prototypischen, funktionsfähigen Implementierungen ein Modell für die Ressourcen Fläche, Latenz und Leistungsaufnahme erstellt, das innerhalb bestimmter Grenzen eine sehr gute Vorhersage für diese Ressourcen für neue Implementierungen erlaubt. Anhand dieser Modelle wurde die Ressourceneffizienz der Implementierungen untersucht. Hier sollen nun die in Abschnitt 1.6 beschriebenen ASIC Implementierungen mit den vorliegenden Varianten verglichen werden; die FPGA Varianten werden in Kapitel 4.3 untersucht.

Die Angaben der Autoren zu den ASIC Implementierungen in [60, 107, 124] beziehen sich nur auf vereinzelte Punkte im Entwurfsraum, obwohl sie sich für Implementierungen unterschiedlicher Größe und Leistung eignen. Daher ist es nicht möglich, die hier vorgestellte Vorgehensweise zum Vergleich der Systeme anzuwenden. Stattdessen werden die Referenzsysteme jeweils einzeln mit einer angepassten Variante der hier vorgestellten Architekturen verglichen. Dabei werden diese angepassten Varianten so gewählt, dass mindestens die in den Publikationen vorgestellten Karten simuliert werden können. Bei der Berechnung der Lernrate wird jeweils eine Karte mit der größtmöglichen Anzahl an Vektorkomponenten l_d verwandt. Da die Referenzimplementierungen nicht in einer 90 nm Technologie erstellt wurden, müssen zunächst einige Abschätzungen für die Skalierung der Technologie vorgenommen werden. Nach Standard-CMOS-Skalierungsregeln [55]

	Skalierte Referenzimpl.						Impl. diese Arbeit	
	l	d_m	w_m [Bit]	A_λ [mm^2]	$P_{Cl,\lambda}$ [MCUPS]	F_λ [MHz]	A [mm^2]	P_{Cl} [MCUPS]
[61]	256	16	12	5,29	11000	400	12,49	28054
[107]	16	128	8	0,36	1156	400	0,77	3750
[124]	64	3	8	1,55	6128	159	3,14	4173

Tabelle 3.5: Skalierte Referenzimplementierungen aus Tabelle 1.1 im Vergleich mit den hier entwickelten Implementierungen bei $F = 500$ MHz

gilt für eine Strukturverkleinerung mit dem Faktor $1/\lambda$:

$$T_\lambda = T/\lambda \tag{3.6}$$

$$A_\lambda = A/\lambda^2 \tag{3.7}$$

$$P_\lambda = P/\lambda^2 \tag{3.8}$$

In Tabelle 3.5 werden die Ergebnisse dieser Skalierung mit entsprechenden (s.o.) Implementierungen aus dieser Arbeit gegenübergestellt. Für den Vergleich wurde jeweils die schnellste Implementierung aus den vorgestellten Architekturvarianten ausgewählt.

Es fällt auf, dass die im Rahmen dieser Arbeit entwickelten Systeme etwa doppelt so viel Speicher benötigen wie die Referenzimplementierungen, was hinsichtlich der größeren Wortbreite (16 Bit) und des zusätzlichen Multiplizierers (außer in Sudha et. al. [124] wird in keiner Implementierung ein Multiplizierer eingesetzt) nicht überraschend ist. Bei der Lernleistung sind die hier entwickelten Varianten etwa drei Mal schneller als die Referenzimplementierung, einzig die auf drei Vektorkomponenten ausgelegte Implementierung von Sudha ist schneller als die hier vorgestellte. In diesem Fall zeigen sich sehr gut die Mehrkosten für die Implementierung eines zur Laufzeit parametrisierbaren Beschleunigers im Gegensatz zu einem Beschleuniger, der hinsichtlich Anzahl und Anordnung der Neuronen, Adaption und Anzahl der Komponenten keine Variation zulässt[3].

Im Idealfall könnte an dieser Stelle die Ressourceneffizienz zur Bewertung der Systeme herangezogen werden. Da in den Veröffentlichungen zu den Referenzimplementierungen einige Parameter zur Leistungsaufnahme fehlen, reduziert sich diese zum sog. Area-Delay Produkt (engl. Fläche-Verzögerung). Es kommt hinzu, dass die angegebenen Werte für die Lernleistung, die als Basis für die Bestimmung der Latenz dient, bei unterschiedlichen Parametern gemessen wurden, typischerweise so, dass der Beschleuniger ein Höchstmaß an Durchsatz bringen kann. Daraus resultiert, dass in unter diesen Umständen die Referenzimplementierungen untereinander nicht über die Ressourceneffizienz verglichen werden können.

[3]Die Implementierung von Sudha ist eine Produktentwicklung, die keiner Parameter zur Laufzeit bedarf.

	Area-Delay Produkt [10^{-6}mm^{2}s]	
	Skalierte Referenzimpl.	Impl. diese Arbeit
[61]	480	445
[107]	311	205
[124]	252	752

Tabelle 3.6: Area-Delay Produkt der Referenzimplementierung im Vergleich mit den hier entwickelten Architekturen

Demnach kann der Vergleich in diesem Fall über

$$RE \propto A/P_{Cl} \tag{3.9}$$

erfolgen. Die so definierte Metrik hat die Einheit $[mm^2/MCUPS = mm^2 s]$, ein kleinerer Wert bedeutet eine bessere Implementierung. Die Ergebnisse dieser Berechnung werden in Tabelle 3.6 dargestellt und zeigen ein ähnliches Bild wie Tabelle 3.5. Im Vergleich mit den beiden parametrisierbaren Beschleunigern aus [61] und [107] schneiden die hier vorgestellten Architekturen besser ab, die fixe Variante in [124] ist vergleichsweise deutlich besser. Wie in Abschnitt 1.6 bereits vorgestellt, kommt dieser Vorteil aber nur in dem einen Spezialfall zum Tragen, für den dieser ASIC entwickelt wurde; Sudhas Implementierung ist insbesondere fix hinsichtlich der Kartenparameter.

3.6 Zusammenfassung

In diesem Kapitel wurden vier verschiedene Architekturen für SOM-Hardwarebeschleuniger betrachtet und hinsichtlich ihrer Ressourcen untersucht. Dabei zeigt sich, dass keine optimale Architektur für alle Anwendungen gefunden werden kann. Vielmehr gibt es für jede Anwendung eine Vielzahl von Möglichkeiten, die von den Anforderungen der Anwendung eingeschränkt werden. Aus dieser Lösungsmenge ist dann die Pareto-optimale Menge auszuwählen und zu bewerten; als Bewertungskriterium wurde hier die Ressourceneffizienz RE herangezogen werden. Dieser Prozess wurde Anhand eines konkreten Beispiels aus der Luft- und Raumfahrttechnik vollzogen, in dem die SOM eine zentrale Rolle bei der Untersuchung hochdimensionaler Daten spielt.

Die hier vorgestellten Lösungen bieten für eine potentielle Implementierung der SOM einen sehr großen Entwurfsraum, innerhalb dessen unterschiedliche Eigenschaften der Implementierungen gegeneinander abgewogen werden können. Die vorgestellten Implementierungen sind hochflexibel hinsichtlich ihrer Parameter. So können vor einer Fertigung als ASIC bzw. Implementierung mit Hilfe eines FPGAs die Anzahl der Prozessorelemente, die Art des parallelen Aufbaus, die Größe der Referenzspeicher und die Rechenpräzision anhand von Parametern eingestellt werden, ohne die eigentliche Implementierung anfassen zu müssen. Zusätzlich können

die relevanten Kartenparameter wie Größe und Form der Karte, Abstandsmaße und Anzahl der Komponenten zur Laufzeit, also ohne neu Implementieren zu müssen, an das jeweilige Anwendungsszenario angepasst werden. Darüber hinaus sind die Implementierungen nicht als monolithisch zu verstehen, denn wie in Kapitel 4 gezeigt werden wird, können mehrere Bausteine sehr einfach zu einem größeren Beschleuniger kombiniert werden, wobei die der Verwaltungsaufwand linear mit der Anzahl der Bausteine wächst.

Im folgenden Kapitel werden zunächst allgemein schnelle Verfahren zum Einbetten und Testen digitaler Hardware in eine FPGA basierte Prototypen Umgebung beschrieben. Darauf aufbauend wird das oben diskutierte Anwendungsbeispiel anhand einer FPGA Implementierung auf dem RAPTOR System verifiziert.

Kapitel 4

FPGA-basierte Implementierung und Test

Die im vorangegangenen Abschnitt entwickelten Hardware-Implementierungen des SOM-Algorithmus wurden in VHDL beschrieben und mit Hilfe von Simulationen auf ihre Funktionalität überprüft. Zusätzlich wurde der Ressourcenbedarf mit Hilfe entsprechender Synthesen auf einen ASIC Prozess bestimmt. Bevor die so überprüften Systeme die weiteren Schritte einer ASIC Fertigung durchlaufen, wird typischerweise eine prototypische Implementierung auf Basis von FPGAs (engl. Field Programmable Gate Arrays, feldprogrammierbare Gatter Matrix) durchgeführt.

FPGAs (engl. Field Programmable Gate Arrays, feldprogrammierbare Gatter Matrix) sind Hardwarebausteine, die aus konfigurierbaren Logik- und Speicherzellen sowie Verbindungsressourcen bestehen. Moderne FPGAs integrieren zusätzlich noch große Zahlen an Funktionsblöcken wie Multiplizierer, RAM-Blöcke, Prozessoren etc., die zusammen mit den rekonfigurierbaren Ressourcen genutzt werden können. Die Kapazität der Bausteine und damit die Komplexität der Schaltungen, die auf diese Bausteine abgebildet werden können, wächst stetig an und ermöglicht somit den Test kompletter Systeme, bevor diese als ASIC gefertigt wurden. Dieser Schritt biete verschiedene Vorteile:

- Durch die in der Regel deutlich höhere Systemtaktrate eines FPGA Prototypens gegenüber einer Software-basierten Simulation wird das Durchführen von Tests aller Art beschleunigt.

- Daraus ergibt sich die Möglichkeit, eine sehr große Anzahl von Tests zur Verifikation der Gesamtfunktionalität durchzuführen.

- Unter Umständen kann es Arbeitsbereiche geben, die in einer späteren Systemumgebung nicht erreicht werden können oder dürfen. Im Rahmen von FPGA-basierten Hardware-in-the-Loop Simulationen können diese problem- und gefahrlos untersucht werden.

- Durch die erhöhte Simulationsgeschwindigkeit, und durch die Nutzbarkeit realer elektrischer Schnittstellen, wie z.B. serielle Verbindungen (UART), kann für Systeme mit Prozessoren schon früh mit der Softwareintegration begonnen werden.

- Allgemein können Eigenschaften der Hardware, die im Rahmen einer Simulation nicht entdeckt wurden, im Rahmen der Tests aufgedeckt werden.

Dies gilt allgemein für digitale mikroelektronische Schaltungen, insbesondere auch für die im vorangegangenen Abschnitt vorgestellten Beschleunigersysteme. Im praktischen Einsatz bedeutet die Implementierung in FPGA-Logik aber auch einen großen zusätzlichen Aufwand, u. A. weil die Schnittstellen des Systems an die Prototypen-Umgebung angepasst werden müssen.

Um also die FPGA Ressourcen für Entwickler nutzbar zu machen werden sog. Rapid Prototyping Systeme (engl. Systeme für die schnelle Prototypenentwicklung) auf Basis von FPGA Technologie angeboten. Gleichzeitig werden Softwaresysteme erstellt, die die Abbildung von Schaltungen auf diese Systeme vereinfachen sollen. Im folgenden Kapitel werden zwei Systeme für FPGA-basierte Tests vorgestellt, die im Rahmen dieser Arbeit entwickelt und verwendet wurden. Diese werden eingesetzt, um die in Kapitel 3 entwickelten SOM-Beschleuniger in eine Rapid Prototyping Umgebung zu integrieren. Anschließend wird die Leistung des SOM-Beschleuniger in einer realen Systemumgebung vermessen und mit den in Tabelle 1.1 vorgestellten FPGA-basierten Implementierungen verglichen. Dabei werden insbesondere die Kosten der Integration des Beschleunigersystems in die Systemumgebung betrachtet.

Die hier vorgestellten Systeme wurden in Zusammenarbeit mit Carlos Paiz erstellt und erscheinen in ähnlicher Form in [99] Kapitel 5 auf. In dieser Arbeit werden insbesondere allgemeine Prinzipien vorgestellt, die für Integration und Test digitaler Schaltungen verwendet werden können, während in [99] besonders auf mechatronische Systeme eingegangen wird.

4.1 Hardware-in-the-Loop

Hardware-in-the-Loop (HiL) Simulationen erlauben es Entwicklern, durchgängig in allen Stadien des Entwurfsablaufs (siehe Abbildung 4.1) Simulationen mit hoher Realitätsnähe durchzuführen. Das zu Grunde liegende Konzept der HiL-Simulation beschreiben Iserman et. al. [71] als eine Interaktion zwischen einer

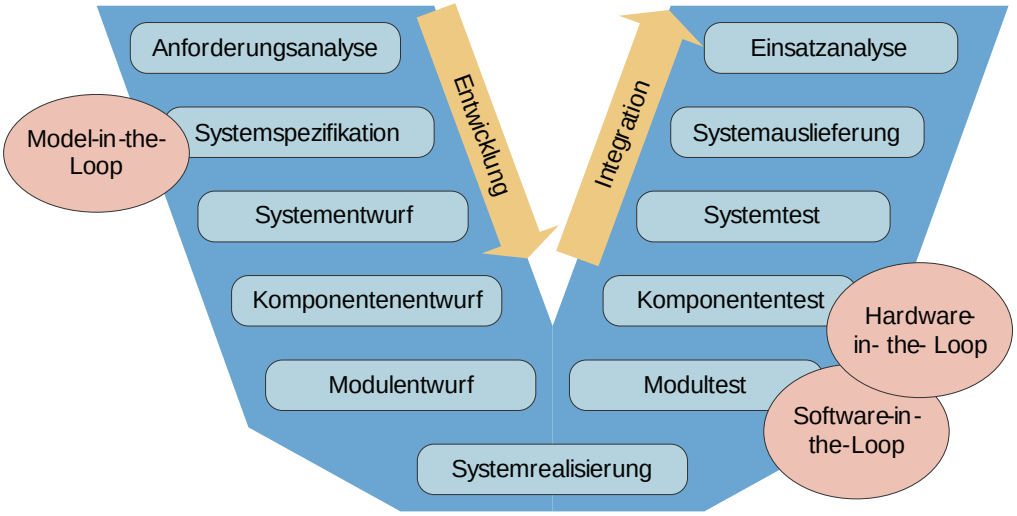


Abbildung 4.1: Das V-Modell mit Einsatzmöglichkeiten für verschiedene HiL-Technologien

realen und einer in Echtzeit simulierten Systemkomponente. Demnach existieren zwei verschiedene Möglichkeiten für HiL-Simulationen:

1. Die Kombination einer realen Informationsverarbeitenden Einheit mit einem Modell des umgebenden Systems, oder

2. die Kombination einer simulierten Informationsverarbeitung mit dem realen zu steuernden System.

Im ersten Fall wird in der Regel ein reales (evtl. prototypisches) Steuergerät in Verbindung mit einem Modell der Umgebung getestet [122, 84, 43], etwa weil das reale System noch nicht zur Verfügung steht oder weil Extremsituationen getestet werden sollen, welche die reale Regelstrecke beschädigen würden. Entsprechend ergibt sich für den zweiten Fall die Möglichkeit, eine reale Regelstrecke mit einem simulierten Regler zu koppeln, etwa weil dieser noch nicht in der Zieltechnologie vorhanden ist [70, 86, 159] bzw. sich gerade in der Evaluierungsphase befindet. Dieser Ansatz erhöht gegenüber einer reinen Softwaresimulation die Aussagekraft der Tests, da hier die durch die Abstraktion der Modellierung bedingten Ungenauigkeiten des Umgebungsmodells entfallen.

Die zentrale Herausforderung aus Sicht der Realisierung einer HiL-Simulation liegt bei beiden Ansätzen in der Synchronisation zwischen dem Modell und dem realen Teilsystem. Dazu muss sichergestellt werden, dass das Modell zu fest definierten Abtastzeitpunkten t aktualisiert wird, also $t = n\delta\tau$, wobei $\delta\tau$ die Abtastperiode darstellt. Die Modelle werden in der Regel mit einer Simulationssoftware wie z.B. Matlab/Simulink erstellt, die Latenz ist in diesem Fall abhängig von der Rechenleistung und Auslastung des zugrunde liegenden Rechnersystems. Sie lässt

sich nicht im Voraus bestimmen und ist auch im Verlauf der Simulation nicht konstant. Um trotzdem harte Echtzeitkriterien gewährleisten zu können, werden spezielle Beschleunigersysteme eingesetzt, die auf Basis eines Echtzeit Betriebssystems vorhersagbare Antwortzeiten produzieren. Solche Systeme werden unter anderem kommerziell von dSpace [104], ETAS [28] und National Instruments [68] angeboten. Es handelt sich dabei um spezialisierte Hardwarekomponenten mit deterministischen Kommunikationskanälen, auf die das Modell mit Hilfe eines Übersetzers abgebildet wird.

Im Rahmen dieser Beschleunigersysteme werden FPGAs derzeit in mehreren Bereichen verwendet. Wegen der großen Anzahl an möglichen Ein- und Ausgängen, die zudem sehr effizient mit Basisfunktionalität ausgestattet werden können, werden FPGAs häufig an der Schnittstelle zwischen Simulationshardware und dem realen System verwendet. In [20] wird diese Kopplung verwendet, um eine sehr exakte Abtastung eines PWM-Signals zu erreichen. Visser et. al. [147, 146] verwenden FPGAs, um Sensoren am zu testenden Steuergerät zu emulieren.

4.1.1 HiL für FPGAs

Die Komplexität heutiger FPGAs ist so groß, dass komplette Regel- und Steuerungssysteme auf einem einzigen FPGA integriert werden können, womit der FPGA an sich zur Zielplattform der Entwicklung wird. Solche Realisierungen sind unter anderem für die Regelung von Motoren [36, 130, 66], Leistungselektronik im Allgemeinen [69, 154], industrielle Regelungen [97, 51] und Sensorüberwachung [30, 62] zu finden. Die Autoren berichten unter anderem von Beschleunigung, erhöhter Flexibilität bei geringeren Kosten und einem niedrigen Energieverbrauch gegenüber Microcontroller-basierten Lösungen.

Da der FPGA also auch selber zum Objekt einer HiL-Simulation wird, ergeben sich neue Möglichkeiten für die Durchführung solcher Tests. Ein besonderer Vorteil von FPGAs in diesem Zusammenhang ist die Möglichkeit, neben der Realisierung der eigentlichen Funktionalität auch die Testschaltungen über denselben Entwurfsablauf zu integrieren. Im Folgenden wird ein System zur Durchführung von HiL-Simulationen vorgestellt, bei dem explizit ein FPGA das zu testende Objekt ist:

Das HAC2 [112] von Gleichmann Research GmbH ist eine HW/SW-Lösung zur Integration von FPGAs in Software-basierte Simulationen wie sie z.B. mit Mentor Graphic's ModelSim möglich sind. Dabei werden Teile des entwickelten Systems auf den FPGA ausgelagert und als sog. Black-Box wieder in die Simulation des Gesamtsystems eingebunden. Die Ein- und Ausgänge der ausgelagerten Teilsysteme werden dabei auf entsprechende Ein- und Ausgänge des FPGAs abgebildet. Zwei weitere FPGAs (sog. I/O Manager) übernehmen dann die Kommunikation zwischen den ausgelagerten Teilsystemen und dem Computer, auf dem das Gesamtsystem simuliert wird. Über ein entsprechendes Software-Modul, das als

Schnittstelle zum Software Simulator dient, wird die Schleife zwischen FPGA und Simulation geschlossen.

Ein großer Vorteil dieser Vorgehensweise liegt in der speziellen Art der Synchronisation beider Teilsysteme: Anstatt die Softwaresimulation mit großem Hardwareaufwand so zu beschleunigen, dass das zu testende System auf dem FPGA in Echtzeit bedient werden kann, wird das FPGA-Teilsystem verlangsamt bzw. direkt von der Simulation gesteuert. Das ist im Falle eines FPGAs immer dann möglich, wenn keine analogen, frequenzsensitiven Bauteile wie DLLs oder PLLs eingesetzt werden, da eine Verlangsamung der Taktfrequenz nicht die Funktionalität des Systems beeinflusst. Da die Dauer eines Taktes in Hardware in der Regel vernachlässigbar klein im Vergleich zu einem simulierten Takt in Software ist, entscheidet nur der Kommunikationsaufwand in Hardware und Software darüber, ob zusätzlich auch die Gesamtsimulationsdauer reduziert wird.

4.2 FPGA-basierte Testsysteme

Während HiL-Simulation hauptsächlich bei Datenfluss-dominierten Anwendungen zum Einsatz kommen, sind auch für allgemeinere Anwendungen FPGA Testsysteme verfügbar, die im Folgenden vorgestellt werden sollen. Die Testsysteme benötigen in der Regel keine speziellen FPGA-Plattformen sondern greifen auf vorhandene FPGAs über die JTAG-Schnittstelle zu.

Das Programm Identify von Synplicity wird mit einer Debugger-ähnlichen Oberfläche ausgeliefert, in der der Benutzer schrittweise durch den Quelltext gehen kann, während dieser eigentlich auf der Hardware ausgeführt wird. Für dieses Konzept gibt es im Wesentlichen zwei Szenarios: Erstens können Diskrepanzen zwischen HDL-Quelltext und FPGA-Version aufgedeckt werden. Zweitens können sehr langwierige Simulationen beschleunigt werden, indem der Debugger das FPGA-System bis zum Eintreten eines bestimmten Ereignisses beobachtet, um dann in den schrittweisen Modus zu wechseln. Beispiele für langwierige Simulationen sind Programme, die auf FPGA-basierten Prozessoren ausgeführt werden oder Anwendungen, bei denen Einschwingvorgänge eine Rolle spielen.

Die Firma Xilinx liefert das Chipscope System für Xilinx FPGAs, mit dem ein Logik-Analysator in beliebige Systeme integriert werden kann. Dieser speichert, vom Benutzer konfigurierbar, interne wie externe Signale in einem FPGA-internen Speicher und stellt diese anschließend in einer Softwareoberfläche grafisch dar. Nachteile von Chipscope liegen in dem begrenzten Speicher einerseits und der fehlenden Möglichkeiten zur Stimulierzeugung und Parametrierung andererseits. Daher wurde im Rahmen dieser Arbeit das HiLDE-System (Hardware-in-the-Loop Design Environment) entwickelt, welches beide Nachteile von Chipscope ausgleicht, ohne dabei den Anspruch zu haben, ein vollständiger Ersatz für Chipscope zu sein.

4.3 Das HiLDE-System für Offline Simulationen

HiLDE [5, 6] ist ein Hardware- Softwaresystem, das auf einfache Weise eine HiL-Simulation in rekonfigurierbarer Hardware ermöglichen. Dabei sind alle Schritte, von der Erzeugung der Hardwareschnittstelle bis zur Ankopplung an verschiedene Simulatoren automatisiert. Im Folgenden werden das zugrunde liegende Funktionsprinzip und die Schnittstellen beschrieben. Darüber hinaus wird die Leistungsfähigkeit des Systems bestimmt und seine Grenzen aufgezeigt.

4.3.1 Funktionsprinzip

Das HiLDE System beschränkt sich im Gegensatz zu den in Abschnitt 4.2 beschriebenen HiL-Systemen auf die Einbindung digitaler, synchroner Logik in eine Software-basierte Simulation. Dabei wird die Hardware auf einem FPGA implementiert und über entsprechende Schnittstellen mit dem Simulator gekoppelt. Der prinzipielle Unterschied zwischen einem klassischen HiL-Simulator und dem hier vorgestellten System liegt in der Synchronisation zwischen DUT (engl. Design Under Test, zu testender Entwurf) und Simulation[12]. Bei digitaler synchroner Logik werden alle Zustandsübergänge ausschließlich über den Takt gesteuert. Diese Tatsache wird in dem vorgestellten System in der Weise genutzt, dass der Takt für das DUT, welches in der Regel wesentlich schneller arbeitet als die Simulation ($T_{\mathrm{DUT}} << T_{\mathrm{Sim}}$), siehe Abschnitt 4.3.7), angehalten bzw. von der Simulation gesteuert wird. Diese Taktsteuerung hat abgesehen vom Zeitverhalten keinen Einfluss auf die Funktionalität des Systems. Daraus ergeben sich zwei Einschränkungen für die Verwendbarkeit des HiLDE Systems. Erstens werden durch die Verlangsamung des Systemtaktes alle Effekte, die durch unterschreiten der minimalen Latenz im normalen Betrieb auftreten können eliminiert. Fehler, die im realen Betrieb durch solche Verletzungen des längsten Pfades auftreten können also nicht gefunden werden. Zweitens können analoge Komponenten, die eine minimale Frequenz oder eine maximale Schwankung des Taktes (engl. Jitter) zur korrekten Funktion benötigen, unter Umständen nicht korrekt arbeiten. Bei den Komponenten handelt es sich zumeist um spezielle analoge Blöcke wie DLLs oder PLLs, wie sie in Systemen mit mehreren Systemtakten vorkommen.

In Abbildung 4.2 wird die Einbettung des DUT in die HiL-Simulation dargestellt. An den Ein- und Ausgängen des DUT werden Speicherelemente instanziiert, die Werte von der Simulation und für die Simulation vorhalten. Diese sind über eine Busschnittstelle mit dem Simulationsrechner verbunden, wo eine entsprechende Simulationssoftware (Matlab Simulink, ModelSim etc.) lesend und schreibend zugreifen kann. Zusätzlich ist der sog. *Synchronizer* an den Bus angekoppelt, der die Taktsteuerung auf Hardwareebene realisiert. Dieser ist mit dem Takttreiber des DUT verbunden und kann über die Busschnittstelle parametriert und gestartet werden, so dass der Systemtakt des DUT durch die Software gesteuert wird.

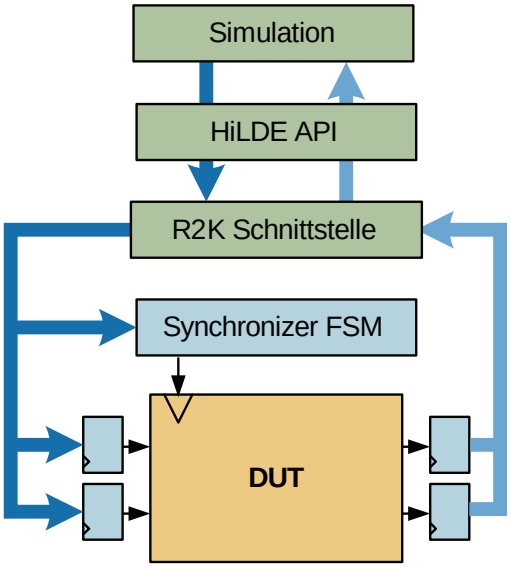


Abbildung 4.2: Funktionsprinzip der HiLDE Synchronisation

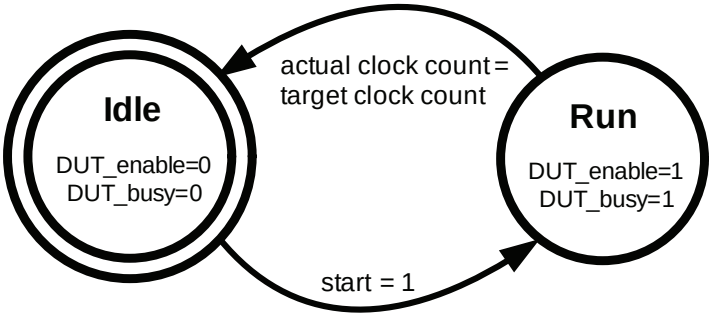


Abbildung 4.3: Taktsteuerungsautomat

Der Synchronizer ist als Moore Automat ausgeführt, die Funktionsweise ist in Abbildung 4.3 zu erkennen.

Das HiLDE System kann im Prinzip über jeden beliebigen Bus an den Simulationsrechner angeschlossen werden; im vorliegenden Fall wird das RAPTOR2000 System eingesetzt, daher wird über den PCI/LocalBus kommuniziert. Dieser bietet gegenüber anderen Systemen wie etwa JTAG den Vorteil, dass bis zu 100 MByte/s netto Transferraten erzielt werden können. Das ist im Gegensatz zu JTAG mit 0,5 MByte/s brutto eine Steigerung um einen Faktor größer 20, was sich stark auf die Latenz eine HiL-Simulation auswirkt.

4.3.2 Softwareschnittstelle

Da HiLDE als plattformunabhängiges System konzipiert wurde [14], ist die Softwareschnittstelle in zwei Module geteilt. Das hardwarespezifische Modul stellt die

Basisfunktionalität zur Kommunikation mit dem FPGA zur Verfügung und ändert sich mit dem verwendeten Bus. Für das RAPTOR2000 System setzt sich dieses Modul im Wesentlichen aus der sog. R2KLIB, einer Bibliothek zum Zugriff auf das RAPTOR2000 System, und einer Abstraktionsschicht zusammen. Die Abstraktionsschicht kapselt Steuerungsfunktionen und Zugriffe auf die Register der HiL-Hardware.

Das simulationsspezifische Modul stellt die Anpassungen an die jeweilige Simulationsumgebung dar. Derzeit existieren Module für drei Simulatoren:

- **ModelSim:** Dieser Simulator der Firma Mentor Graphics dient der Simulation von Netzlisten in der Hardwareentwicklung. Mentor bietet das sog. Foreign Language Interface (FLI, Erweiterte Sprachschnittstelle) an, mit dem die HiLDE-Softwarekomponente integriert wird. Nachdem diese innerhalb einer Simulation gestartet wurde, verhält sich das HiLDE Modul wie eine Entität mit der Schnittstelle der ausgelagerten Hardware und übernimmt das Senden und Empfangen von Daten.

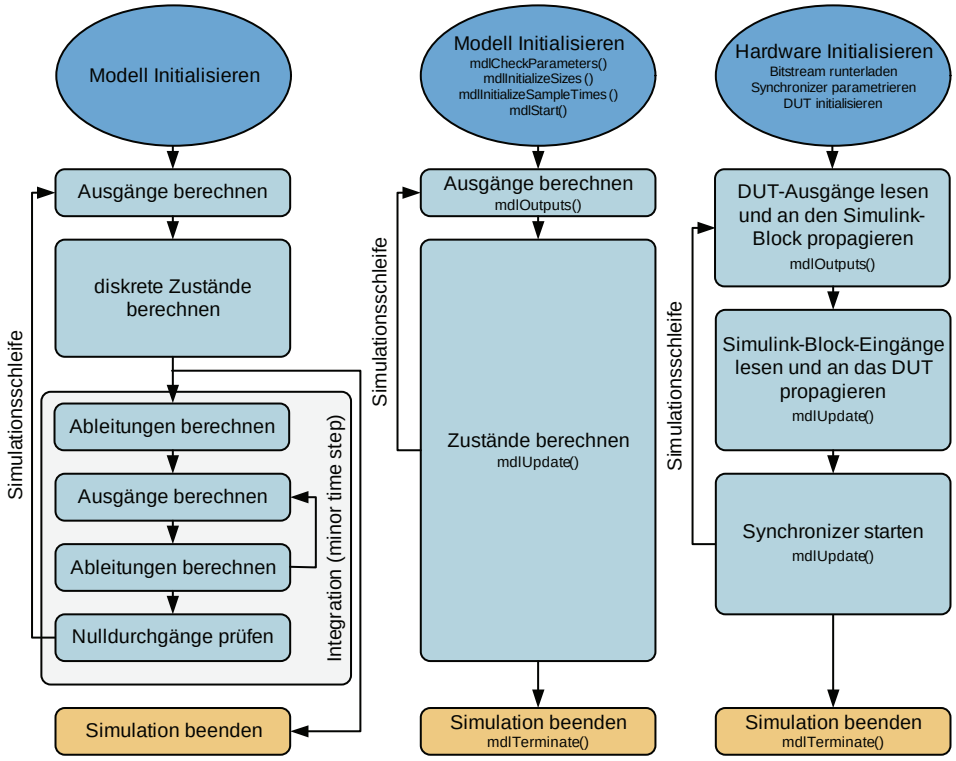
- **Simulink:** Das Datenfluss-orientierte Simulationspaket Simulink wird von verschiedenen Herstellern von EDA-Software um sog. Block-Sets erweitert, die das grafische Erstellen von Hardwarebeschreibungen für FPGAs oder ASICs ermöglichen (Xilinx System Generator, Altera DSPBuilder, Synplicity SynplifyDSP). Hier bietet HiLDE die Möglichkeit, Teile eines neuen Entwurfes auf den FPGA auszulagern und über eine sog. S-Funktion in die Simulation zu re-integrieren. Die S-Funktion ist Simulinks Schnittstelle für Benutzerprogramme und bietet Funktionen, um die Ein- und Ausgänge des SFunktions-blocks (die grafische Repräsentation der S-Funktion) zu lesen und zu schreiben. Das HiLDE-Modul für Simulink nutzt diese Schnittstelle, um die Kommunikation zwischen Simulink und der FPGA-Hardware zu ermöglichen.

- **CAMeL-View:** Das Simulationspaket CAMeL-View von iXtronics ist sehr ähnlich wie Simulink aufgebaut und hauptsächlich für Anwendungen aus dem Maschinenbau optimiert. Die HiLDE-Schnittstelle für CAMeL [7] entspricht im Wesentlichen der Simulink-S-Funktion.

Neben diesen Integrationen für verschiedene Simulatoren wurde auch eine Schnittstelle für Systeme entwickelt, die keinen intrinsischen Zeit- und Ereignisbegriff haben. Diese werden in Abschnitt 4.3.5 besprochen.

4.3.3 Integration in Simulationswerkzeuge

Die Integration externer Hardware-Komponenten in einen Simulator ist insofern wenig komplex, als der Simulator einen intrinsischen Zeit- und/oder Ereignisbegriff hat. Dieses Konzept ermöglicht eine direkte Ankopplung einer HiL-



(a) HiL-Simulationsschleife (b) Allgemeine S-Funktion (c) HiLDE-S-Funktion

Abbildung 4.4: Integration von HiLDE in Simulink

Simulation, ohne Datenabhängigkeiten extern prüfen zu müssen. Entsprechend fügt sich eine HiL-Simulation ohne konzeptuelle Änderungen in die vom Simulator vorgegebenen Schritte. Die folgende Beschreibung der Integration mittels einer S-Funktion gilt analog für andere Simulatoren.

In Abbildung 4.4a ist der (vereinfachte) Ablauf einer Simulation in Simulink inklusive der Lösungsschritte für Differentialgleichungen dargestellt. Entsprechend ist in Abbildung 4.4b das S-Funktionsschema für einen externen Funktionsblock dargestellt, bei dem es dem Programmierer überlassen bleibt, kontinuierliche oder diskrete Funktionen zu realisieren. Für eine HiL-Simulation, in der die gesamte Funktionalität in der externen Hardware vorhanden ist, reduziert sich der Ablauf auf die in Abbildung 4.4c aufgeführten Schritte:

1. **Initialisierung:** In dieser Phase wird zunächst die HiLDE Konfiguration geladen und ausgewertet. Diese enthält neben Informationen über den FPGA-Bitstrom auch sämtliche Informationen zu den Abtastraten und Datenformaten der Ein-und Ausgänge. Anschließend wird das FPGA mit dem Bitstrom, der sowohl das DUT als auch die Testumgebung enthält, initialisiert und der *Synchronizer* mit der Anzahl der auszuführenden Takte parame-

triert.

2. **Lesen der Ausgänge:** Die Informationen aus den Speichern am Ausgang des DUT werden gelesen und an den SFunktions-Block weitergeleitet. Ggf. wird eine Nachbearbeitung durchgeführt, in der die Datenformate zwischen DUT (in der Regel Festkommadarstellung) und Simulation (Festkomma oder Gleitkommadarstellung) umgerechnet werden.

3. **Schreiben der Eingänge:** Entsprechend werden in diesem Schritt die Eingangsdaten vom Simulink Block entgegen genommen, auf die gleiche Weise nachbearbeitet und an das DUT weitergeleitet.

4. **Starten des Synchronizer:** Der Synchronizer, der die Taktsteuerung des DUT übernimmt, wird gestartet, woraufhin dieser das Taktsignal des DUT entsprechend seiner Parametrierung n-mal umschaltet.

5. **Simulation beenden:** Nach dem letzten Starten des Synchronizers werden interne Zwischenspeicher für Konfiguration und Daten, sowie die Schnittstelle zum FPGA freigegeben und die Simulation beendet.

Diese Vorgehensweise ist (abgesehen vom Synchronizer, siehe dazu Abschnitt 4.3.5) der grundlegendste Ansatz um eine FPGA-HiL-Simulation durchzuführen, da alle Signale ungeachtet ihrer Bedeutung von der Simulation zum DUT und umgekehrt durchgeschleift werden. Abgesehen von den oben genannten Einschränkungen lassen sich dadurch beliebige Hardware-Entwürfe an einen Simulator koppeln. Der Preis für diese Generalität liegt in der Menge der zu übertragenden Daten, die in diesem Falle maximal ist. Ein einfaches Verfahren, mit dem die Datenmenge reduziert werden kann, ist das Ereignis-basierte Übertragen der Daten, d.h. nur Ein- bzw. Ausgänge die sich verändert haben werden übertragen. Ein weiteres Verfahren liegt in der Kapselung von Protokollen durch sog. Transaktoren.

4.3.4 Ereignis-basierte Datenübertragung

Das Reduktionspotential der Ereignis-basierten Datenübertragung gegenüber dem oben beschriebenen Verfahren hängt auch von der Arbitrierung des Datenbusses zwischen DUT und Simulationsrechner ab. Bei einem System mit nur einem Bus-Master wird das zu übertragene Datenvolumen V_e zu

$$V_e = \delta(V_b) + S \left\lceil \frac{O}{w_b} \right\rceil \tag{4.1}$$

wobei $\delta(V_b)$ die Anzahl der Änderungen bei den Ausgangswerten, S die Anzahl der Simulationsschritte, O die Anzahl der Ausgänge und w_b die Wortbreite des

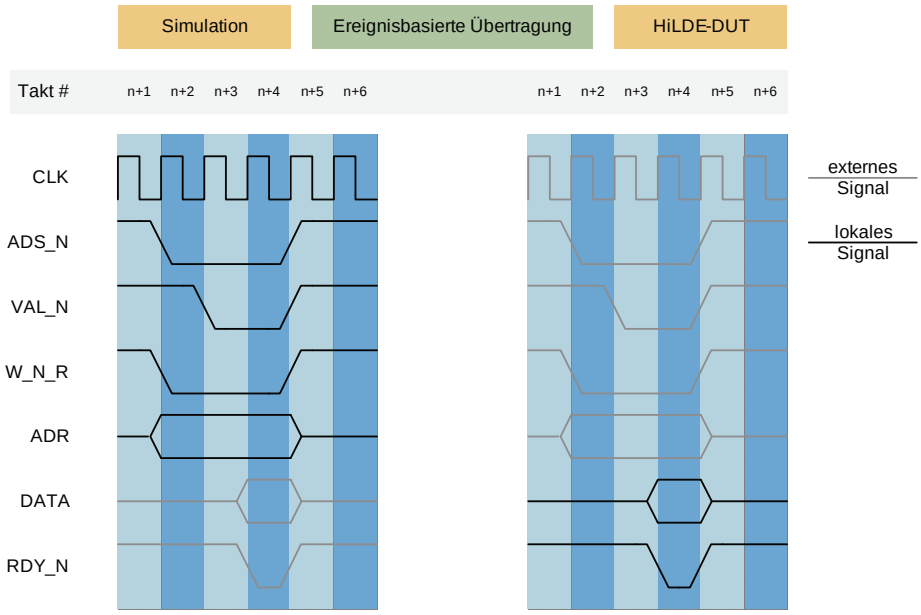


Abbildung 4.5: Signalverläufe bei Standard- und Ereignis-basiertem Übertragungsverfahren.

Datenbusses darstellt. Der Bruchterm beschreibt hier die Notwendigkeit, bei jedem Simulationsschritt zu überprüfen, welche Daten übertragen werden müssen. Bei einem System, bei dem die Datenübertragung von beiden Seiten (DUT und Simulation) initiiert werden kann, entfällt der Bruchterm und es gilt für jeden Fall $V_b \geq V_e$. Ansonsten wird die Datenmenge nur dann reduziert, wenn die Anzahl der Änderungen $\delta(V_b) \leq V_b + S\lceil\frac{O}{w_b}\rceil$, was für jede Anwendung überprüft werden muss.

Abbildung 4.5 zeigt die Signalverläufe bei Standard- und Ereignis-basiertem Übertragungsverfahren für ein einfaches Übertragungsprotokoll. Beim Standardverfahren sind 43 Übertragungen nötig, Ereignis-basiert reduziert sich dies auf 24 (18 Änderungen bei den Signalen zzgl. 6 Abfragen der Änderungen). Für dieses Beispiel ergibt sich also eine Einsparung von 19 Übertragungen entsprechend 44%.

Gleichzeitig erhöht sich der Hard- und Softwareaufwand, der benötigt wird, um Änderungen zu detektieren. In Hardware wird ein n_o Bit breites Register benötigt, in dem die Ergebnisse von n_o zusätzlichen Vergleichern, die Änderungen in den Ausgangsregistern detektieren, gespeichert werden. Dieser Hardwareaufwand hat keine Auswirkungen auf die eigentliche Simulationsdauer, es ist lediglich zu prüfen, ob genügend Ressourcen für dieses Verfahren vorliegen. In Software wird ein zusätzlicher Speicherbereich benötigt, in dem die Eingangsdaten des Simulink-Blockes gespeichert werden, damit eine entsprechende Vergleichsroutine feststellen kann, welche Daten sich geändert haben und an das DUT gesendet werden müs-

sen. Die Ausführung der Vergleichsroutine wirkt sich auf das Zeitverhalten der Simulation aus, da n_i zusätzliche Vergleiche ausgeführt werden müssen.

Die Effizienz der Ereignis-basierten Datenübertragung muss für jedes DUT evaluiert werden. In der Regel werden bei Datenfluss-orientierten Algorithmen, etwa Reglern ohne Steuersignale keine Geschwindigkeitsvorteile erzielt. Ist hingegen die Anzahl der Steuersignale, die sich selten ändern groß, steigt auch der Geschwindigkeitsvorteil gegenüber der Standardübertragung an. Eine wichtige Klasse von Systemen, bei denen viele Steuersignale benötigt werden, sind Busse. Für jede Transaktion auf dem Bus muss ein Protokoll befolgt werden, das in der Regel mindestens zwei Steuersignale (Request und Accept) beinhaltet. Für diesen häufigen Sonderfall wurde das Konzept der Transaktoren entwickelt.

4.3.5 Transaktoren

Wie bei der Ereignis-basierten Übertragung handelt es sich bei den Transaktoren um ein Verfahren zur Reduktion der zu übertragenden Datenmenge. Transaktoren sind funktionale Abstraktionen von Protokoll-basierten Übertragungsverfahren, bei denen die Verarbeitung eines Protokolls auf beiden Seiten der HiL-Simulation (also sowohl in der Simulation als auch im DUT) lokal durchgeführt wird, und nur die notwendigen Daten zwischen beiden Systemen übertragen werden. Abbildung 4.6 zeigt die Struktur einer HiLDE-Simulation mit integrierten Transaktoren. Hierbei übernimmt der Simulations-seitige Transaktor die Funktion der DUTs und der DUT-Seitige Adapter die Funktion der Simulation. Im einfachsten Fall werden nur noch drei Signale, der Übertragungsmodus, die Adresse und die Daten übertragen, alles andere wird von den lokalen Transaktoren abgewickelt.

In Abbildung 4.7 werden die Signalverläufe für das gleiche Protokoll wie in Abbildung 4.5 unter Zuhilfenahme von Transaktoren dargestellt. Das Prinzip folgt dem folgenden Muster:

1. Detektieren eines Zugriffs, Anhalten des DUT.
2. Anhalten der Simulation, wenn der Zugriff gültig ist.
3. Übertragen von Adresse und Modus und (bei Schreibzugriffen) Daten.
4. Starten des Transaktors am DUT, Zugriff wird in Hardware ausgeführt.
5. Daten vom DUT an die Simulation übertragen (bei Lesezugriffen), Simulation fortführen.

Der Einsatz von Transaktoren reduziert die zu übertragende Datenmenge auf drei Zugriffe (Modus, Adresse und Datum), entsprechend einer Reduktion um 93% beim Standardverfahren und $87,5\%$ beim Ereignis-basierten Verfahren. Die Effizienz dieses Verfahrens ist ebenfalls abhängig vom konkreten System bzw.

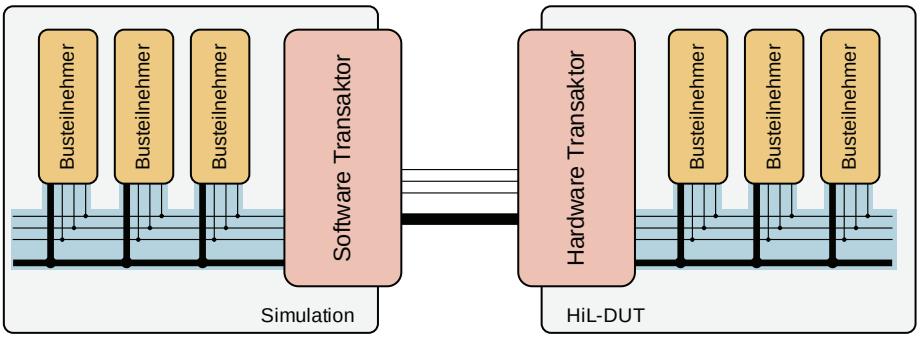


Abbildung 4.6: Transaktor-basierte Datenübertragung für Busse

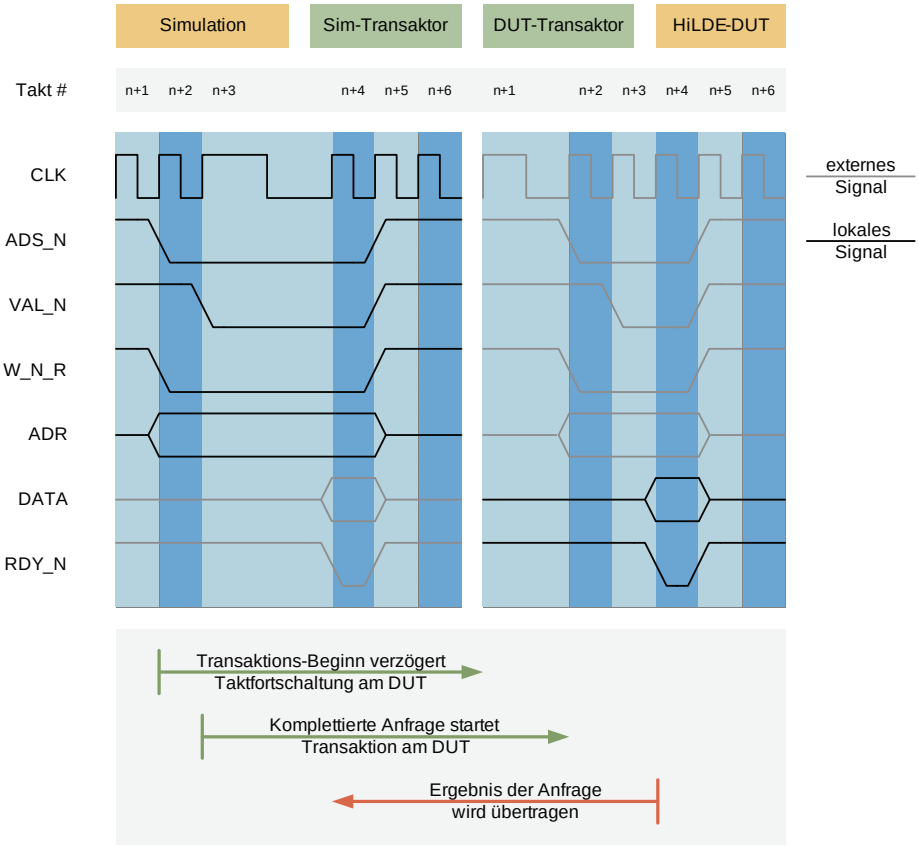


Abbildung 4.7: Signalverläufe für Transaktor-basierte Datenübertragung

vom Protokoll, der Aufwand für die Implementierung steigt mit der Komplexität des Protokolls an. Eine teilautomatische Erzeugung der Transaktoren für Simulation und DUT wird in den Abschnitten vMAGIC (Abschnitt 4.5) und SiLLis (Abschnitt 4.5.3) beschrieben.

4.3.6 Hardwareaufwand

Der Hardwareaufwand für den HiLDE-Adapter wächst mit der Anzahl der zu überwachenden Schnittstellen. Abbildung 4.8(a) zeigt diesen Zusammenhang anhand eines einfachen Beispiels mit n Zählern á 32 Bit, deren Steuereingänge (aktivieren, aufwärts/abwärts, zurücksetzen) über einen Buszugriff gesteuert werden. Die Anzahl der Gatteräquivalente entspricht im einfachsten Fall (ein Eingang, ein Ausgang) etwa der von zwei 32 Bit Zählern, wächst jedoch im Wesentlichen logarithmisch an, was auf das logarithmische Wachstum der Datenschnittstelle zurückzuführen ist. In Abbildung 4.8(b) wird der Unterschied zwischen Standard- und Ereignis-basierter Übertragung dargestellt, der in allen Fällen weniger als 10% beträgt.

4.3.7 Leistungsfähigkeit

Die Leistungsfähigkeit des HiLDE Systems wird über die sog. Simulationsfrequenz, also die Anzahl der simulierten Takte pro Sekunde bestimmt. Diese ist abhängig von vielen Faktoren, angefangen bei dem verwendeten Rechner über das Betriebssystem bis hin zum verwendeten FPGA und den Ein- und Ausgängen des DUT. Um trotzdem eine allgemeine Aussage über die Leistungsfähigkeit machen zu können, wird die Simulationsfrequenz zunächst unabhängig von Plattform und Simulator modelliert:

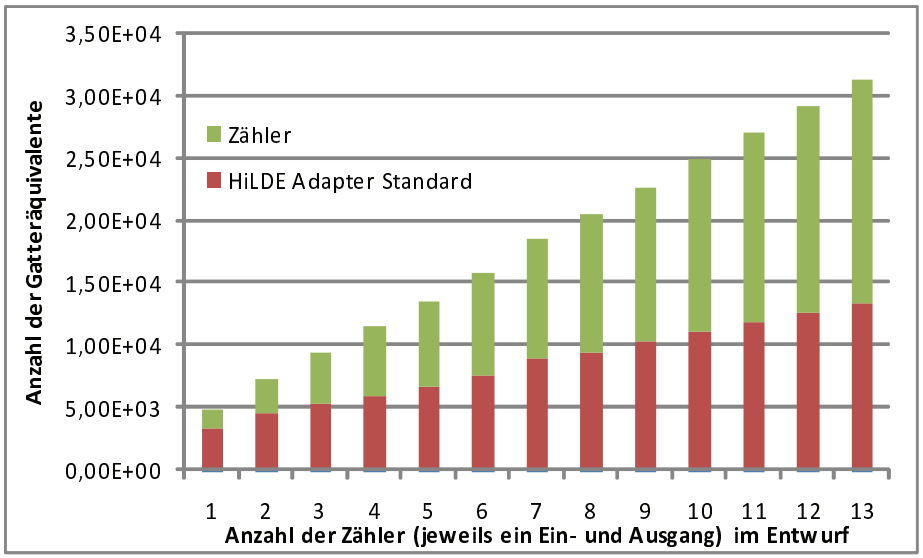
$$F_{sim} = (t_{lesen} + t_{schreiben} + t_{DUT})^{-1} \tag{4.2}$$

Hierbei sind t_{lesen} und $t_{schreiben}$ die Zeit für die benötigten Lese- und Schreibzugriffe für die Ein- und Ausgangsspeicher des Systems, $t_{ausfuehren}$ entspricht einem Schreibzugriff (zum Starten) und der Latenz des DUT. Die Latenz entspricht dem sog. kritischen Pfad und darf nicht unterschritten werden, da ansonsten die Ergebnisse der Hardware fehlerhaft werden können.

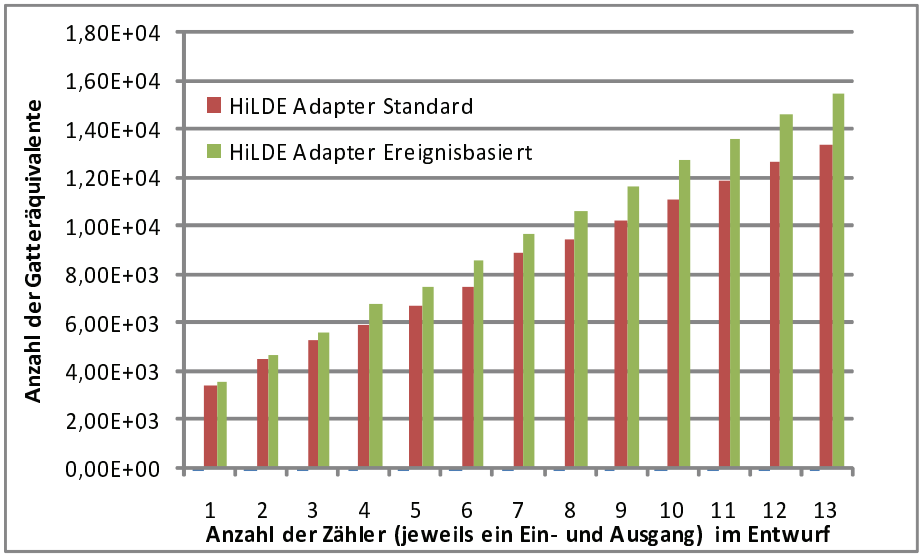
Unter Berücksichtigung der Leistungsfähigkeit verschiedener Übertragungsverfahren auf dem PCI-Bus ergeben sich für die maximal erreichbare Simulationsfrequenz die in Abbildung 4.9 gezeigten Werte.

4.4 Das HiLDE-System für Online Simulationen

Nachdem mit Hilfe einer HiLDE-basierten Simulation sicher gestellt wurde, dass der Benutzerentwurf auch in Hardware funktional korrekt ist, kann dieser prototypisch in die reale Regelschleife integriert werden. Speziell in diesem Entwicklungsstadium sind eventuell noch nicht alle Parameter festgelegt, und es ist wünschenswert, diese komfortabel testen zu können. Darüber hinaus ist es wichtig, die Ein- und Ausgangssignale des Entwurfes zu beobachten: einerseits können



(a) Hardwareaufwand HiLDE Adapter mit n Ein- und n Ausgängen



(b) Vergleich Hardwareaufwand Standard- und Ereignis-basierte Übertragung

Abbildung 4.8: Hardwareaufwand für den HiLDE-Adapter am Beispiel eines DUT mit n Zählern

so ggf. Fehler in einer Simulation nachgestellt werden, andererseits können Messungen in der realen Regelschleife ohne zusätzliche Messgeräte (z.B. einen Logikanalysator) durchgeführt werden. Das Programm HiLDEGART (HiLDE for Generic Active Realtime Testing) [14, 8] wurde entwickelt, um genau diesen Anforderungen zu entsprechen. Das HiLDGART System basiert auf einer ähnlichen Hardware/Software-Kombination wie HiLDE, allerdings ist hier die (ebenfalls automatisch erzeugte) Hardware im Wesentlichen als Beobachter ausgelegt. Um die-

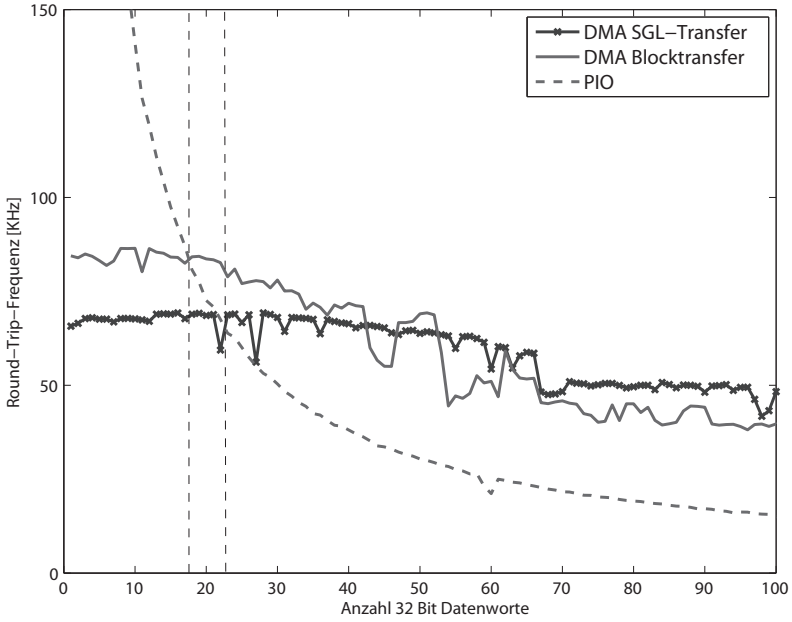


Abbildung 4.9: Maximale Simulationsfrequenz F_{sim} (hier Round-Trip Frequenz) für variierende Zahlen für Ein- und Ausgänge sowie Übertragungsverfahren

se Funktion in einem System erfüllen zu können, das sich im laufenden Betrieb befindet, werden verschiedene Abtasteinheiten erzeugt, die sowohl langsam abgetastete Echtzeitdaten zur Beobachtung als auch schnell abgetastete Daten zur Analyse des Systems liefern. Dazu können Einheiten zur Reduktion des zu kommunizierenden Datenaufkommens, Einheiten zur Parametrierung eines Designs und dedizierte Triggereinheiten integriert werden. Daneben besteht HiLDEGART aus einer graphischen Oberfläche, die eine komfortable Parametrierung einerseits und die Beobachtung der Ein- und Ausgänge des Entwurfes andererseits ermöglicht. Ein wichtiges Merkmal des HiLDEGART Systems ist, dass das zu beobachtende System *nicht verändert wird*, d.h. das Kommunikationsverhalten wird in keinem Falle beeinflusst. Dies ist z.B. dann wichtig, wenn eingebettete Prozessoren zum Einsatz kommen, deren Zeitverhalten sich durch das Einfügen von speziellem Programmcode zur Beobachtung verändern würde.

Die Erzeugung der oben skizzierten Schnittstellen ist ein sehr gleichförmiger Prozess, der sich im Wesentlichen automatisieren lässt. Mit Hilfe der in Abschnitt 4.5 vorgestellten vMAGIC Bibliothek lässt sich diese Automatisierung erreichen. Zusätzlich können so die Konfigurationsdateien, die das Aussehen der HiLDEGART-Oberfläche bestimmen, automatisch erzeugt werden.

Im Folgenden werden zunächst der Aufbau von Hardware und Software für den Einsatz im Bereich Datenflussorientierter Algorithmen beschrieben. Im Anschluss

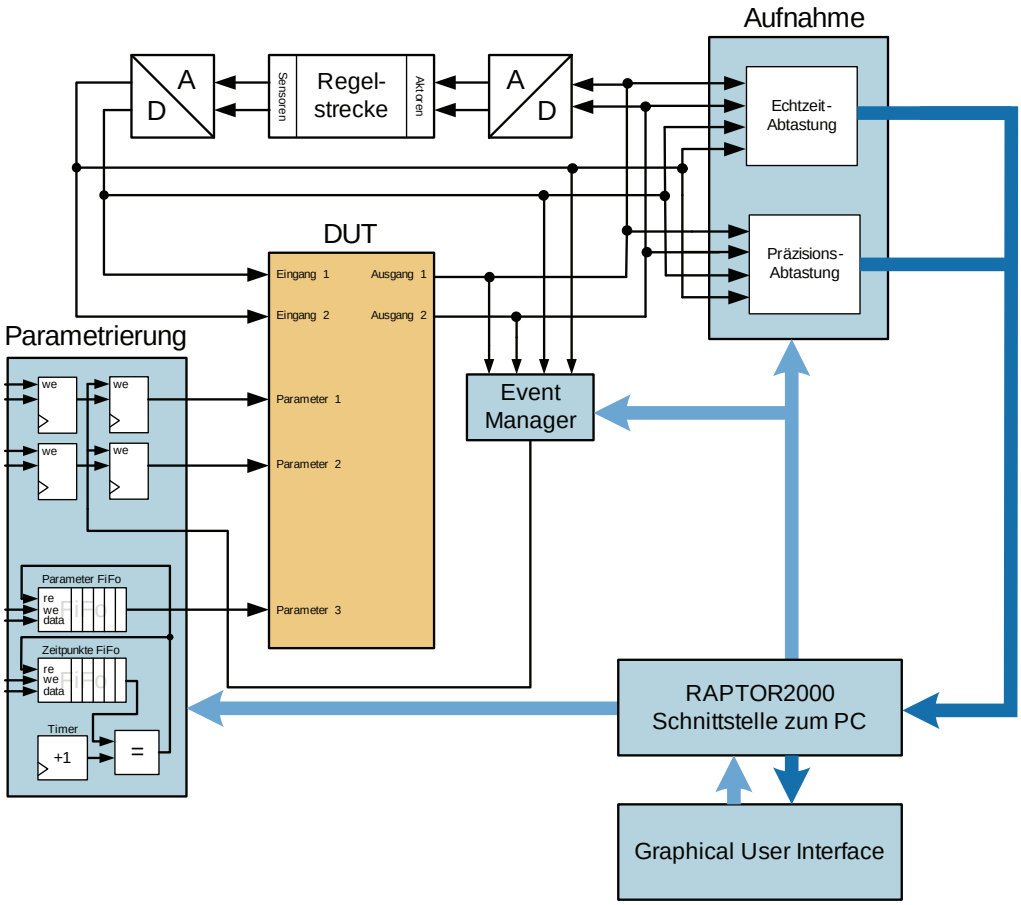


Abbildung 4.10: Die HiLDEGART Hardwareschnittstelle

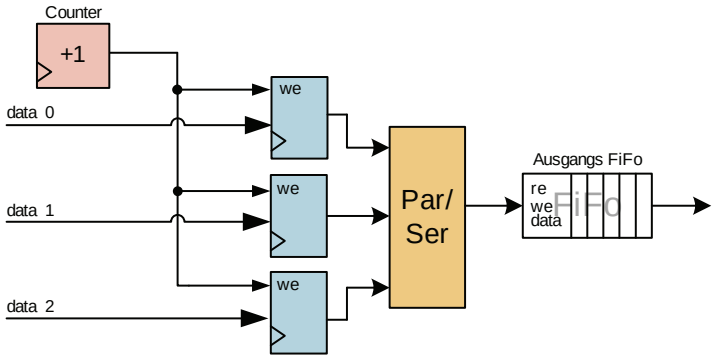
wird eine Erweiterung für die Beobachtung Protokoll-basierter Datenübertragungen vorgestellt.

4.4.1 HiLDEGART Wrapper

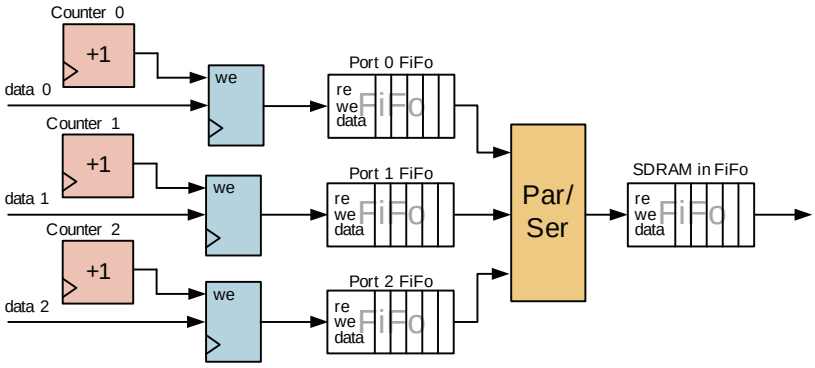
Der HiLDEGART Wrapper (Abbildung 4.10) setzt sich aus denn Subsystemen Parametrierung, Triggerung und Aufnahme zusammen, deren Funktionalität und Interaktion im Folgenden beschrieben werden. Alle Hardwaresysteme werden, wie schon bei HiLDE über ein Programm auf Basis von vMAGIC (Abschnitt 4.5 automatisch erstellt.

Aufnahme

Die Aufnahmeeinheit ist aus zwei Subsystemen aufgebaut; die Echtzeit-Abtastung dient der Erfassung von Ein- und Ausgangsdaten mit einer sehr niedrigen Abtstrate ($<< 1kHz$, vom Benutzer wählbar), die in der graphischen Benutzeroberfläche



(a) Echtzeitabtastung



(b) Präzisionsabtastung

Abbildung 4.11: Die HiLDEGART Abtastsysteme

zur Laufzeit dargestellt werden. Demgegenüber dient die Präzisionsabtastung der Speicherung von Ein- und Ausgangsdaten mit sehr hoher Abtastrate (bis zu 25 MHz), die erst nach Beendigung des Tests in der Oberfläche angezeigt werden. Die Kombination beider Verfahren erlaubt einerseits die Anzeige relevanter Daten in Echtzeit, andererseits die nachträgliche Untersuchung des Tests auf Basis sehr hochauflösender Abbilder der Ein- und Ausgänge. Diese können insbesondere wieder in die Simulation integriert werden, so dass eine Simulation mit realen Eingangsdaten stattfinden kann; diese Vorgehensweise erhöht die Realitätsnähe der Simulation und ermöglicht die Suche nach Effekten, die in der Modell-basierten Simulation nicht auftreten.

Beide Abtastblöcke (Abbildung 4.11) bestehen aus einer Umtasteinheit, Zwischenspeichern für die Ein- und Ausgangsdaten und einem Ausgangsfifo. Die Abtastraten der Umtasteinheiten sind konfigurierbar, bei der Präzisionsabtastung für jeden Kanal separat. Eine genaue Charakterisierung beider Abtastsysteme wird in Abschnitt 4.4.1 vorgenommen.

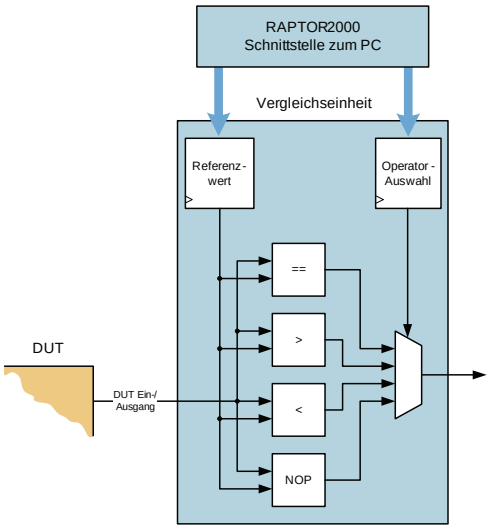
Parametrierung

Bei der Implementierung von Reglern oder ähnlichen, datenflussorientierten Systemen wie Filtern ist es häufig wünschenswert, dass die Parameter dieser Systeme, z.B. Filterkoeffizienten oder Zeit- und Proportionalitätskonstanten im laufenden Betrieb veränderlich sind. Gründe für solche Anpassungen können beispielsweise eine Verschiebung des Arbeitspunktes einer Regelstrecke im laufenden Betrieb oder aber eine ungenaue Modellierung der Regelstrecke sein. Um in diesen Situationen von außen eingreifen zu können, werden die in Frage kommenden Parameter in Speichern vorgehalten, die an die Kommunikationsschnittstelle zum Host angeschlossen werden, so dass diese über die HiLDEGART GUI zugreifbar sind.

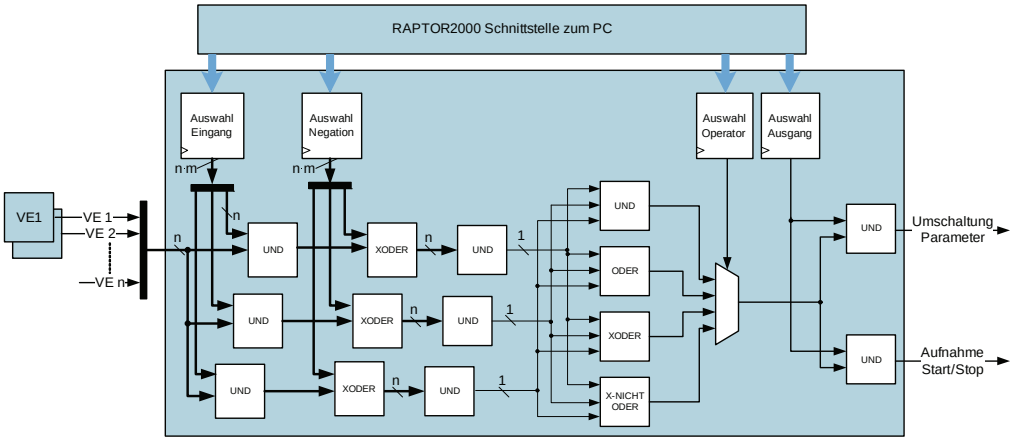
HiLDEGART sieht zwei Verfahren zur Umschaltung vor: bei der Online-Parametrierung werden die vom Benutzer vorgegebenen Werte sobald als möglich (engl. ASAP, As Soon As Possible) an das Benutzersystem weitergeleitet. Bei der Werte-basierten Parametrierung wird die Umschaltung auf einen neuen Wert durch ein Ereignis (siehe Trigger) ausgelöst. Für beide Verfahren werden die Parameterspeicher durch zwei in Reihe geschaltete Register implementiert, wobei das erste Register vom Lokalbus beschrieben wird, während das DUT-seitige Register den neuen Wert erst, wie oben beschrieben, später übernimmt. Wichtigste Eigenschaften dieser Implementierung sind die Möglichkeit zur synchronen Umschaltung mehrerer Parameter zu einem vorgegebenen Zeitpunkt oder auf Basis eines Ereignisses.

Trigger

Die in den vorigen Abschnitten beschriebenen Funktionen zur Umschaltung von Parametern benötigen unterschiedliche Auslöser (engl. Trigger) um den Zeitpunkt der Umschaltung zu bestimmen. Bei der online Parametrierung wird dieser über die Benutzeroberfläche bestimmt (direkt nach der Eingabe eines neuen Wertes, bzw. bei der gleichzeitigen Umschaltung mehrerer Werte nach Drücken eines entsprechenden Knopfes), bei der Werte-basierten Umschaltung muss der Zeitpunkt in der Hardware bestimmt werden. Dazu wurde die Trigger-Einheit entworfen, die aufgrund von relationalen Vergleichen zwischen Referenzwerten und den Ein- und Ausgängen des Reglers Ereignisse auslösen kann. Die Trigger Einheit setzt sich aus parametrierbaren, relationalen Vergleichseinheiten einerseits und Kombinationseinheiten andererseits zusammen. Für jeden Ein- und Ausgang wird eine Vergleichseinheit implementiert, die die Operationen $> x$, $< x$ und $= x$ unterstützen, wobei x eine über den Lokalbus übertragene Konstante ist. In den Kombinationseinheiten werden die Ergebnisse der Vergleichseinheiten mit Hilfe Bool'scher Operatoren zusammengefasst, so dass sich für die Umschaltung von Parametern Bedingungen der folgenden Form definieren lassen: (Signal1 $>$ x) UND NICHT (Signal2 $<$ y)



(a) Vergleichseinheit



(b) Kombinationseinheit

Abbildung 4.12: Das HiLDEGART Trigger System

In Abbildung 4.10 ist der schematische Aufbau von Vergleichs- und Kombinationseinheit dargestellt. Die Kombinationseinheit ist so aufgebaut, dass bei einem System mit n Ein- und Ausgängen, also auch n Vergleichseinheiten, die Anzahl der Teilbedingungen bis zu m betragen kann. Dazu werden die Ausgänge der Vergleichseinheiten zu einem Bus zusammengefasst, aus dem über einen *und*-Block einzelne Signale ausgewählt werden, die anschließend über einen *exklusiv oder*-Block (im Bild XODER) negiert werden können. Die so ausgewählten und ggf. negierten Signale werden dann über einen *und*-Block zu einem 1-Bit Signal reduziert, woraufhin verschiedene dieser Signale über einen Wählbaren Operator Verknüpft werden. Über die Auswahl eines Ausgangs kann mit der zuvor überprüften Bedingung entweder die Umschaltung von Parametern, oder aber der Start- bzw.

Stop einer Aufnahme erreicht werden. Diese Start- Stop Funktion ist insbesondere dann wichtig, wenn ein System erst nach einer längeren Initialisierungsphase bzw. in einem bestimmten Betriebsbereich überwacht werden soll.

Erweiterte Datenspeicherung

Um die maximale Abtastrate des HiLDEGART Systems zu bestimmen, müssen neben dem Bussystem zur Übertragung der Daten vom FPGA zum PC weitere Faktoren untersucht werden. Neben den verschiedenen Übertragungsmodi, die einige Bussysteme unterstützen, und dem Programm zur Verarbeitung und Darstellung der Daten ist vor allem das Betriebssystem zu berücksichtigen. Bei einem Multitasking Betriebssystem wie Microsoft Windows wird die Verteilung der zur Verfügung stehenden Rechenleistung vom sog. Scheduler übernommen. Dieser weist allen Benutzerprogrammen und Hintergrunddiensten einen Teil der Rechenzeit zu. Daraus folgt, dass ein Benutzerprogramm wie HiLDEGART immer wieder unterbrochen werden kann, was einen Überlauf der FIFOs an den Ein- und Ausgängen und damit unvollständige Daten im Speicher verursachen kann. Dieses Verhalten des Betriebssystems lässt sich nicht unterbinden, womit die Länge der Zeitscheibe t_s (die Zeiteinheit, die der Scheduler einem Programm zuweisen kann) zusammen mit der FIFO-Tiefe d_f zum limitierenden Faktor für die Datenrate R_{max} wird.

$$R_{max} = \min\left(\frac{d_f}{t_s}, R_{Bus}(N_{d_B})\right) \tag{4.3}$$

Dabei stellt $R_{Bus}(d_B)$ die Datenrate des Bussystems in Abhängigkeit von der Anzahl der am Block zu übertragenden Werte d_B dar.

In der Regel gilt $R_{Bus}(d_B) >> \frac{d_f}{t_s}$, was die maximale Datenrate auf relativ kleine Werte beschränkt: bei einer Zeitscheibe von $50ms$ und einer FIFO-Tiefe $d_f = 2048$ gilt $R_{max} \approx 40KHz$, wobei weder Verzögerungen durch die Datenübertragung noch durch die Benutzeroberfläche in die Berechnung einbezogen wurden. Messungen auf einem PC mit einem 3,2 GHz Pentium 4 ergaben für diese Konfiguration eine maximale Abtastrate von ca. 10 KHz. Um die Abtastrate zu erhöhen und gleichzeitig Daten mit nur geringer Latenz in der Benutzeroberfläche darstellen zu können, wurde eine zusätzliche Speicherschnittstelle implementiert: Über eine weitere Umtasteinheit werden die Daten in einen SDRAM Speicher geschrieben, der direkt durch das FPGA zugreifbar ist. Auf diese Weise ist die Aufnahmelänge durch die Kapazität des Speichers begrenzt, aber die maximale Abtastrate wird durch die Bandbreite der Speicherschnittstelle begrenzt:

$$R_{max} \leq F_{\text{SDRAMI}} w_{\text{SDRAMI}} \tag{4.4}$$

Wobei F_{SDRAMI} die Geschwindigkeit der SDRAM Schnittstelle und w_{SDRAMI} die Anzahl der Worte, die parallel geschrieben werden können, bezeichnet. Im

Falle des RAPTOR Moduls DB-V2 lässt sich der mit einem 64 Bit breiten Bus angebundene Speicher mit maximal 133 MHz betreiben, was bei einer Wortbreite von 32 Bit pro abzutastendem Wert die Datenrate in diesem einfachen Modell auf $R_{max} = 266$ MSamples/s erhöht. Tatsächlich liegt die Abtastrate bei ca. 200 MSamples/s, da zu den Schreiboperationen von der SDRAM-Schnittstelle noch weitere Operationen eingefügt werden.

Die im SDRAM gespeicherten Daten werden nach Ende der Aufnahme an den PC übertragen, so dass während der Aufnahme Daten mit geringer Auflösung visualisiert werden, und im Anschluss daran Daten mit einer sehr hohen zeitlichen Auflösung für weitere Analysen zur Verfügung stehen.

4.4.2 Transaktoren und Filter

Die oben beschriebene Hardwareschnittstelle eignet sich für viele Anwendungen, bei denen die Datenkommunikation keine Protokollverarbeitung benötigt, also beispielsweise bei einzelnen Reglern, Filtern usw. Bei komplexeren Systemen, die aus mehreren Subsystemen und einer Kommunikationsinfrastruktur bestehen, ist das reine Aufzeichnen der Daten- (und Steuer-) Kanäle wenig sinnvoll. Ein Benutzer ist bei der Beobachtung eines Datenbusses im Wesentlichen an der übertragenen Information, den kommunizierenden Knoten und am Zustand des Busses interessiert, die eigentliche Protokollverarbeitung ist nebensächlich. Diese Vorgehensweise hat zwei Vorteile gegenüber der Übertragung der reinen Datenkanäle: Einerseits ergibt sich durch eine automatische Extraktion dieser Informationen aus dem Busprotokoll eine Reduktion des an die graphische Oberfläche zu übertragenen Datenvolumens (siehe Abschnitt 4.3.5). Andererseits reduziert sich die Anzahl der zu implementierenden Schnittstellen, da über die Buskommunikation in der Regel mehrere Subsysteme gleichzeitig beobachtet werden können. Zusätzlich kann dieses Verfahren auch in Bereichen eingesetzt werden, in denen die Implementierung einer direkten HiLDEGART-Schnittstelle nicht möglich ist. Dies ist z.B. im Bereich der dynamischen Rekonfiguration der Fall, da hier in der Regel alle verfügbaren Kommunikationskanäle für die Kommunikation zwischen den rekonfigurierbaren Subsystemen belegt werden.

Der Aufbau der Transaktoren erfolgt analog zu dem der HiLDE-Transaktoren in Abschnitt 4.3.5 und der Benutzer wird bei der Entwicklung von dem Transaktorgenerator SiLLis (siehe Abschnitt 4.5.3) unterstützt. Zusätzlich zu der schon beschriebenen Transaktor-Funktionalität kommen hier noch Filter-Funktionen zum Einsatz, die das Datenvolumen weiter reduzieren. SiLLis ermöglicht den Aufbau beliebiger wertebasierter Filter, so dass beispielsweise nach Quell- oder Zieladresse oder bestimmten Werten gefiltert werden kann. Diese Funktion kommt bei der Beobachtung rekonfigurierbarer Systeme zum Einsatz, z.B. um ausschließlich Rekonfigurationsanweisungen zu beobachten.

4.4.3 Software

Die graphische Benutzeroberfläche HiLDEGART (engl. GUI, Graphical User Interface) stellt dem Benutzer alle Funktionen des HiLDEGART-Systems wie Parametrierung, Beobachtung, Ereignisse usw. in einer integrierten Umgebung zusammen. Die Oberfläche wird für jedes HiLDEGART-Projekt dynamisch zur Laufzeit auf Basis einer XML-Beschreibung des ebenfalls automatisch erzeugten HiLDEGART-Wrappers aufgebaut und lässt sich auf die Bedürfnisse des Benutzers anpassen. Im Folgenden werden die wesentlichen Funktionen kurz dargestellt, für tiefer gehende Informationen wird auf das HiLDEGART-Benutzerhandbuch verwiesen.

- **Projekte:** Alle Konfigurationseinstellungen (Hardware-Adressen, Signalnamen. Zahlformate, Standardwerte usw.) werden in einer XML Datei abgelegt, die zum Teil automatisch erzeugt werden kann. Alle Parameter, die sich innerhalb der Benutzeroberfläche variieren lassen (Abtastraten, Triggerkonfigurationen, Graphen) werden ebenfalls hier gespeichert.

- **Parameter:** Alle DUT-Parameter lassen sich über Drehinstrumente und direkte Eingabe im Gleitkomma-Format angeben, die Umrechnung in das jeweilige Zielformat erfolgt automatisch über die in der XML Datei gespeicherte Konfiguration.

- **Graphen:** Für die Darstellung kontinuierlicher Werte werden Graphen verwendet, die vom Benutzer konfiguriert werden können. Es lassen sich mehrere Kurven mit verschiedenen Farben und bis zu zwei Werteachsen zur Laufzeit darstellen.

- **Weitere Darstellungen:** Zusätzlich zu den Graphen werden Ereignisbasierte Werte (Trigger, Transaktoren) in Text-basierten Tabellen bzw. Listen dargestellt. Für die Visualisierung dynamischer Rekonfiguration wurde eine Schnittstelle zu dem Programm ReBit [35] implementiert, dass speziell für den Entwurf und die Visualisierung dynamisch rekonfigurierbarer FPGA Systeme entwickelt wurde.

- **Trigger:** Die Oberfläche für die Konfiguration der Triggerfunktionen wird ebenfalls dynamisch erzeugt und erlaubt die Definition der Triggerfunktionen über Konstanten und Bool'sche Operatoren. Die Trigger können als Auslöser für das Umschalten von Parametern und für das Starten der Präzisions-Abtastung verwendet werden. Abfolgen von Triggerkonfiguration können über ein spezielles Dateiformat angegeben werden.

Die HiLDEGART Oberfläche wurde mit Hilfe des Plattform-unabhängigen Qt-Paketes erzeugt und nutzt zusätzlich das QWT Plugin [110] zur Darstellung der

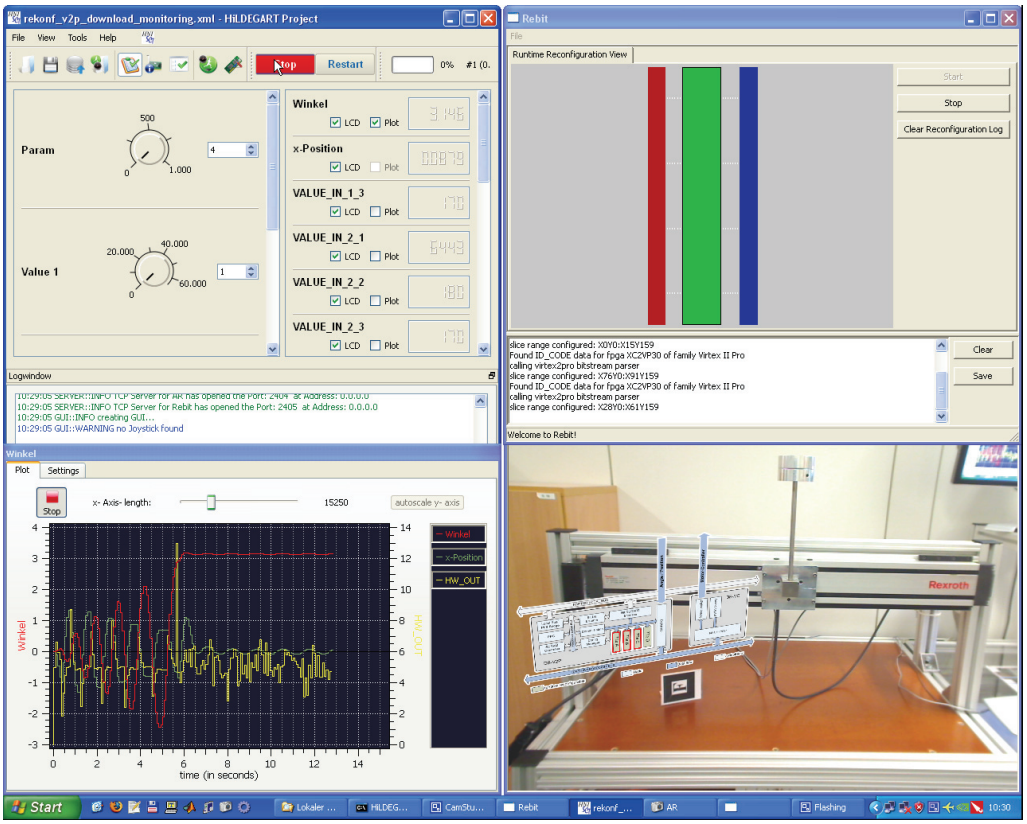


Abbildung 4.13: HiLDEGART am inversen Pendel (links), darauf aufbauend das SiLLis-basierte REBIT zur Überwachung der Rekonfiguration (rechts oben) sowie eine Augmented Reality unterstützte Videoaufnahme des realen Pendelsystems (rechts unten)

Graphen. Eine Beispielkonfiguration der Oberfläche mit Graphen ist in Abbildung 4.13 zu sehen.

Beobachtung verteilter Systeme

Für sehr große FPGA-basierte Systeme kann es vorkommen, dass die FPGA Trägersysteme, z.B. RAPTOR-XPress Rapid Prototyping Boards auf mehrere PCs verteilt werden. Wenn diese Systeme bzw. die darauf installierten FPGAs über eine Leistungsfähige Kommunikationsinfrastruktur Daten austauschen können. Ein solches System ist der RC^2 [18], bei dem 64 FPGAs verteilt auf 16 RAPTOR-XPress Boards in 8 PCs über RocketIOs kommunizieren können. In einem solchen Cluster sind leistungsfähige Werkzeuge zur Initialisierung, Beobachtung und Wartung nötig. Ein Teil dieser Aufgaben kann mit dem HiLDEGART Werkzeug abgedeckt werden. Dazu wurde HiLDEGART um eine Netzwerkschnittstelle erweitert, die es ermöglicht mehrere PCs mit RAPTOR-XPress Boards zu kontaktieren.

4.5 vMAGIC - VHDL Manipulation and Generation Interface

Die in den vorangegangenen Abschnitten zu HiLDE 4.3 und HiLDEGART 4.4 beschriebenen Kommunikationsschnittstellen sind im Wesentlichen gleichförmig. Verschiedene HiLDE oder HiLDEGART-Projekte unterscheiden sich im Wesentlichen durch die Anzahl und die Wortbreite der angeschlossenen Speicherelemente. Trotz dieser Gleichförmigkeit ist die manuelle Erzeugung dieser Schnittstellen so zeitaufwändig wie fehleranfällig. Um eine automatische Erzeugung dieser Schnittstellen zu ermöglichen, wurde die vMAGIC Bibliothek entwickelt, die sich verschiedener Techniken aus dem Bereich der automatischen Quelltext-Erzeugung bedient. Der Oberbegriff *automatische Quelltext-Erzeugung* (engl. Automatic Code Generation, ACG) fasst diverse Techniken zusammen, die den Benutzer bei der Programmierung im Allgemeinen unterstützen. Während einige dieser Techniken lediglich die Schreibgeschwindigkeit erhöhen, indem z.B. Vorlagen (engl. Templates) eingesetzt werden, existieren auch Systeme, die komplexere Aufgaben automatisieren können. Werkzeuge, die zu letzterer Kategorie gehören, arbeiten auf syntaktischer Ebene, d.h. auf Basis der Grammatik der zugrunde liegenden Programmiersprache. Dem Benutzer werden Vorschläge gemacht, wie das aktuelle Konstrukt vervollständigt werden kann (automatische Vervollständigung, engl. Automatic Code Completion, ACC). Diese Vorschläge können dann, falls möglich und eindeutig, automatisch an die bisher im Quelltext bzw. Konstrukt verwendeten Namen angepasst werden. Die resultierenden Abschnitte sind auf jeden Fall syntaktisch korrekt, die zugrunde liegende Semantik wird jedoch vollständig vom Benutzer definiert. Diese Techniken lassen sich bei jeder Art von Programmierung einsetzen, wenn eine wohl definierte Grammatik zu Grunde liegt.

Eine Technik, welche die Erzeugung von Quelltext auf einer höheren Abstraktionsebene unterstützt, ist die Quelltext-Transformation. Solche Ansätze kommen beispielsweise im Bereich des Modell-basierten Entwurfes zum Einsatz. Hier wird eine Komponente in einer Quellsprache (häufig eine grafische Repräsentation: die Quell-Sprache) definiert, die dann unter Annahme bestimmter Randbedingungen automatisch in die Ziel-Sprache übersetzt wird. Im Rahmen dieses Übersetzungsprozesses wird oft eine abstrakte Zwischenrepräsentation (z.B. ein Datenflussgraph, DFG) eingesetzt, auf der Ziel-Sprachen-spezifische Optimierungen, wie zum Beispiel das Erkennen und Ausnutzen von parallelen Operationen, durchgeführt werden. Der Vorteil bei dieser Vorgehensweise liegt in der hohen Abstraktionsebene der Quell-Sprache: im Idealfall führt eine hohe Abstraktionsebene zu einer höheren Geschwindigkeit sowohl bei der Erzeugung des Quellcodes, als auch bei Anpassungen des Quellcodes. Zusätzlich können Freiheitsgrade in der abstrakten Repräsentation für eine automatische Untersuchung von Realisierungsalternativen (engl. Design Space Exploration) ausgenutzt werden.

Im Folgenden wird ein Ansatz untersucht, dessen Ziel es ist, Quelltext in der Hardware-Beschreibungssprache VHDL zu erzeugen. Beispiele für ähnliche Systeme mit grafischer Eingabe sind der Xilinx System Generator [156] und Synplify DSP [126]. Andere Systeme wie zum Beispiel DWARV [158] verwenden C oder C-ähnliche Sprachen als Ausgangspunkt. Im Unterschied zu diesen Beispielen basiert der vorgestellte Ansatz auf VHDL als Quell- und Ziel-Sprache und dient der Automatisierung komplexer, sich wiederholender Aufgaben in der Hardware-Entwicklung, insbesondere bei der Erzeugung der oben beschriebenen Schnittstellen für HiLDE und HiLDEGART.

4.5.1 vMAGIC - Architektur und Funktionalität

Um eine Zwischenrepräsentation eines beliebigen Quelltextes zu erzeugen, verwendet man in der Informatik eine Kombination aus einem lexikalischen Analysator (engl. lexical analyzer, lexer) und einem syntaktischen Analysator (engl. Parser). Der Lexer zerlegt den Quelltext in einen Datenstrom aus atomaren Symbolen (engl. token) wie reservierte Worte oder Konstanten, der dann vom Parser interpretiert wird. Der Parser, der aus einer Grammatik erzeugt wurde, überprüft die syntaktische Korrektheit und erzeugt eine Zwischenrepräsentation des Quelltextes, den sog. Syntaxbaum. Da der Syntaxbaum bedingt durch die textuelle Beschreibung sehr viel Redundanz enthält, wird aus dem Syntaxbaum unter Verwendung von Umschreibregeln (engl. rewrite rules) ein abstrakter Syntaxbaum (engl. Abstract Syntax Tree, AST) erzeugt, der sich effizienter manipulieren lässt. Im Falle von VHDL enthält der AST alle Informationen über Bibliotheken, Ein- und Ausgänge des Entwurfes, die eigentliche Implementierung und ggf. eine Konfiguration, so dass alle Manipulationen direkt im AST durchgeführt werden können. Auf diese Weise kann einfach sichergestellt werden, dass der später erzeugte Quelltext syntaktisch korrekt ist. Für die Erzeugung eines Parser existieren Werkzeuge wie ANTLR [100] oder JavaCC [75], die basierend auf einer entsprechenden Grammatik automatisch sowohl Lexer als auch Parser erzeugen. Um den manipulierten AST wieder in einen lesbaren Quelltext zu verwandeln, wird zusätzliche Programmlogik benötigt. Da ANTLR für diese Aufgabe eine sehr gute Unterstützung über die sog. String Templates bietet, wurde vMAGIC (VHDL Manipulation and Generation Interface) [13] mit Hilfe von ANTLR entwickelt.

Abstrakt betrachtet stellt vMAGIC die folgende Funktionalität zur Verfügung:

- **VHDL-Lexer/Parser:** Der auf Basis von ANTLR 3.1 erzeugt Parser akzeptiert VHDL'93 Konstrukte und erstellt aus beliebigen VHDL Dateien einen für Manipulationen optimierten AST. Die Optimierungen beziehen sich vor allem auf die Erzeugung sinnvoller Teilbäume, die das Erstellen der sog. Meta-Klassen vereinfachen.

- **VHDL Manipulation/Erzeugung:** Der Kern von vMAGIC sind die sog. Meta-Klassen, die Funktionen auf dem AST definieren. Die Meta-Klassen werden jeweils für ein VHDL-Konstrukt (z.B. eine Zuweisung) definiert und gruppieren Funktionen auf diesen Konstrukten. So kann etwa eine Zuweisung erzeugt und in den AST eingefügt werden, oder aber ein bestehendes Register kann mit einem neuen Ausgangssignal verbunden werden.

  Es wurden Metaklassen [17] sowohl für die grundlegenden Sprachkonstrukte als auch für Konstrukte einer höheren Abstraktionsebene erzeugt. Durch die Einführung von Klassen wie *Register* oder *LocalBus* können komplexe Aufgaben wie das erstellen des HiLDE-Wrappers sehr effizient bewältigt werden.

- **Informationskanal:** Um dem Benutzer die Möglichkeit zu geben, weitere Informationen (Informationen, die nicht in VHDL codiert sind) an vMAGIC-basierte Programme weiterzugeben, wurde ein Kommunikationskanal auf Basis der VHDL Kommentare entworfen. Alle Kommentare, die mit einem bestimmten Kennzeichen markiert wurden, werden vom vMAGIC Parser automatisch den entsprechenden VHDL-Sprachkonstrukten zugeordnet und können aus dem AST gelesen werden. So können etwa bei HiLDE Informationen über Zahlformate direkt in der VHDL Datei codiert werden.

- **Quelltexterzeugung:** Mit Hilfe der sog. String-Templates kann jeder mit vMAGIC erstellte Syntaxbaum als lesbarer VHDL-Quelltext ausgegeben werden, und so von jedem VHDL-basiertem Werkzeug weiter verarbeitet werden.

Im Folgenden wird die Erzeugung neuer Konstrukte bzw. die Manipulation bestehender Konstrukte genauer beschrieben.

Erzeugung neuer Konstrukte

Für die Manipulation des AST wurden Meta-Klassen entwickelt, die als Schnittstelle zu allen benutzerdefinierten Applikationen dienen. Meta-Klassen enthalten bzw. erzeugen einen AST der VHDL Konstrukte und dienen der Repräsentation von VHDL-Konstrukten im Programmcode des Benutzers. Objekte dieser Klassen können auf zwei Wegen erzeugt werden:

- basierend auf einem AST: Wenn bereits ein Quelltext in Form eines AST vorhanden ist, so kann mit Hilfe von Suchfunktionen ein bestimmtes VHDL Konstrukt anhand von Typ oder Name gesucht werden. Auf diese Weise können beispielsweise alle Eingänge eines Entwurfes oder die Definition eines Signals mit dem Namen *next_reg_1* gesucht werden. Dies dient insbesondere der Manipulation bestehender Quelltexte, z.B. um ein Signal, das in einer Hierarchie verborgen ist, auf die oberste Ebene zu führen.

- basierend auf Vorlagen: um neue VHDL Konstrukte in einen AST einzufügen, müssen diese im AST Format vorliegen. Damit dieser nicht manuell erzeugt werden muss, können Meta-Klassen basierend auf Vorlagen erzeugt werden.

Für beide Wege sind entsprechende Konstruktoren implementiert worden, die ein Objekt erzeugen und konfigurieren. Die Meta-Klassen werden folgendermaßen eingeteilt:

- die Basisklassen (engl. low-level classes) repräsentieren VHDL Konstrukte wie Signal-Definitionen, Zuweisungen, und Prozesse.
- die Komplexklassen (engl. high-level classes) bauen auf die Basisklassen auf und repräsentieren komplette Einheiten wie Register und Multiplexerprozesse. Neben diesen allgemeinen Einheiten können auch sehr spezielle Einheiten wie Busschnittstellen und sonstige Protokolle implementiert werden. In den nachfolgenden Beispielen wird gezeigt, wie die Busanbindung in HiLDE auf Basis von Komplexklassen automatisch erzeugt wird.

Die Vorlagen, aus denen sowohl Basis- als auch Komplexklassen konstruiert werden können, bestehen aus VHDL-Quelltext Fragmenten, mit entsprechenden Platzhaltern, die den Metaklassen bekannt sind. Zur Erzeugung eines AST aus diesen Vorlagen wird die Vorlage geparst und die entsprechenden Platzhalter ersetzt. Eine weitere wichtige Funktion der Meta-Klassen ist die Fähigkeit, Kopien von Objekten zu erzeugen. Es gibt zwei Arten von Kopien im AST:

- Kopien gleichen Typs: Hierbei handelt es sich zumeist um Signale, deren Eigenschaften (Größe und Typ) einem neuen Signal zugeordnet werden. So kann z.B. bei der Erzeugung eines Registers das Ausgangssignal durch Kopieren des Eingangssignals erzeugt werden.
- Kopien unterschiedlichen Typs: Zwischen einigen Konstrukten in VHDL bestehen sehr enge Beziehungen, teilweise können sie sogar äquivalent umgeformt werden. So z.B. bei der Schnittstellenbeschreibung eines Entwurfes (engl. entity) die äquivalent mit der Definition ihrer Instanz (engl. component declaration) ist. Diese Umwandlungen kommen häufig vor, daher wurden die gängigsten Umwandlungen in den Meta-Klassen implementiert.

Manipulation bestehender Konstrukte

Neben der eigentlichen Erzeugung der Objekte stehen pro Klasse auch spezifische Funktionen zur Manipulation der Objekte zur Verfügung. So gibt es in allen Klassen eine Funktion *rename()*, die es erlaubt, beispielsweise einem Signal einen neuen Namen zu geben. Zu beachten ist, dass diese Operationen die semantische Korrektheit des AST beeinflussen kann. Während dies bei einer Namensänderung

automatisch aufzulösen ist (es muss sichergestellt sein, dass kein anderes Konstrukt mit dem selben Namen im Quelltext auftaucht, ggf. wird der neue Name erweitert, und es muss sichergestellt werden, das alle Instanzen des Signals ebenfalls umbenannt werden, sog. refactoring), kann die semantische Korrektheit z.B. bei der Änderung von Signalgrößen nicht automatisch sichergestellt werden:

```
signal a : std_logic_vector(15 downto 0);
signal b : std_logic;
[...]
b <= a(15);
[...]
```

In dem vorgestellten Beispiel wird deutlich, dass eine Umbenennung problemlos möglich ist, wenn der neue Name nicht `b` lautet, eine Veränderung der Größe des Signals nach `std_logic_vector(7 downto 0)` ist aber semantisch ambivalent, da nicht klar ist, wie die Zuweisung `b <= a(15)` jetzt lauten soll. Diese und ähnliche Ambivalenzen können nicht automatisch, sondern nur durch das Aufstellen zusätzlicher semantischer Regeln für den AST erkannt werden. Diese Funktionalität ist nicht in vMAGIC integriert, sondern soll in späteren Erweiterungen zu vMAGIC erstellt werden.

4.5.2 vMAGIC - Entwurfsablauf

Im Folgenden wird der vMAGIC Entwurfsablauf am Beispiel des HiLDE-Wrappers beschrieben. Die Erzeugung dieses Wrappers beinhaltet die folgenden Aufgaben:

- Instanziieren einer Standard Entity auf Basis eines Templates
- Deklaration und Instanziierung des DUT im Template
- Anschließen des Synchronizers an den DUT-Takteingang
- Instanziierung von Registern und verbinden mit den Daten-Signalen des DUT
- Erstellen und Konfigurieren einer Lokalbusschnittstelle, die mit den zuvor erstellten Registern verbunden wird
- ggf. Erzeugung der Logik für Ereignis-basierte Datenübertragung

Im Falle von HiLDE startet der vMAGIC Entwurfsablauf mit einem Benutzerdesign und einem VHDL-Rumpf des Hardware-Wrappers. Im Rumpf sind alle statischen Teile des Wrappers enthalten, also die Entity-Definition, die Architecture und der Synchronizer (siehe Abschnitt 4.3), da diese in der Regel nicht mehr verändert werden müssen. In Bild 4.14 wird der vMAGIC-Entwurfsablauf

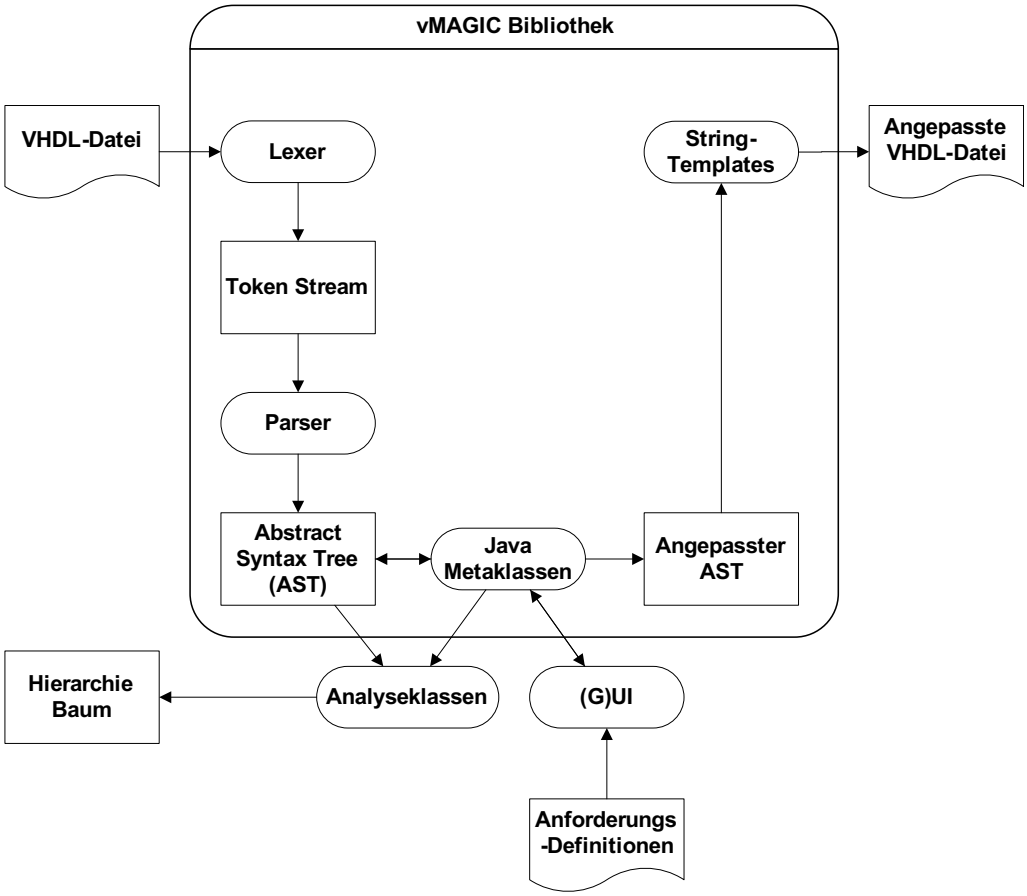


Abbildung 4.14: Der vMAGIC Entwurfsablauf

gezeigt. Beide VHDL Quelltexte werden vom vMAGIC Lexer und Parser in jeweils einen eigenen AST transformiert. Im nächsten Schritt wird mit Hilfe der Metaklassen der Benutzerentwurf in der HiLDE Vorlage unter Verwendung der Kopierfunktionen im AST der Vorlage instanziiert und verbunden. Dabei werden alle Ein- und Ausgänge des Benutzerentwurfs an neue Register angeschlossen, die ihrerseits wiederum an die Busschnittstelle angeschlossen werden. Zusätzlich werden die Taktsignale des Benutzerentwurfes an den Synchronizer angeschlossen, was den Vorgang abschließt. Der so erzeugte AST wird über die Stringtemplates in lesbaren VHDL Quelltext transformiert, der anschließend synthetisiert werden kann.

Zusätzlich zu den in diesem Beispiel gezeigten Funktionen können mit der vMAGIC API weitere Programme zur Unterstützung der VHDL Entwicklung entwickelt werden:

- das Routen von Signalen über Hierarchiegrenzen hinweg vereinfacht die Visualisierung interner Zustände

- die Transformation des AST, der eine Baumstruktur darstellt, in einen gerichteten, spärlichen Graphen ermöglicht weitere Graphen-orientierte Transformationen. Mit Graphen-APIs wie dem JUNG-framework [151] können beispielsweise spezielle Optimierungen, Partitionierungen usw. entwickelt werden.

- durch das Definieren weiterer Regeln lassen sich sog. Style-checker oder Linting-Werkzeuge entwickeln, die sicherstellen, dass bestimmte Stilvorgaben von den Benutzern eingehalten werden.

4.5.3 SiLLis

Um die oben beschriebenen Transaktoren für HiLDE und HiLDEGART effizient beschreiben zu können, wurde SiLLis (engl. Simple Language for Listeners) [4, 3, 2] entwickelt. In der Grammatik von SiLLis können sog. Pfade definiert werden, die Bedingungen mit einer zeitlichen Reihenfolge darstellen. So kann etwa bei einer Buskommunikation das Protokoll beschrieben werden, um den genauen Zeitpunkt, an dem Daten auf dem Bus anliegen, festzustellen. SiLLis wandelt diese Pfade in hierarchische Zustandsautomaten um, die dann zur Ansteuerung externer Module wie Speicherschnittstellen verwendet werden können. Um eine weitere Reduktion der Datenmenge zu erreichen, können auch Filter beschrieben werden, die einen zu beobachtenden Adressbereich oder Ähnliches beschreiben.

SiLLis stellt einen Compiler zur Verfügung, der die Transaktoren in eine abstrakte Hardwarebeschreibung übersetzt. Derzeit ist SiLLis [4] kompatibel mit vMAGIC und erzeugt auf diese Weise synthetisierbaren VHDL-Quelltext, der direkt in die Zielhardware integriert werden kann. SiLLis kann in Verbindung mit dem ebenfalls auf vMAGIC basierenden VHDL Editor EVE - Extendable VHDL Editor [15] verwandt werden.

4.6 Zusammenfassung

In diesem Kapitel wurde ein Entwurfsablauf für FPGA-basiertes Testen vorgestellt. Dabei wurden die Möglichkeiten und Grenzen dieser Technologie sowohl für zyklenakkurate als auch für Echtzeit-basierte Verfahren vorgestellt. In diesem Zusammenhang wurden zwei konkrete Testumgebungen (HiLDE und HiLDEGART) beschrieben, die im Rahmen von Tests eingebetteter Systeme eingesetzt werden können. Des Weiteren wurden Werkzeuge vorgestellt, mit denen die benötigten Hardwaresysteme teilweise automatisch erzeugt werden können. Die hier vorgestellten Testumgebungen haben das Potential sowohl einzelne Tests zu beschleunigen, als auch in einem frühen Stadium eines Hardwareentwurfes Fehler beim Übergang zur Zieltechnologie zu erkennen. Es können externe wie interne Signale beobachtet und manipuliert werden, wobei der Entwickler die gewohnte Umge-

bung (z.B. Matlab/Simulink oder ModelSim) nicht verlassen muss. Die Systeme sind plattformunabhängig und auch die automatisierte Erzeugung von Hardwarekomponenten kann ohne weiteres auf neue Bussysteme abgebildet werden.

Im Rahmen dieser Arbeit wurden die vorgestellten Werkzeuge für den Test der SOM-Prozessorfelder auf dem Rapid Prototyping System RAPTOR eingesetzt. Die hierbei gewonnenen Erkenntnisse fließen in die Implementierung eines prototypischen Steuerwerks für einen SOM-Hardwarebeschleuniger im nächsten Kapitel ein.

Kapitel 5

SOM-Implementierung

Die Schnittstellen der in Kapitel 3 entwickelten SOM-Hardware sind für den Betrieb als eingebettetes System, etwa an Bord eines autonomen Roboters oder eines Satelliten ausgelegt. Dabei übernehmen die Systemprozessoren des Trägersystems in einer engen Kopplung die Übertragung von Datenvektoren und Steuerbefehlen an die SOM-Hardware. Diese Schnittstelle kann auch im Rahmen eines SOM-Beschleunigers auf Basis des RAPTOR Prototyping Systems verwandt werden, aufgrund der vergleichsweise hohen Latenzen zwischen CPU und Beschleuniger[1] ist aber eine lokale Steuerung an Bord des RAPTOR Systems sinnvoll. Diese kann insbesondere die Lerndaten in einem eigenen Speicher vorhalten und gleichzeitig an mehrere Beschleuniger-FPGAs senden, die dann gemeinsam an der Berechnung der SOM arbeiten. Im Folgenden wird die Implementierung einer entsprechenden Steuereinheit für RAPTOR, und deren Softwareanbindung beschrieben. Anschließend werden Ressourcenbedarf und Leistung der Prozessorfelder als FPGA-basierte Implementierung vermessen und mit den Implementierungen aus Tabelle 1.1 verglichen. Dabei wird zwischen der Leistung als eingebettetem System und als Hardwarebeschleuniger im PC unterschieden.

5.1 Steuereinheit für RAPTOR

Das RAPTOR System eignet sich mit seinen bis zu sechs FPGA-Modulen für die prototypische Implementierung eines massiv parallelen SOM-Beschleunigers, der die Lernleistung von gängigen CPUs unter Umständen übersteigen kann, wobei gleichzeitig der Energieverbrauch zurückgeht. Im Folgenden wird die Steuereinheit beschrieben, die in Verbindung mit den SOM-Prozessorfeldern ein massiv

[1] Im Ausblick wird eine weitere Möglichkeit der Implementierung vorgestellt, bei der diese Beschränkungen in dieser Form nicht mehr bestehen

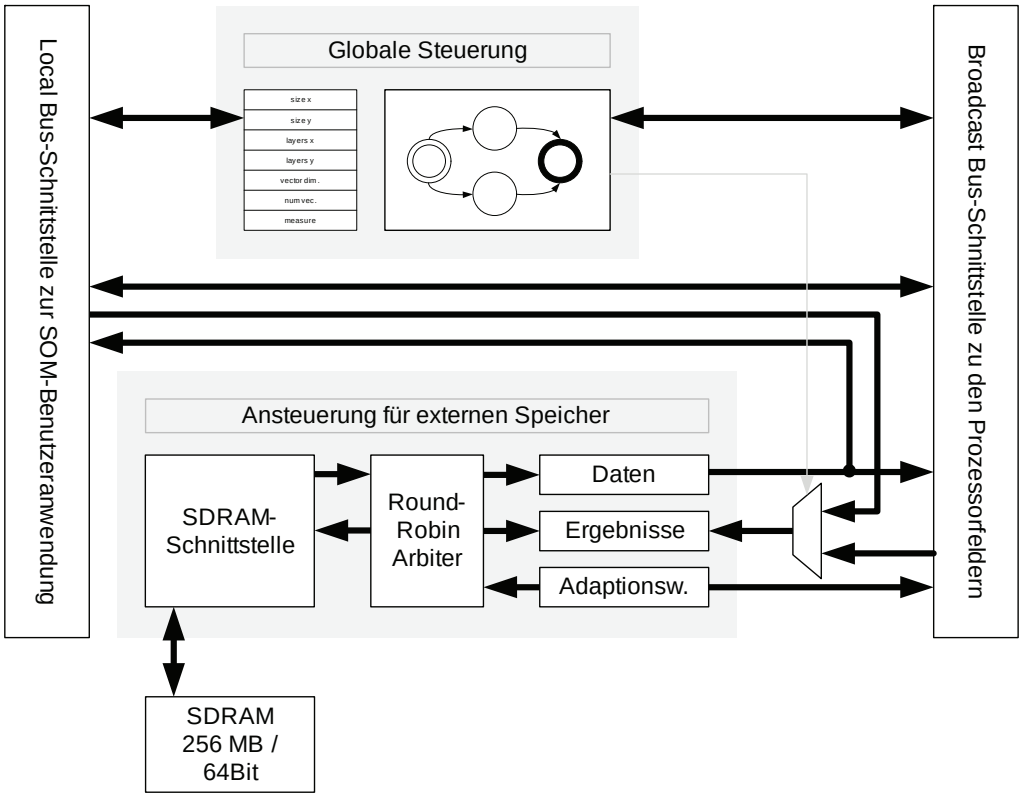


Abbildung 5.1: Steuereinheit für SOM auf RAPTOR2000

paralleles Beschleunigersystem ergibt. In Abbildung 5.1 werden die wesentlichen Komponenten des SOM-Controllers für RAPTOR gezeigt. Die zentralen Elemente sind die Steuereinheit einerseits und die Speicheransteuerung andererseits. Die Steuereinheit besteht aus einigen Konfigurationsregistern und einer FSM, die sowohl die Steuerbefehle für die Prozessorfelder als auch die Kontrollsignale für die Speicheransteuerung generiert. Die Speicheransteuerung dient als Schnittstelle zum externen Speicher, der auf dem FPGA Modul vorhanden ist, und vereinigt drei round-robin arbitrierte FIFO Speicher und die SDRAM-Ansteuerung. Es wurden ein Schreib-FIFO für die Ergebnisse und zwei Lese-FIFOs für die Daten und die Alpha-Programme implementiert. Gleichzeitig werden der Daten- und der Ergebnis FIFO-auch als Schnittstelle für den Zugriff auf das SDRAM von außen genutzt. Über die 32 Bit breite Broadcast-Busschnittstelle kommuniziert die Steuerung mit den Prozessorfeldern, um Daten und Statusinformationen auszutauschen. Die Lokalbusschnittstelle dient der Anbindung des Beschleunigers an den PCI Bus des PCs und ermöglicht so die Kommunikation zwischen dem Benutzer und der Hardware. Diese wurde mit Hilfe eines vMAGIC-basierten Hardware-Generators erstellt.

Die Steuereinheit stellt also im Wesentlichen einen Mittler zwischen Prozes-

sorfeld und Benutzer dar, der lokal Daten vorhält, Ergebnisse speichert und dabei die Steuerung mehrerer Prozessorfelder übernehmen kann. Da die Datentransfers aufgrund der implementierten Schnittstelle nicht ausnahmslos parallel zur Berechnung ausgeführt werden können, kommt es durch den Controller zu Verzögerungen bei den Berechnungen und somit zu einer Schmälerung der Nettoleistung. Die Wesentlichen Faktoren sind hier:

- **Datentransfers zum/vom PC:** Alle Vektoren, Adaptions- und Konfigurationswerte, sowie (nach dem Lernvorgang) alle Ergebnisse müssen über den PCI-Bus übertragen werden. Die dazu benötigte Zeit ist für die Gesamtleistung nicht vernachlässigbar und kann anhand von entsprechenden Busmodellen wie in [72] abgeschätzt werden. Die Auswirkung dieser Latenzen wird in Abschnitt 5.3 vermessen.

- **Datentransfers zu den Prozessorfeldern:** Sowohl die Vektoren als auch die Adaptionswerte werden während des Lernvorgangs in die lokalen Cache-Speicher der Prozessorfelder übertragen. Dabei kann es abhängig von der Kommunkationsstruktur zu Verzögerungen kommen.

- **Ermittlung des globalen Erregungszentrums:** Beim Einsatz mehrerer Prozessorfelder müssen jeweils im Anschluss an eine Abstandsberechnung Distanz und Position der lokalen Erregungszentren an die Steuereinheit übertragen werden. Diese kann dann das globale Erregungszentrum ermitteln und dessen Position (für die Adaption) an die Prozessorfelder übermitteln.

In Abschnitt 5.3 werden die Auswirkungen dieser Faktoren auf die Gesamtleistung betrachtet.

5.2 Softwareumgebung

Zur Kommunikation mit dem RAPTOR2000 System wurde die Software-Bibliothek R2Klib [72] erstellt, die neben der reinen Datenkommunikation auch Funktionen für das Konfigurieren der FPGAs und den Austausch von Statusinformationen enthält. Diese wurde verwendet, um einer Benutzerschnittstelle die SOM-Funktionalität in Form einer C++ Klasse zur Verfügung zu stellen, die alle Konfigurationsmöglichkeiten und die Funktionalität der Hardware bündelt. Im Folgenden werden die wichtigsten Funktionen der Klasse erläutert:

- Die Anzahl der Kartenelemente in x- und y-Richtung sind frei wählbar, solange der lokale Speicher ausreicht (siehe Abschnitt 3.2.3).

- Die Abstandsmaße für die Abstandsberechnung und Adaption (euklidisch oder Manhattan), sind separat einstellbar.

	l	d_m (max)	w_m [Bit]	F [MHz]	Ress. [Slices]	$P_{C,l}$ [MCUPS]	$\frac{P_{C,l}}{F}$	$\frac{P_{C,l}}{Ress}$
[11]	64	1024	16	50	24384	1184	23,68	0,0486
NP-B	64	2048	16	62	25371	1967	31,72	0,0775
[128]	64	4	8	33	k.A.	40	1,21	k.A.
NP-B	64	2048	16	62	25371	634	10,22	0,0250
[102]	25	2	8	27	3009	28	1,03	0,0093
NP-B	25	4096	16	91	25371	216	2,37	0,0085
[82]	128	3	8	24	37312	1651	66,04	0,0442
NP-B	128	1024	16	52	48460	831	16,00	0,0171
[109]	256	16	8	71	21961	124	1,74	0,0056
NP-B *	128	1024	16	52	48460	2597	49,95	0,0536

Tabelle 5.1: Bestehende FPGA Implementierungen aus Tabelle 1.1 im Vergleich mit den hier entstandenen Implementierungen. Die Ressourcen beziehen sich auf eine Implementierung mit Unterstützung für Conscious SOM auf Basis der Virtex2 Familie von Xilinx; Speicherblöcke und Multiplizierer wurden nicht separat angegeben, weil sie linear mit l gehen.

- Die Adaptionsfunktion(en) können in Form von double-Feldern im Wertebereich [0..1] übergeben werden. Für jede Funktion kann die Gültigkeitsdauer separat gewählt werden.

- Nach dem Lernvorgang können (a) die Referenzvektoren der Karte, (b) die Positionen der Erregungszentren und (c) die Abstände zu den Erregungszentren abgerufen werden.

Um das Hardware-Beschleunigersystem einem möglichst großen Benutzerkreis zur Verfügung zu stellen, wurde eine zusätzliche Schnittstelle für Matlab entwickelt, die alle Funktionen der Hardware für Matlab Skripte nutzbar macht. Diese Schnittstelle verwendet Techniken, wie sie auch für das HiLDE System (Kapitel 4.3.7) zum Einsatz kommen. Ebenso wurde ein Adapter für die SOM-Toolbox [141] entwickelt, so dass viele zusätzliche Werkzeuge aus dem Bereich SOM, wie z.B. Visualisierung, Pre- und Postprocessing usw. mit der Hardware verwendbar sind.

5.3 Ressourcenbedarf und Leistungsfähigkeit

In Tabelle 5.1 werden analog zu Tabelle 3.5 die Ressourcen und Leistungsdaten von FPGA-basierten Implementierungen aus der Literatur mit entsprechenden Implementierungen aus dieser Arbeit verglichen. Dabei wurde jeweils die gleiche Anzahl an Prozessorelementen verwandt. Die mit '*' gekennzeichnete Zeile stellt

eine Ausnahme dar, da kein Bauteil der Virtex 2 Familie von Xilinx 256 Prozessorelemente fassen kann. Stattdessen wurde ein Prozessorfeld mit einer geringeren Anzahl an Prozessorelementen verwandt, bei dem dann über Virtualisierung mehr Speicherressourcen genutzt wurden.

Bei den FPGA-basierten Implementierungen zeigt sich der Ressourcenunterschied zwischen den hier entwickelten und den aus der Literatur bekannten Implementierungen wesentlich deutlicher, als bei den ASIC-basierten Systemen (siehe Abschnitt 3.5). Dies begründet sich, abgesehen von den strukturellen Unterschieden, vor allem durch die Vorgehensweise der Abbildungswerkzeuge, die die vorhandenen Logikressourcen zu Gunsten einer höheren Taktfrequenz nicht optimal ausnutzen. Insgesamt zeigt sich, dass die hier entwickelten Implementierungen häufig mehr als doppelt so viele Logikressourcen benötigen, was durch die doppelte Wortbreite verursacht wird. Gleichzeitig sind die hier vorgestellten Implementierungen aber auch bis auf eine Ausnahme [82] deutlich schneller als die Referenzen. Um diese Leistungsunterschiede auf rein architektureller Ebene betrachten zu können, wurde die Leistung zusätzlich auch normiert auf die Ressourcen (Anzahl der Slices) und die Frequenz angegeben; hierbei ist anzumerken, dass die so gewonnenen Werte nur zwischen zwei Implementierungen mit der gleichen Anzahl an Prozessorelementen und mit der gleichen Anzahl an verarbeiteten Komponenten möglich ist. Alle anderen Vergleiche sind nicht zulässig und eine weitere Normierung ist nicht möglich, da der Zusammenhang zwischen Leistung und Anzahl der Komponenten nur für die hier entwickelten Architekturen bekannt ist. Bei diesem Vergleich zeigt sich also, dass die hier entwickelten Architekturen im Vergleich (bis auf [82]) ein deutlich besseres Verhältnis von Leistung und Latenz haben, während das Verhältnis von Leistung zu Ressourcen unter Berücksichtigung der größeren Wortbreite sehr gut ist. Anhand der Implementierung von Kurdthongmee [82] lässt sich wie auch bei den ASIC-basierten Implementierungen zeigen, inwieweit sich die Flexibilität und die höhere Präzision der hier entworfenen Hardwarebeschleuniger auf die Leistung auswirkt. Kurdthongmees Implementierung wird mit der halben Taktfrequenz (24 MHz) betrieben und erzielt trotzdem eine in dem betrachteten Bereich zweifach höhere Lernleistung. Dies resultiert in einen Faktor vier beim Vergleich der Lernleistung pro Takt und einer etwa dreifach höheren Lernleistung pro Slice. Auch ist der Kontrollaufwand bei Kurdthongmee sehr gering, weil das System auf genau eine Anwendung, eine SOM mit 256 Neuronen und drei Komponenten spezialisiert ist. An dieser Stelle muss erneut darauf hingewiesen werden, dass es bei dieser Implementierung um eine sehr spezialisierte Lösung für genau eine Kartengröße mit genau einer Referenzvektorgröße handelt. In dieser Form lässt sich die Hardware nicht für andere als den angegebenen Fall verwenden. Insbesondere basieren die Referenzspeicher hier auf Registern, während in der hier entwickelten Hardware RAM Speicherblöcke mit entsprechenden Addresslogiken etc. verwandt werden. Diese, und viele andere Bausteine, die die

	l	d_m (max)	w_m [Bit]	F [MHz]	Ress. [Slices]	$P_{C,l}$ [MCUPS]	$\frac{P_{C,l}}{F}$	$\frac{P_{C,l}}{Ress}$
[11]	64	1024	16	50	24384	1184	23,68	0,0486
NP-B	32	2048	16	74	13327	1184	15,86	0,1012
[128]	64	4	8	33	k.A.	40	1,21	k.A.
NP-B	4	4096	16	70	3314	40	0,40	0,0130
[102]	25	2	8	27	3009	28	1,03	0,0093
NP-B	4	4096	16	70	3314	28	0,40	0,0130
[82]	128	3	8	25	25944	1651	66,04	0,0636
NP-B	nicht erreichbar							
[109]	256	16	8	71	21961	124	1,74	0,0056
NP-B *	4	4096	16	93	3314	124	1,33	0,0435

Tabelle 5.2: Bestehende FPGA Implementierungen aus Tabelle 1.1 im Vergleich mit den hier entstandenen Implementierungen. Dabei wurden möglichst kleine Implementierungen gewählt, die die vorgegebene Leistung erfüllen können, und die Frequenz entsprechend herunter skaliert.

Flexibilität der in dieser Arbeit vorgestellten Lösung ausmachen, erklären den Geschwindigkeitsvorteil von Kurdthongmees Implementierung. Gleichzeitig wird deutlich, dass der Geschwindigkeitsvorteil Kurdthongmees nur bei sehr kleinen Karten zur Geltung kommen kann.

In Tabelle 5.2 werden ebenfalls die Referenzimplementierungen mit den hier entwickelten gegenübergestellt, wobei hier die Leistung den Vorgaben angepasst wurde. Es wurden möglichst kleine Implementierungen gewählt, und die Taktfrequenz herunter skaliert, so dass die Lernleistung exakt den Vorgaben entspricht. Dabei zeigt sich, dass sich die vorgegebene Lernleistung auch bei einer höheren Wortbreite oft mit wesentlich weniger Ressourcen erreichen lässt, als in der Literatur beschrieben.

5.3.1 Messung der Lernleistung

Die zuvor beschriebenen Systeme (sowohl Referenzimplementierungen als auch hier entwickelte Systeme) gehen davon aus, dass sie direkt in eine Anwendung integriert sind, d.h. Verluste durch eine übergeordnete Steuerung, wie sie in Abschnitt 5.1 beschrieben wurde, werden nicht berücksichtigt. Für ein Beschleunigersystem, das in eine PC-Umgebung eingebettet ist, kann diese Steuerung nicht vernachlässigt werden; zahlreiche Datentransfers zwischen Steuerung und PC bzw. zwischen Steuerung und Prozessorfeld müssen durchgeführt werden und können z.T. große Leistungseinbußen verursachen. Um diese Verluste genauer zu untersuchen, wurde ein prototypischer SOM-Prozessor mit Hilfe des RAPTOR Systems aufgebaut, an dem diese Faktoren genau vermessen werden können. Dazu wurde

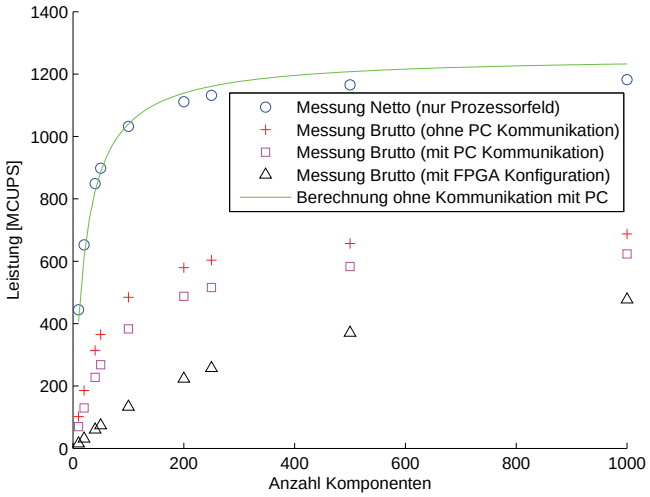


Abbildung 5.2: Leistungsmessung mit RAPTOR

das Modell mit verschiedenen Messungen gegenübergestellt:

- **Nettoleistung:** Die Leistung, die der SOM-Prozessor erbringt, wenn keinerlei Verzögerungen durch eine zentrale Steuerung auftritt.
- **Bruttoleistung ohne PC Kommunikation:** Hierbei werden die Effekte der Datentransfers zwischen Prozessorfeld und der FPGA-basierten Steuerung einbezogen.
- **Bruttoleistung mit PC Kommunikation:** Zusätzlich werden die Verzögerungen durch die Datentransfers zum/vom PC einbezogen (Vektoren, Adaptionswerte, Konfiguration, Gewichtsvektoren, Distanzen und Positionen der Erregungszentren, sowie Vor- und Nachverarbeitungsschritte)
- **Bruttoleistung mit FPGA Konfiguration:** Im letzen Schritt wird auch die Konfiguration der FPGAs (herunterladen der Bitströme) in die Messung mit einbezogen. Dieser Schritt muss prinzipiell nur einmal nach dem aktivieren der Stromversorgung erfolgen, und kann für alle weiteren Lernvorgänge entfallen.

Abbildung 5.2 zeigt die Ergebnisse dieser Messung für zwei Prozessorfelder á 36 Prozessorelementen, jeweils auf einem XC2VP30 FPGA. Es wurde eine neuronenparallele Implementierung mit binärer Suche implementiert, und ein Datensatz mit 1000 Vektoren zufällig generiert. Dieses Beispiel stellt in jeder Hinsicht den schlechtesten Fall für das System dar:

1. **Kurze Transfers:** Die Zeitdauer für die Initialisierung eines Transfers ist besonders bei kurzen Transfers in der gleichen Größenordnung wie der ei-

gentlich Transfer, so dass hier effektiv sehr geringe Bandbreiten erzielt werden.

2. **Kurze Lerndauer:** Die konstanten Zeiten für das Initialisieren des Systems (konfigurieren der FPGAs und der Steuerung) sind hier sehr groß im Verhältnis zu der Gesamtlerndauer. Dadurch wird, falls das System nach jedem Lernvorgang neu initialisiert werden muss (was nicht zwingend notwendig ist), eine geringe Gesamtleistung erzielt.

Diese Effekte zeigen sich in Abbildung 5.2 durch die Abstufungen in der Lernleistung. Während der theoretisch berechnete Wert (Berechnung ohne Kommunikation mit PC) sehr gut mit den gemessenen Werten (Messung Netto) übereinstimmen, weichen die Werte für die Gesamtleistung bis um den Faktor $2,5$ ab. Im Einzelnen erklären sich diese Abweichungen wie folgt:

Die Ungenauigkeiten zwischen Modell und Messung werden im Wesentlichen durch die Abschätzung der durchschnittlichen Adaptionsfunktion und durch die im Modell nicht berücksichtigten algorithmischen Optimierungen (siehe Abschnitt 3.4.6 verursacht. Dadurch kommt es im Bereich kleiner etwa 100 Komponenten zu einer besseren gemessenen als berechneten Leistung. Im Bereich größer 100 Komponenten liegen die gemessenen Werte leicht unter den theoretisch ermittelten, da die durchschnittliche Anzahl der adaptierten Komponenten, und der daraus folgende Radius der Adaptionsfunktionen zu klein angenommen wurden. Dies begründet sich durch die willkürliche Wahl der Adaptionsfunktion in diesem Beispiel, die aufgrund zuvor erzielter Ergebnisse größer als der berechnete Durschnitt gewählt wurde. Solche Anpassungen (siehe Tabelle 3.4) lassen sich im Modell berücksichtigen. Eine erneute Auswertung des Modells mit der korrigierten Annahme für die Adaptionsfunktion zeigt in diesem Beispiel, dass durch die Algorithmischen Optimierungen eine Leistungssteigerung von bis zu 10% erreicht werden kann. Diese Ergebnisse können gleichzeitig als Leistungsmaß für eine eingebettete SOM-Hardware benutzt werden.

Die deutliche Abweichung zwischen den Messungen mit und ohne globale Steuerung weist auf eine sub-optimale Implementierung sowohl der Steuerung an sich, als auch der Kommunikation zwischen Steuerung und Prozessorfeld hin. Tatsächlich wurden in dieser prototypischen Implementierung keine sog. Burst-Transfers implementiert. Das bedeutet, dass die Übertragung jedes Datenwortes neu initialisiert werden muss und keine Bündeleffekte auftreten können. Zusätzlich werden nicht alle Möglichkeiten der parallelen Datenübertragung (Daten und Adaptionswerte) ausgenutzt. Dieser Leistungsverlust von ca. 42% kann durch eine bessere Implementierung nahezu vollständig ausgeglichen werden, und muss bei einer praktischen Implementierung unbedingt berücksichtigt werden. Eine exakte Betrachtung liefert folgende Gleichung für den minimalen zusätzlichen Aufwand bei der Kombination mehrerer FPGAs (oder ASICs) über eine gemeinsame Steue-

rung:

$$T_{sync} = n_{Pf} \left\lceil \frac{w_d + 2w_c}{w_b} \right\rceil + \left\lceil \frac{2w_c}{w_b} \right\rceil \tag{5.1}$$

Die Anzahl der Bustransaktionen für die Suche und Kommunikation eines globalen Erregungszentrums T_{sync} berechnet sich aus der Anzahl der Transaktionen für Distanz und Position der individuellen Erregungszentren pro Prozessorfeld n_{Pf} und der Kommunikation der Position des globalen Erregungszentrum an alle Prozessorfelder. Unter Berücksichtigung der gewählten Busarchitektur ergibt sich für die hier vermessene Implementierung eine zusätzliche Latenz von 28 Takten für jeden Lernschritt.

Die Datenübertragungen zwischen PC und Steuerung erzeugen einen weiteren Leistungsverlust von ca. 9%, der nur durch die Implementierung schnellerer Übertragungsverfahren, etwa den Einsatz von PCI-XPress verringert werden kann. Zusätzlich werden in dieser prototypischen Implementierung der Steuerung auch alle Erregungszentren (Position und Distanz zum Eingabevektor) über den gesamten Lernvorgang gesammelt und an den PC übertragen. Für eine praktische Implementierung ist dieser Schritt nicht unbedingt notwendig und kann daher eingespart werden. Alle Vor- und Nachverarbeitungsschritte (Konvertierung von Gleitkomma nach Festkomma, Anordnung der Vektoren im Speicher etc.) wurden ebenfalls in diese Messung integriert.

Die Konfiguration der FPGAs, in diesem Fall zwei Virtex2Pro 30 Bausteine und ein Virtex2 4000 als Steuerung, wirkt sich mit weiteren bis zu 25% Leistungsverlust aus. Wie eingangs erwähnt ist dieser Schritt aber in der Regel nur einmal beim Systemstart notwendig, und kann für alle weiteren Lernvorgänge vernachlässigt werden. Dementsprechend wurde dieser Teil bei der Messung der Leistungsaufnahme in Abbildung 5.4 nicht berücksichtigt.

Als Fazit dieser Messungen ergeben sich im Wesentlichen zwei Punkte: Die Einbettung der Prozessorfelder in eine PC Umgebung, also ein FPGA-basiertes SOM-Beschleunigersystem kostet aufgrund der Verzögerungen durch Datenübertragungen zwischen PC und FPGA etwa 9% der Lernleistung. Verbesserungen dieses Wertes können durch optimierte Übertragungsverfahren erreicht werden. Als zweiter Punkt ist die Implementierung der globalen Steuerung zu nennen, die in diesem Beispiel ca. 42% der Lernleistung kostet. Diese muss angesichts der Tatsache, dass eine optimierte Steuerung ausschließlich zur Ermittlung eines globalen Erregungszentrums sequentiell arbeiten muss, überarbeitet werden.

5.3.2 Messung der Leistungsaufnahme

Die Leistungsaufnahme wurde für die drei Versorgungsspannungen mit Hilfe eines Hall-Sensor-basierten Messverfahrens (siehe Abbildung 5.3) aufgenommen. Die Ergebnisse für 5 V (Kernspannung, wird auf 1,5 V gewandelt, siehe Abbildung 5.4 (a)) 3,3 V (Ein- und Ausgangsspannung, siehe Abbildung 5.4 (b)) und 2,5

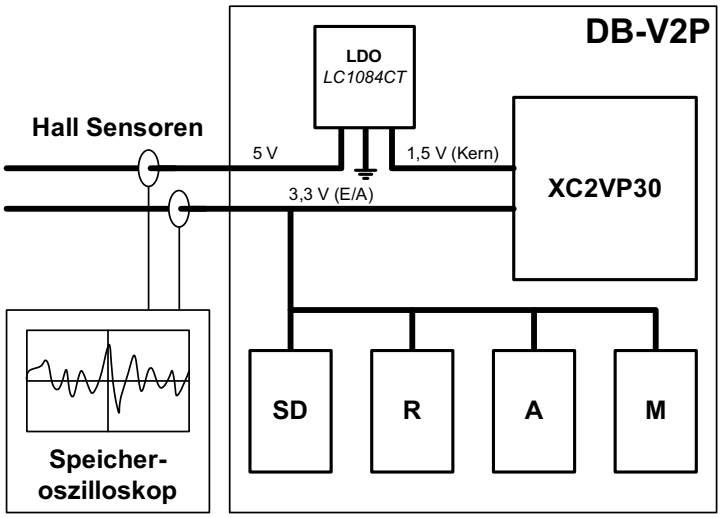
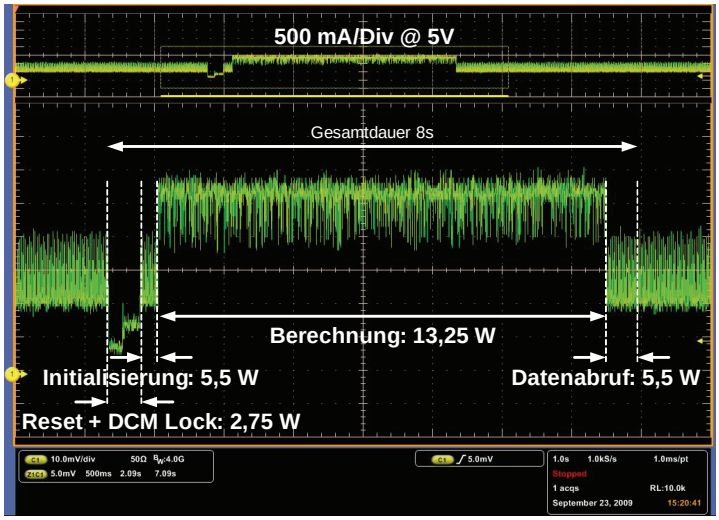


Abbildung 5.3: Messaufbau Leistungsaufnahme DB-V2Pro

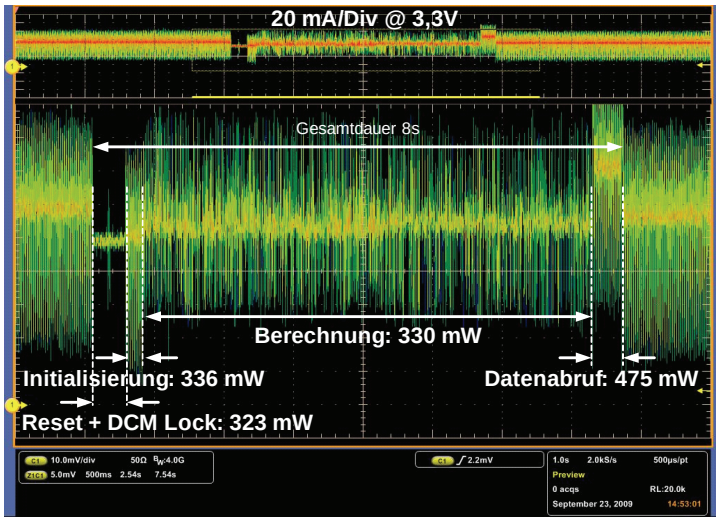
V (Hilfsspannung für Taktnetze, DCMs, Multiplizierer etc., hier nicht gezeigt da konstant bei 50 mW) zeigen, dass sich die Leistungsaufnahme in den einzelnen Arbeitsbereichen unterscheidet:

- **Vorbereitung:** In dieser Phase werden alle Register zurückgesetzt, einige Kommunikationstests durchgeführt und gewartet, bis die Takterzeuger für die SDRAM-Steuerung (DCM, Digital Clock Manager) einen eingeschwungenen Zustand erreichen.
- **Initialisierung:** Die Speicherblöcke werden mit zufälligen Werten beschrieben, für die Positionen der Neuronen im Gitter müssen einige Werte übertragen werden. Insgesamt ist nur ein sehr kleiner Teil der Gesamtschaltung aktiv.
- **Berechnung:** Hier werden Abstandsberechnung, Suche des Erregungszentrums und Adaption durchgeführt, dabei müssen die Datenvektoren, Adaptionswerte, lokale und globale Erregungszentren übertragen werden.
- **Datenabruf:** Beim Abruf der Referenzvektoren ist wiederum nur ein kleiner Teil der Schaltung aktiv, es werden aber viele Werte über den Datenbus übertragen.

In Tabelle 5.3 sind die Messergebnisse zusammengefasst, wobei für die 5 V Versorgungsspannung der Linearregler *LC1084CT* berücksichtigt wurde. Dabei zeigt sich, dass der bei weitem größte Anteil an der Leistungsaufnahme auf der Kernspannungsschiene geschieht, also durch die eigentlichen Logikfunktionen des FPGA. Eine Größenordnung darunter liegen die Werte für die Ein- und Ausgangsspannung und wieder eine Größenordnung darunter die Hilfsspannung. Die



(a) Leistungsaufnahme 5 V Versorgungsspannung, Kernspannung (gewandelt auf 1,5 V)



(b) Leistungsaufnahme 3,3 V Versorgungsspannung (Ein- und Ausgangstreiber)

Abbildung 5.4: Messung der Leistungsaufnahme am FPGA Modul DB-V2Pro30

Aufteilung der Versorgungsspannungen in Kern-, E/A- und Hilfsversorgung ermöglicht zusätzlich eine Abschätzung für die Leistungsaufnahme pro Prozessorelement. Demnach liegt während der Ausführung der Kernelemente des Algorithmus bei ca. 110 mW pro Prozessorelement bei 50 MHz. Dieser Wert lässt sich als Basis für alle weiteren Virtex2 und Virtex2Pro-basierten Implementierungen heranziehen.

	1,5 V [W]	1,5 V [mW/PE]	3,3 V [mW]	2,5 V [mW]
Vorbereitung	0,83	23,1	323	50
Initialisierung	1,65	45,8	336	50
Berechnung	3,98	110,6	330	50
Datenabruf	1,65	45,8	475	50

Tabelle 5.3: Leistungsaufnahme des FPGAs XC2VP30 auf den unterschiedlichen Versorgungsspannungen bei 36 PEs@50MHz

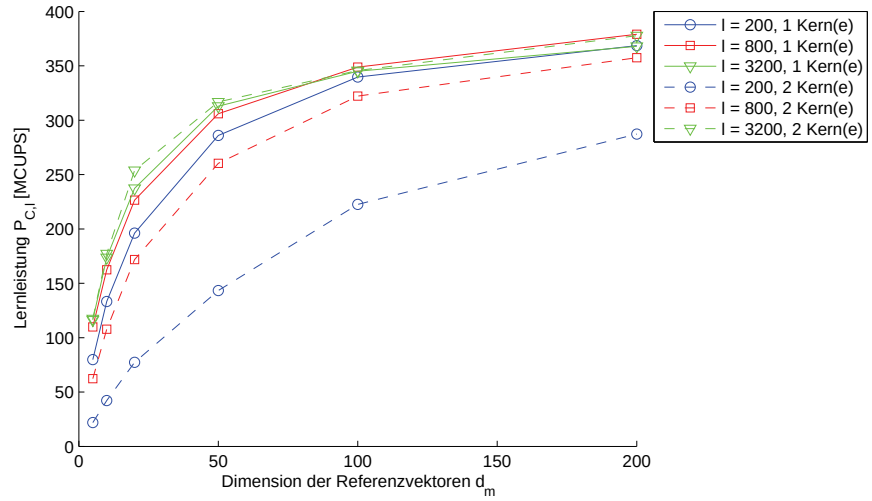
Prozessor	Architektur	TDP [W]	Fläche mm^2	PEs/ (HT)	F [MHz]	Techn. [nm]	Cache [KBytes] L1	L2	L3	Start
Pentium 4	Northwood	71	146	1/1	2405	130	8	512		01/2002
Pentium 4	Northwood	89	146	1/2	2993	130	8	512		01/2002
Core 2 Duo	Wolfdale	65	107	2/2	3000	45	2×32	6144		01/2008
Core 2 Duo M	Penryn	25	81	2/2	2530	45	2×32	3072		01/2009
Core i7	Bloomfield	130	263	4/8	2582	45	4×32	4×256	8192	11/2008
Atom	Diamondville	9	52	2/4	1596	45	2×32	2×512		09/2008

Tabelle 5.4: Technische Daten der Referenzplattformen

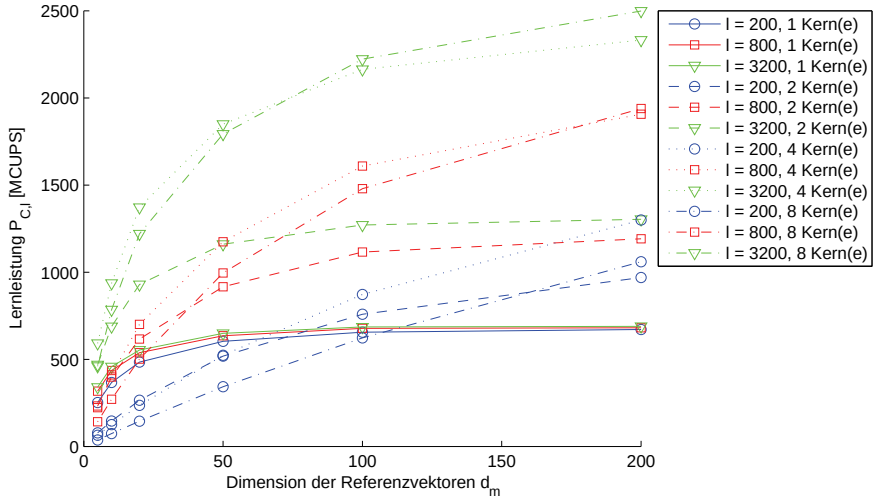
5.4 Referenzimplementierungen

Um die im vorangegangenen Abschnitt gemessenen Ergebnisse bewerten zu können, wird hier eine Software-basierte Lösung auf unterschiedlichen Plattformen vermessen. Als Referenzimplementierung dient ein in C++ entwickeltes Programm, welches (algorithmisch) genau wie die oben beschriebenen Hardwareimplementierungen mit 16 Bit Festkommazahlen arbeitet. Die Ausnutzung der SIMD Befehlssätze der Prozessoren (MMX, SSE, SSE2) wurde durch die Verwendung der entsprechenden Compileroptionen sichergestellt, die Parallelisierung auf mehrere Rechenkerne erfolgte durch die Nutzung von OpenMP [44]. Die vermessenen Referenzplattformen werden in Tabelle 5.4 vorgestellt.

Abbildung 5.5 zeigt die Lernleistung zweier Intel Desktopprozessoren aus den Jahren 2002 (Pentium 4 Northwood) und 2008 (Core i7 Bloomfield). Die Durchführung der Messungen erfolgte mit Hilfe künstlicher Datensätze, die Parameter der SOM wurden dabei in einem weiten Spektrum variiert. In der Abbildung sind alleine die Parameter d_m (Dimension der Referenzvektoren) und l (Anzahl der Neuronen im Netzwerk dargestellt, die sich mit Abstand als die wichtigsten Einflussfaktoren herausgestellt haben. Ein Vergleich der beiden Messungen zeigt deutlich, dass sich der moderne Prozessor abgesehen von einer deutlich gesteigerten Lernleistung durch die bessere Ausnutzung paralleler Einheiten auszeichnet. Während der Pentium 4 Prozessor, der mit Hilfe der sog. hyperthreading Technologie [90] über zwei logische Rechenkerne verfügt, bei der Aufteilung auf zwei Ausführungsstränge (engl. Threads) deutliche Leistungseinbußen aufweist, kann



(a) Lernleistung eines Pentium 4 (Northwood) Prozessors bei 2,99 GHz mit Hyperthreading



(b) Lernleistung eines Core i7 Prozessors bei 2,93 GHz mit vier Kernen und Hyperthreading

Abbildung 5.5: Unterschiedliche Prozessorgenerationen im Vergleich

bei dem modernen Prozessor eine Leistungssteigerung, jeweils bis nahe Faktor zwei pro Kern verzeichnet werden. Daraus lässt sich schließen, dass die Kommunikationsinfrastruktur der modernen Prozessoren sehr gut auf die parallele Verarbeitung rechenintensiver Anwendungen ausgerichtet ist. Da die Hyperthreading Technologie (HT) nicht auf unabhängigen Rechenkernen, sondern vielmehr auf einer schnellen Umschaltung zwischen zwei Threads beruht, die z.B. immer dann ausgeführt wird, wenn ein Thread wartet (beispielsweise auf Daten), liegt die Vermutung nahe, dass die Geschwindigkeitseinbußen beim Pentium 4 auf ein schlechteres Cache-Management zurückzuführen sind. Die Messungen an moder-

nen Prozessoren mit HT Technologie (Core 2 Duo, Core i7) zeigen jedoch, dass diese Engpässe inzwischen beseitigt wurden.

Genauso lässt sich aber auch der Flaschenhals bei modernen mehrkernigen Prozessoren erkennen. Erst wenn alle Kerne bzw. Threads eine ausreichend große Datenmenge in den Zwischenspeicher (Cache) laden können, wird auch deren Leistung vollständig ausgenutzt. Für den Core i7 liegt diese kritische Datenmenge, bei der die Lernleistung in eine Sättigung übergeht bei ca. 320 kBytes pro Kern. In Abbildung 5.5 zeigt sich dies durch den Eintritt in die Sättigung; ein Kern geht bei ca. 100 Komponenten in Sättigung, zwei Kerne bei ca. 200. Dieses Verhalten ist bei allen untersuchten Mehrkernprozessoren (außer P4 Northwood) mit leicht variierenden kritischen Datenmengen zu beobachten.

5.4.1 Vergleich

Die Messungen aus Abschnitt 5.3 zeigen, dass die theoretischen Werte für die Lernleistung der SOM-Hardware insbesondere in den relevanten Bereichen gut die tatsächlich erzielten Werte widerspiegeln. Damit können für ASIC- und FPGA-basierte SOM-Hardwarebeschleuniger sehr genaue Voraussagen über ihre Leistungsfähigkeit getroffen und mit der Leistungsfähigkeit von Software-basierten Verfahren verglichen werden. Da für einen realitätsnahen Vergleich der Implementierungen die eingebettete Architektur nicht in Frage kommt (Desktop CPUs werden aus offensichtlichen Gründen in eingebetteten Systemen nicht verwendet, insbesondere dann nicht, wenn die Systeme autonom und energieautark arbeiten; in [19] werden verschiedene Beschleunigerarchitekturen auf Basis von eingebetteten Multiprozessoren verglichen), muss ein SOM-Beschleunigersystem in einem PC als Grundlage herangezogen werden. In Abschnitt 5.3 wird gezeigt, dass in diesem Szenario auch mit einer optimierten Steuerung Leistungseinbußen (siehe Gleichung 5.1) gegenüber der eingebetteten Lösung in Kauf genommen werden müssen, diese werden in den folgenden Szenarien berücksichtigt.

Als Bezugsgröße für den Vergleich kommen im Wesentlichen zwei Faktoren in Frage. Einerseits können Implementierungen auf Basis der gleichen Chipfläche (engl. area) verglichen werden, andererseits auf Basis der gleichen Leistungsaufnahme. Da die bisher schon bekannten Ergebnisse zeigen, dass Leistungsaufnahme und insbesondere Energiebedarf zwischen CPU und FPGA/ASIC um Größenordnungen auseinanderliegen, macht die Normierung auf die Leistungsaufnahme hier keinen Sinn. Stattdessen wird der Energiebedarf im Fall mit gleicher Fläche diskutiert. Die dazu benötigten Werte werden aus verschiedenen Quellen zusammengetragen: Die Lernleistung wird mit Hilfe der Modelle und Messungen aus den vorangegangenen Abschnitten bestimmt. Die Chipfläche kann für die ASIC Implementierung berechnet, für die CPUs aus Datenblättern entnommen und für FPGAs aus Röntgenbildern und Chipfotos bestimmt werden. Bei der Leistungsaufnahme kommen für die ASIC Implementierungen nur die entwickelten Modelle,

	Fläche [mm^2]	Techn. [nm]	F [MHz]	PE	Speicher [kByte]
Pentium 4 Northwood	146	130	2993	1	520
Xilinx XC2V1500	≈150	120	91	21	98
ASIC (a)	146	130	<346	2000	98
ASIC (b)	146	130	<346	1800	520

Tabelle 5.5: Technische Daten der Referenzplattformen

bei CPUs Datenblätter und bei FPGAs Messungen in Frage. Die Unterschiedlichkeit dieser Quellen hat zur Folge, dass die folgenden Abschätzungen nur in Ihren Größenordnungen genau sind.

Vergleichsszenario: Pentium 4 Northwood

Der Pentium 4 (Northwood) wird in einer 130 nm Technologie gefertigt und belegt 146 mm^2 Siliziumfläche. Als FPGA-Vergleich dient eine neuronenparallele Implementierungen mit bitserieller Suche auf Basis eines Xilinx XC2V1500. Da der FPGA nur eine begrenzte Menge an lokalem Speicher hat (deutlich weniger als die CPU), werden zwei verschiedene, neuronen- und komponentenparallele ASIC Implementierungen untersucht, jeweils angepasst an das Speichervermögen des FPGAs und der CPU. Die Details der Implementierungen sind in Tabelle 5.5 dargestellt, Größe und Taktfrequenz der Prozessorelemente wurden mit Hilfe der CMOS Skalierungsregeln aus den in Kapitel 3 ermittelten Werten errechnet. Als Anwendungsszenario kann aufgrund der vorhandenen Speicherressourcen das in Abschnitt 3.4 beschriebene Beispiel nicht direkt übernommen werden; für diesen Vergleich wird die Karte von 40×40 Neuronen auf 10×20 reduziert, die dann nur noch 76 kByte Speicher für Referenzvektoren beansprucht. Tabelle 5.5 zeigt die Implementierungsdetails für die verschiedenen Varianten. Dabei ist zu beachten, dass die Fläche des FPGAs nicht exakt bekannt sind, sondern aufgrund einer Interpolation aus entsprechenden, mit Röntgenbildern vermessenen FPGAs aus derselben Produktfamilie abgeschätzt wurde. Darüber hinaus wurde bei der Bestimmung der maximalen Taktfrequenz der ASIC Implementierungen ausschließlich die entsprechende CMOS Skalierungsregel verwandt. Tatsächlich ist aufgrund der gemeinsamen Kommunikationskanäle eine geringere Taktfrequenz (oder eine tiefere Pipeline) zu erwarten.

Die Lernleistung und benötigte Energie für das Anwendungsbeispiel werden in den Abbildungen 5.6 (a) und (b) dargestellt. Dabei wurde jeweils zwischen einer eingebetteten und einer PC-basierten Lösung unterschieden. Bezüglich der Lernleistung fällt auf, dass die FPGA-basierte Lösung etwa doppelt so schnell arbeitet wie der PC, wobei sich eingebettete und PC-basierte Lösung kaum unterscheiden. Dies resultiert aus dem Verhältnis zwischen eigentlicher Lerndauer (ca. 525

	P_{cl} [MCUPS]	E [Ws]	T [s]	P [W]	A [mm^2]	RE
ASIC (a) eing.	$5{,}07{\cdot}10^4$	$95{,}51{\cdot}10^0$	7,9	12,1	146	$0{,}64{\cdot}10^{+0}$
ASIC(b) eing.	$5{,}07{\cdot}10^4$	$81{,}32{\cdot}10^0$	7,9	10,3	146	$0{,}75{\cdot}10^{+0}$
FPGA eing.	$7{,}61{\cdot}10^2$	$4{,}40{\cdot}10^4$	525,9	4,2	150	$0{,}27{\cdot}10^{-1}$
ASIC (a) PC	$3{,}78{\cdot}10^4$	$1{,}07{\cdot}10^3$	10,6	101,1	292	$0{,}29{\cdot}10^{-1}$
ASIC (b) PC	$3{,}78{\cdot}10^4$	$1{,}05{\cdot}10^3$	10,6	99,3	292	$0{,}29{\cdot}10^{-1}$
FPGA PC	$7{,}57{\cdot}10^2$	$4{,}71{\cdot}10^4$	528,63	93,2	296	$0{,}62{\cdot}10^{-5}$
CPU	$3{,}68{\cdot}10^2$	$9{,}66{\cdot}10^4$	1085,4	89,0	146	$0{,}63{\cdot}10^{-5}$

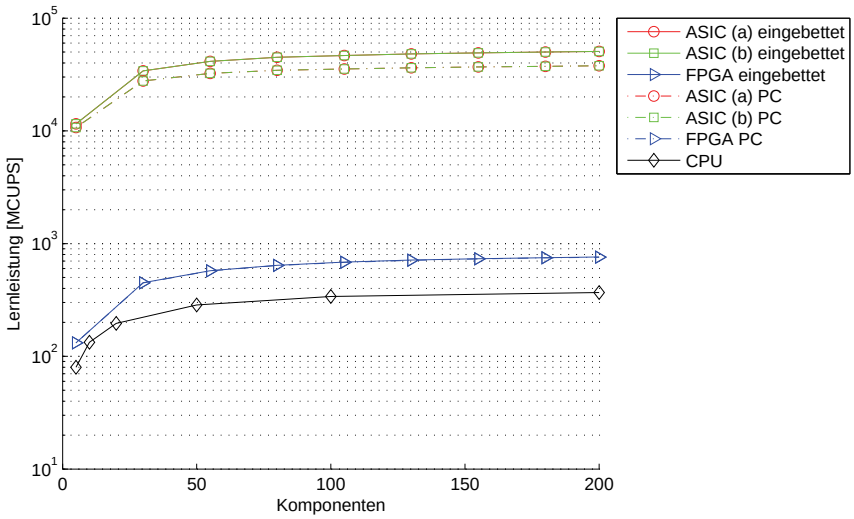
Tabelle 5.6: Ergebnisse des Vergleichs

s) und der Dauer der Datenübertragung (ca. 3 s). Bei den ASIC-basierten Lösungen, die einen etwa 130-fach größeren Durchsatz haben, ist der Unterschied wesentlich deutlicher, da hier die Lerndauer nur ca. 8 s beträgt.

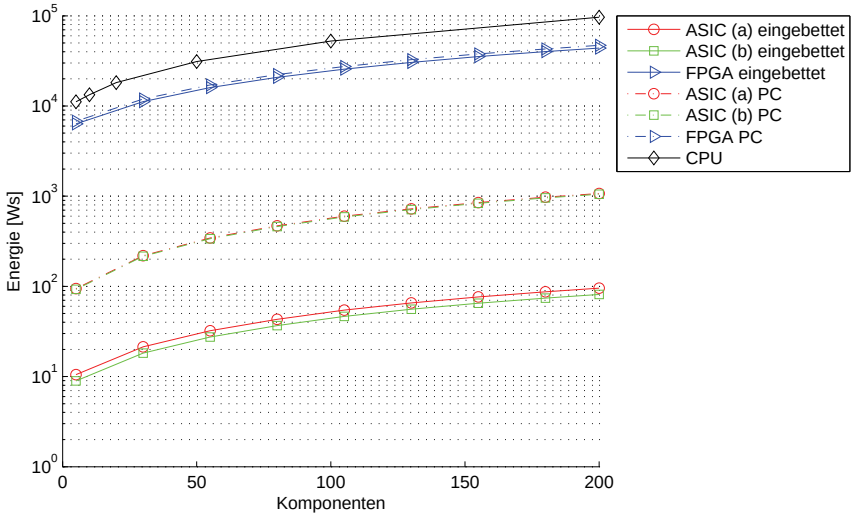
Bei der Betrachtung des Energiebedarfs wird sehr deutlich, dass unabhängig von Implementierungsvariante und Kopplung (eingebettet oder PC) FPGA und ASIC Varianten deutlich weniger Energie benötigen als der vergleichbare Prozessor. Das bedeutet, dass die Integration eines FPGA oder ASIC zur SOM-Beschleunigung für dieses Beispiel auf jeden Fall eine deutliche Energieersparnis bringt. Hier fällt zwar der Unterschied zwischen eingebetteter und PC-Variante deutlicher ins Gewicht als bei der Lernleistung, trotzdem braucht die FPGA Implementierung maximal die Hälfte der Energie, die ASIC Implementierung maximal 1,1%. Zur genaueren Betrachtung sind diese Messwerte in Tabelle 5.6 für das gewählte Szenario (mit 196 Komponenten) aufgetragen, zusätzlich dazu wurde die Ressourceneffizienz gemäß Gleichung 3.3 bestimmt. Erwartungsgemäß haben die beiden ASIC-basierten Implementierungen die höchste Ressourceneffizienz, wobei Variante (b) mit mehr Speicher und weniger Prozessoren die beste Implementierung ist. Zwischen der besten ASIC und der schlechtesten (CPU) Implementierung liegen fünf Größenordnungen, die PC-basierte FPGA Implementierung und die reine CPU Implementierung liegen nahezu gleich auf.

5.5 Ausblick auf zukünftige Technologien

Um die in dieser Arbeit präsentierten Architekturen und Entwürfe auch hinsichtlich ihrer Zukunftsfähigkeit beurteilen zu können, müssen Vorhersagen für die Entwicklungen der einzelnen Plattformen getroffen werden. Während dies für die reine ASIC-Implementierung über die CMOS-Skalierungsregeln und die International Technology Roadmap for Semiconductors (ITRS, Internationale Technologie Roadmap für Halbleiter)[22] relativ einfach durchzuführen ist, kann diese Vorhersage für Prozessoren und FPGAs nur sehr ungenau getroffen werden. Dies liegt zum einen daran, dass neu entwickelte Prozessoren und FPGAs häufig mehr



(a) Lernleistung



(b) Energieaufnahme

Abbildung 5.6: ASIC, FPGA und CPU mit gleicher Fläche und Technologie

als reine Portierungen zu einer neuen Technologie, sondern vielmehr architekturelle Neuerungen mit sich bringen. Zum anderen können Flaschenhälse, die in einer aktuellen Architektur identifiziert wurden, in der nächsten Generation völlig unkritisch sein, dafür können dann neue Flaschenhälse mit neuen Charakteristika gefunden werden. Im Folgenden werden für die drei Plattformen Voraussagen getroffen und hinsichtlich ihrer Zuverlässigkeit diskutiert.

Jahr	2009	2010	2011	2012	2013	2014
Technologie	32 nm	29 nm	27 nm	24 nm	22 nm	20 nm
Taktfrequenz [GHz]	1,41	1,55	1,67	1,88	2,05	2,25
Chipfläche [mm^2]	1,93	1,58	1,37	1,08	0,91	0,75
Leistungsaufnahme [mW]	305	276	257	229	210	190
Lernleistung [GCUPS]	13,8	15,3	16,4	18,4	20,1	22,1
Lernzeit [s]	224	203	189	168	154	140
Energiebedarf [Ws]	68,4	56,2	48,7	38,5	32,3	26,7

Tabelle 5.7: ASIC Implementierung mit zukünftigen Technologien

5.5.1 ASIC

Um einen realistischen Referenzpunkt für die Hochrechnung zu definieren, wird eine ASIC-basierte Implementierung mit einer Lernleistung $P_{Cl,ASIC} \geq P_{Cl,CPU(P4)}$ gewählt, also eine Implementierung die die gleiche wie oder eine bessere Lernleistung als der P4 Northwood Prozessor besitzt. In einer neuronen- und komponentenparallelen Implementierung kann dieses Ziel mit fünf Neuronen á vier Komponenten erreicht werden. Diese Implementierung und die Vorhersage bei der Portierung auf zukünftige Implementierungen wird in Tabelle 5.7 dargestellt.

5.5.2 Prozessoren

Die Beobachtungen in Abschnitt 5.4, nachdem die Lernleistung von Mehrkernprozessoren im Wesentlichen von der Datenmenge im Cache und damit von der Größe der SOM abhängen zeigen deutlich, dass der Geschwindigkeitszuwachs bei aktuellen Prozessoren nicht linear mit der Anzahl der Kerne wächst, jedenfalls nicht sofern nicht auch die Kartengröße mitwächst. Vielmehr strebt die Lernleistung einer Sättigung entgegen, welche nur durch eine Anpassung des Speichermanagements durchbrochen werden kann. Unterschreitet die Datenmenge im Cache eine bestimmte kritische Menge, so führt die Benutzung mehrerer Kerne zu einer Verringerung der Lernleistung.

Ausgehend von der Vermutung, dass die beobachtete kritische Datenmenge für den Übergang der Lernleistung in die Sättigung auch für zukünftige Systeme gültig ist, ist die Lernleistung für das vorgestellte Beispiel beschränkt. Eine genauere Betrachtung von Abbildung 5.5 (b) zeigt, dass sich die maximale Lernleistung einer solchen Plattform nicht jenseits von 2250 MCUPS befinden wird. Damit ist auch die eingebettete ASIC Variante (siehe Tabelle 5.6) in einer sehr alten Technologie (130 nm) noch deutlich schneller, als zukünftige Prozessoren sein können, natürlich unter der Voraussetzung, dass der limitierende Faktor Speichermanagement nicht aufgelöst werden kann.

Typ	Techn. [nm]	F [MHz]	Slices	FF	DSP48/ Mult	max PEs	$P_{C,l}$ [MCUPS]
XCV1000	180		12288	24576	0		
1 PE		46	* 522	336	0	23	$4{,}9{\cdot}10^2$
XC2V8000	120		46592	93184	168		
1 PE		124	* 340	337	1	137	$7{,}5{\cdot}10^3$
XC4VLX200	90		89088	178176	* 96		
1 PE		184	339	335	1	96	$7{,}9{\cdot}10^3$
XC5VLX330	65		207360	207360	* 192		
1 PE		245	412	335	1	192	$1{,}9{\cdot}10^4$
XC6VLX550T	40		343680	687360	* 864		
1 PE		240	377	355	1	864	$7{,}3{\cdot}10^4$

Tabelle 5.8: FPGA Implementierung eines PE mit aktuellen Technologien, limitierende Faktoren sind mit einem * gekennzeichnet, $P_{C,l}$ bezieht sich auf einen voll ausgelasteten FPGA

5.5.3 FPGA

Um den Einfluss zukünftiger FPGA-Architekturen für die Leistungsfähigkeit bei der Berechnung von SOM zu veranschaulichen, wurden Synthesen auf den aktuell verfügbaren FPGA-Familien von Xilinx, jeweils mit der höchstmöglichen Geschwindigkeitsstufe (engl. Speedgrade) und dem größten FPGA durchgeführt.

Die Ergebnisse dieser Synthesen sind in Tabelle 5.8 eingetragen und geben eine gute Vorstellung von der Entwicklung der FPGA Technologie. Die Betrachtung der Taktfrequenzen zeigt, dass die CMOS Skalierungsregeln auf diese Anzuwenden sind. Bezüglich der Anzahl der Logikblöcke (engl. Slices) und den darin enthaltenen Registern (engl. Flip Flops) lassen sich keine Aussagen über die Skalierungsregeln machen, da die entsprechende Siliziumfläche der FPGAs nicht vorliegt. Auffällig ist, dass der begrenzende Faktor bei den ersten beiden FPGA Familien die Logikblöcke sind, später wird die Anzahl der Prozessorelemente nur durch die Multipliziererblöcke bzw. DSP Blöcke begrenzt. Interessanter Weise bewegt sich die Anzahl der Logikblöcke bei allen Familien in einem relativ engen Bereich, obwohl die zugrundeliegende Architektur massiven Änderungen unterworfen wurde.

Insgesamt scheint sich die Anzahl der DSP Blöcke als begrenzender Faktor für die Anzahl der Prozessorelemente herauszustellen, eine Voraussage über zukünftige Architekturen lässt sich angesichts der Schwankungen allerdings kaum machen. Ein Vergleich der drei Plattformen muss sich also auf die hier aktuell vorhandenen Technologien beschränken.

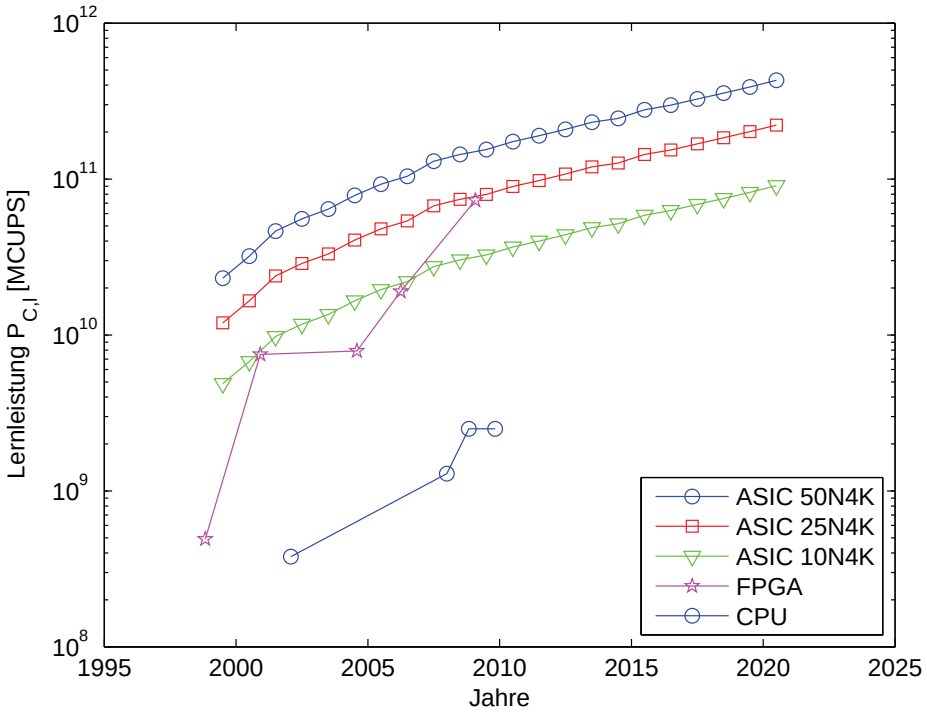


Abbildung 5.7: Lernleistung aktueller und zukünftiger Plattformen/Technologien

5.5.4 Vergleich

In Abbildung 5.7 wurden die Ergebnisse der einzelnen Abschnitte zur Lernleistung zusammengefasst, wobei für die ASIC Implementierung (willkürlich) eine neuronen- und komponentenparallele Implementierung in verschiedenen Konfigurationen ausgewählt wurde. Da die einzelnen Plattformen nicht auf Größe normiert werden können (die Größe der FPGAs ist weitgehend unbekannt), sind hier bestenfalls Trends für die einzelnen Technologien, wie sie bereits in den vorangegangenen Abschnitten beschrieben wurden, abzulesen. Es kann also keine sinnvolle Aussage darüber getroffen werden, ob und wann eine Plattform eine andere übertreffen wird, oder (mangels entsprechender Daten) wie sich die Ressourceneffizienz der Implementierungen zueinander verhält.

Die Erkenntnisse aus dieser Arbeit zeigen, dass es nicht *die* beste Lösung für die Berechnung von SOM gibt. Vielmehr gibt es eine Reihe von Pareto-optimalen Implementierungen, aus denen ausgewählt werden kann, wobei die Charakteristika gegeneinander abgewogen werden. Auch wenn keine detaillierte Analyse mit Hilfe der Ressourceneffizienz durchgeführt werden kann zeigen die Kurven in Abbildung 5.7, dass es für eine gewünschte Lernleistung viele Implementierungen gibt, und dass die zu erwartende Geschwindigkeit mit zukünftigen Technologien steigt.

	NVIDIA GT200	STI CELL	Xilinx Virtex6 SX475T
B_{mem} [TBit/s]	1,3	2,5	19,2
V_{mem} [MByte]	>512 (VideoRAM)	1,5 (Cache)	4,8 (BRAM)

Tabelle 5.9: Speicherbandbreite B_{mem} und -volumen V_{mem}

5.6 GPGPUs und andere Multiprozessoren

Im Rahmen dieser Arbeit wurden im Wesentlichen FPGAs und ASICs als Plattform für die Beschleunigung von selbst-organisierenden Karten untersucht, GPGPUs und weitere Multiprozessoren (etwa der STI Cell Prozessor [152]) wurden nur am Rande betrachtet. Tatsächlich ist für zahlreiche Anwendungen bekannt, dass gegenüber einer Standard Desktop-CPU enorme Beschleunigungen erreicht werden können [155, 42]. Für die SOM, die aus Sicht der Hardware nur sehr geringe Anforderungen stellt sind diese Plattformen zunächst überdimensioniert, da hier vor Allem mit Gleitkommaoperationen gerechnet wird. Die geringen Anforderungen bringen ebenfalls mit sich, dass die Lernleistung eines parallelen Hardwarebeschleunigers durch die Bandbreite der Speicherschnittstelle begrenzt wird. Im Falle der SOM ist dies der Speicher für die Referenzvektoren $\mathbf{m}_i$, der Angesichts der Masse der zu speichernden Daten in der Größenordnung ab einem Megabyte liegen sollte. In Tabelle 5.9 werden die Speicherbandbreiten aktueller Vertreter der jeweiligen Architektur verglichen. Es wird sofort ersichtlich, dass die Speicherbandbreite B_{mem} des FPGAs um eine Größenordnung über denen der Multiprozessoren liegen, wobei jeweils der am performantesten angebundene Speicher betrachtet wurde, der die geforderte Größe von $V_{mem} \geq 1$ MByte erfüllt. Das Beschleunigungspotential der FPGA Implementierung liegt damit deutlich über denen der Multiprozessoren, daher wurden diese hier nicht betrachtet.

5.7 Zusammenfassung

Auf Basis des im vorangegangenen Kapitels vorgestellten Entwurfsablaufs wurde eine prototypische Implementierung für eine Steuereinheit für die in Kapitel 3 vorgestellten Prozessorfelder entwickelt und vermessen. Die Messungen zeigen einerseits die Genauigkeit der theoretischen Voraussage zur Lernleistung, andererseits die Effekte einer Integration in ein PC-basiertes System. Schon die relativ kleine Konfiguration mit 72 Prozessorelementen bei 50 MHz hat das Potential, eine größere Lernleistung als aktuelle Desktopprozessoren zu vollbringen, auch wenn alle Aspekte einer Integration in einen PC berücksichtigt werden. Gleichzeitig wird schon in dieser FPGA-basierten Implementierung deutlich weniger Energie für die Berechnung benötigt als mit aktuellen Prozessoren.

Dieses Verhältnis spiegelt sich auch in der Ressourceneffizienz wider, die erwartungsgemäß für die ASIC-basierte Implementierung am höchsten ist. Die PC-Varianten sind um den Faktor 20 schlechter als ihre eingebetteten Pendants, damit aber weiterhin um den Faktor 5000 besser als die Softwareimplementierung. Da bei der Bewertung einer in einen PC integrierten Implementierung sowohl die Ressourcen für den Beschleuniger, als auch für die CPU berücksichtigt wurden, liegen Software-Implementierung und PC-basierte FPGA-Implementierung etwa gleich auf, obwohl die FPGA-Implementierung etwa die doppelte Lernleistung zeigt.

Insgesamt kann also festgestellt werden, dass sowohl ASIC, als auch FPGA-basierte SOM-Beschleuniger einen großen Geschwindigkeitsvorteil und eine ebenso großen Energieersparnis mit sich bringen. Eine wichtige Erkenntnis ist ebenfalls, dass bereits eine PC integrierte FPGA-Implementierung, die typischerweise als reiner Prototyp betrachtet wird, eine deutlich größere Ressourceneffizienz aufweist, als eine vergleichbare Softwareimplementierung, und das unter Berücksichtigung des Gesamtsystems.

Resümee und Ausblick

Im Folgenden werden zunächst die Inhalte dieser Arbeit zusammen gefasst und dann die Ergebnisse diskutiert.

Kapitel 1 beginnt mit einem Überblick über die Funktionalität des SOM-Algorithmus und den entsprechenden Anwendungsgebieten aus Medizin, Biologie, Pharmazie, Geologie etc. Dabei werden auch diverse Erweiterungen des Algorithmus betrachtet, die die Nutzbarkeit hinsichtlich der Anwendungsszenarien erhöhen, in dem sie bestimmte Charakteristika der Karte verändern oder steuerbar machen. Um diese und andere Effekte bewerten zu können, werden entsprechende Maße vorgestellt und ihre Aussagekraft diskutiert. Das Kapitel schließt mit einer Diskussion von SOM-Implementierungen auf diversen Plattformen (Software, analoge Hardware, ASIC, FPGA) aus den Jahren 1999 bis 2009, die in späteren Kapiteln als Referenzen dienen.

Aufbauend auf dieser Grundlage werden in **Kapitel 2** mögliche Anpassungen für digitale Hardwarerealisierungen und deren Effekte auf das Verhalten des Algorithmus diskutiert. Da eine theoretische Analyse derzeit noch nicht möglich ist, wird eine empirische Analyse mit einer Auswahl an Datensätzen durchgeführt. Bei dieser Analyse werden die hardwarespezifischen Anpassungen z.B. die Wortbreite der Verarbeitungseinheiten) als Parameter des Lernvorgangs modelliert und in zahlreichen Versuchen variiert. Mit Hilfe der im vorangegangenen Kapitel vorgestellten Maße werden dann die Auswirkungen dieser Anpassungen analysiert. Da die Auswirkungen der Variationen bei den ausgewählten Maßen völlig unterschiedlich darstellen, wird ein neues skalares Qualitätsmaß definiert, welches die ausgewählten Kriterien normiert und zu einer einzelnen Metrik Q zusammenführt. Die Analyse der mit Hilfe der Q-Metrik erzielten Ergebnisse führt zur Definition von Mindestanforderungen an eine SOM-Hardwareimplementierung, die von den Eigenschaften des Datensatzes abhängig ist.

Um Kosten und Performanz eines solchen Hardwarebeschleunigers abschätzen zu können, werden in **Kapitel 3** prinzipielle Architekturen für den Aufbau eines massiv-parallelen Hardwarebeschleunigers aufgezeigt und hinsichtlich ihres Ressourcenbedarfes modelliert. Als Basis für das Modell werden Ergebnisse aus

Synthesen in einer 90nm CMOS Technologie herangezogen, wobei alle Architekturen in verschiedenen Größen (Anzahl der parallelen Recheneinheiten) betrachtet werden. Anhand des Modells konnte gezeigt werden, dass die Menge der Paretooptimalen Implementierungen abhängig vom gewählten Anwendungsszenario ist, und damit jeweils neu betrachtet werden muss. In einem Vergleich der hier entwickelten Hardware mit den aus der Literatur bekannten Implementierungen zeigt sich, dass die hier entwickelte Hardware bis auf einen Spezialfall um Größenordnungen performanter ist als die Referenzen, wobei sich die benötigten Hardwareressourcen in ähnlichen Bereichen bewegen.

Für die so entwickelte Hardware werden in **Kapitel 4** Werkzeuge für die prototypische Implementierung auf Basis von FPGAs vorgestellt. Der Entwurfsablauf sieht dabei eine zweistufige Portierung vor, wobei zunächst die logische Funktionalität und anschließend das korrekte Zeitverhalten überprüft werden. Die Werkzeuge bieten für diese Tests einen hohen Grad an Automatisierung an, der mit Hilfe einer ebenfalls im Rahmen dieser Arbeit entwickelten Java-basierten Bibliothek realisiert wurde. Mit Hilfe dieser Werkzeuge wurden Schaltungsteile der SOM-Beschleunigersysteme getestet und teilweise in eine Matlab Umgebung integriert.

Der Entwurfsablauf mündet in **Kapitel 5** in einer prototypischen Implementierung einer Steuerung für die parallelen Prozessorelemente, die dann in verschiedenen Varianten vermessen werden. Es zeigt sich dabei unter anderem, dass schon eine verhältnismäßig kleine, FPGA-basierte Implementierung performanter als eine vergleichbare Software-basierte Lösung ist, wobei gleichzeitigt deutlich weniger Energie verbraucht wird; ASIC-basierte Lösungen sind entsprechend um weitere Größenordnungen schneller und energieeffizienter. Auch unter Berücksichtigung der Gesamtperformanz eines Beschleunigersystems, also PC mit integriertem Beschleuniger, ergibt sich schon bei der FPGA-Variante ein deutlicher Vorteil gegenüber der reinen Softwareimplementierung.

Die Kernaussagen dieser Arbeit können im Wesentlichen in zwei Punkten zusammengefasst werden:

1. Hardwarespezifische Anpassungen des SOM-Algorithmus haben nicht zwingend negative Auswirkungen auf das Verhalten des SOM-Algorithmus. Vielmehr ist festzustellen, dass für die untersuchten Datensätze Mindestanforderungen bezüglich der Wortbreite aufgestellt werden müssen, die das im Rahmen statistischer Abweichungen mögliche Einhalten eines Qualitätskriteriums sicherstellen. Eine der Mindestanforderungen ist dabei das Vorhandensein eines Multiplizierers für die Durchführung der Adaption, denn ohne diesen ist zwar eine Selbstorganisation feststellbar, allerdings sind die hier beobachteten Ergebnisse teilweise diametral verschieden zu denen einer Software-basierten Lösung (mit Gleitkommazahlen doppelter Präzision nach IEEE758). Diese Mindestanforderung widerspricht damit den Aussagen aus zahlreichen Publikationen zum Thema, siehe Tabelle 1.1.

2. Für das betrachtete Szenario sind die hier entwickelten FPGA- und ASIC-basierten Implementierungen jeweils deutlich effizienter als Softwarelösungen. Dies gilt sowohl für Lösungen, die in ein technisches System eingebettet wurden, als auch für solche, die in einen PC integriert wurden und dementsprechend mit zusätzlichem Ressourcenbedarf beaufschlagt werden müssen. Ein Vergleich von Leistungsdaten der drei Plattformen über die Zeit zeigt, dass aktuell und auf absehbare Zeit die Lernleistung bei FPGA und ASIC deutlich größer ist als bei den Software-basierten Lösungen, wobei gleichzeitig deutlich weniger Energie benötigt wird. Es ist hinsichtlich des allgemeinen Trends zur besseren Unterstützung paralleler Abläufe durch zukünftige Prozessorarchitekturen zu erwarten, dass die Lernleistung zukünftiger Prozessoren wieder steigt. Trotzdem ist deutlich zu erkennen, dass FPGA-basierte Beschleunigersysteme als Bestandteil eines PCs auch auf lange Sicht sinnvoll sind.

In der Literatur sind viele Quellen zu finden, bei denen spezielle Hardwarearchitekturen für die Berechnung neuronaler Netze im Allgemeinen und selbstorganisierender Karten im Speziellen eingesetzt werden. Dabei werden häufig eine Reihe von Annahmen getroffen, die bislang nicht auf ihren Wahrheitsgehalt untersucht wurden. Mit Hilfe der im Rahmen dieser Arbeit entwickelten Methode zur systematischen Untersuchung der Auswirkungen von hardwarespezifischen Anpassungen konnten einige dieser Annahmen bestätigt werden, andere wurden widerlegt. Anhand von Implementierungen, die diese Erkenntnisse umsetzen konnte gezeigt werden, dass spezialisierte Hardware deutlich weniger Ressourcen benötigt als die entsprechenden Software-basierten Lösungen, und dass dies unabhängig von den klassischen Szenarien gilt. Damit kann die oft vage geführte Diskussion um Sinn und Unsinn von Spezialhardware für neuronale Netze um einige belastbare Ergebnisse erweitert werden.

Für zukünftige Untersuchungen bieten sich im Wesentlichen vier Schwerpunkte an. Zum einen sind alle Untersuchungen bezüglich der Eigenschaften selbstorganisierender Karten mit diskretisierten Gewichten empirischer Natur. Dies ist nicht weiter verwunderlich, da die Theorie für selbst-organisierende Karten im Allgemeinen noch keine entsprechenden Ergebnisse vorweisen kann. Sobald die Ordnungseigenschaften der Karten aber theoretisch verstanden sind, können entsprechende Untersuchungen mit diskretisierten Gewichten noch detailliertere wie allgemeingültigere Ergebnisse liefern, die dann einen noch breiteren Einsatz der SOM rechtfertigen können.

Zum zweiten konnten speziell für Implementierungen ohne Multiplizierer, also Implementierungen bei denen die Adaptionswerte ausschließlich Zweierpotenzen annehmen können, interessante und reproduzierbare Eigenschaften bezüglich des Vergrößerungsfaktors beobachtet werden. Eine Untersuchung der zugrundeliegenden Wirkprinzipien könnte zu sehr einfachen Methoden für die Beeinflussung des

Vergrößerungsfaktors führen.

Zum dritten wurden Defizite im Speichermanagement moderner Prozessoren festgestellt, die die Geschwindigkeit bei der Berechnung der SOM begrenzen. Da die SOM als hochparalleler Algorithmus eine sehr breite Klasse von Anwendungen vertritt, sind detailliertere Untersuchungen bezüglich der Ursachen dieses Verhaltens notwendig.

Im Rahmen dieser Arbeit wurden vor Allem Architekturen für den eigentlichen Rechenkern eines SOM-Beschleunigers untersucht, die ebenfalls benötigten Kontrollstrukturen wurden dabei im Wesentlichen nur prototypisch implementiert. Neue Entwicklungen auf dem Gebiet der FPGA-PC Integration können hier aber völlig neue Wege aufzeigen: durch eine extrem enge Kopplung von Prozessor und FPGA, etwa durch den Prozessorbus, können Kontrollaufgaben ohne große Latenzen an den Prozessor abgegeben werden, so dass neue Hardware-Software Partitionierungen attraktiv werden können. Speziell für den SOM-Beschleuniger könnten etwa die Selektion von Lernvektoren oder die Bestimmung von Adaptionswerten durch den Prozessor erfolgen, wodurch die Flexibilität des Beschleuniger, vor Allem auch hinsichtlich der Implementierung neuer SOM-Varianten, verbessert wird. Eine detailliertere Beschreibung eines solchen Systems findet sich in [16]

Glossar

Abkürzungen

ACC ..... engl. Automatic Code Completion, automatische Quellcode Vervollständigung.

ACG ..... engl.Automatic Code Generation, automatische Quellcode Erzeugung.

ASIC ..... engl. Application Specific Integrated Circuit, Anwendungsspezifischer Integrierter Schaltkreis.

AST ..... engl. Abstract Syntax Tree, ein abstrakter Baum, der die syntaktischen Elemente eines Quellcodes enthält.

BDH ..... Abgewandelter SOM=Algorithmus von Bauer et al., mit dem sich der Vergrößerungsfaktor ρ einstellen lässt.

CPPS .... engl. Connection Primitives Per Second, Atomare Verbindungen pro Sekunde.

CPS ..... engl. Connections Per Second, Verbindungen pro Sekunde.

CSOM .... engl. Conscious SOM, SOM mit Gewissen.

CUPPS ... engl. Connection Update Primitives Per Second, Atomare Verbindungsanpassungen pro Sekunde.

CUPS .... engl. Connection Updates Per Second, Verbindungsanpassungen pro Sekunde.

DFG ..... Datenflussgraph.

DLL ..... engl. Delay Locked Loop, Verzögerungsregelschleife.

DMA ..... engl. Direct Memory Access, Direkter Speicherzugriff, ohne Beteiligung des Prozessors.

DTF ..... Differentielle Topografische Funktion.

DUT ..... engl. Design Under Test, zu testende Implementierung.

Epoche ... Eine Epoche bezeichnet k Lernschritte.

FIFO ..... engl. First In First Out, Speicherelement mit einem *älteste zuerst* Zugriffsschema.

FPGA .... engl. Field Programmable Gate Array, feldprogrammierbare Gatter-Matrix.

FSM ..... engl. Finite State Machine, endlicher Automat.

GDTF .... Gewichtete Differentielle Topografische Funktion.

GPGPU .. engl. General Purpose Graphics Processing Unit, Allzweck Graphikrecheneinheit.

HiL ..... engl. Hardware-in-the-Loop, Hardware in der (Simulations-) Schleife.

HiLDE ... engl. Hardware-in-the-Loop Design Environment, Entwicklungsumgebung für HiL.

HiLDEGART engl. HiLDE for Generic Active Real Time Test, HiLDE für generische Echtzeittests mit aktiver Parametrierung.

JTAG .... engl. Joint Test Action Group, Verfahren zum Debuggen von Hardware nach IEEE-Standard 1149.1.

KNN ..... Künstliche Neuronale Netze.

PAF ..... Passives Autofocus System.

PCI ..... engl. Peripheral Component Bus, Peripheriebus.

PIO ..... engl. Programmable Input Output, Programmierbarer (Prozessorgesteuerter) Dateneingang und -ausgang.

PLL ..... engl. Phase Locked Loop, Phasenregelschleife.

QWT ..... engl. Qt Widgets for Technical Applications, eine Bibliothek zur graphischen Darstellung von Messwerten.

RC^2 ..... engl. Reconfigurable Compute Cluster, ein eng vermaschtes Netz von FPGA basierten Rechenknoten [18]..

SiLLis .... engl. Simple Language for Listeners, Einfache Sprache für Busprotokollierer.

SLAM .... engl. Self Localisation And Mapping, Selbstlokalisierung und Kartographierung.

SOM ..... engl. self-organizing map, selbst-organisierende Karte.

U-Matrix . Unified Distance Matrix.

UMC ..... United Microelectronics Corporation, einer der weltweit größten Halbleiterhersteller.

VHDL .... engl. Very High Speed Integrated Circuit Hardware Description Language, eine vor allem in Europa verbreitete Hardwarebeschreibungssprache.

WDF ..... Wahrscheinlichkeitsdichtefunktion.

ASAP .... engl. As Soon As Possible, so schnell wie möglich.

EDA ..... engl. Electronic Design Automation, Automatisierte Elektronikentwicklung.

FLI ..... engl. Foreign Language Interface, ModelSim Schnittstelle zur Einbindung externer Anwendungen.

R2KLIB .. Softwarebibliothek zur Ansteuerung der RAPTOR Prototyping Systeme.

WTA ..... engl. Winner Take All, der Gewinner bekommt alles. Bezeichnung für den Mechanismus, der das Erregungszentrum findet.

Griechische Buchstaben

$\alpha(t)$ ..... Die Lernrate, die die Stärke der Adaption beschreibt. I.d.R. gilt $0 \leq \alpha(t) \leq 1$ und $\alpha(t)$ ist monoton fallend.

$\beta(t)$ ..... bestimmt die Adaption der sog. winning frequency $F_{C,i}$ beim Conscious SOM=Algorithmus.

η ..... Prozessparamter eines CMOS Prozess.

$\gamma(t)$ ..... bestimmt den Einfluss des Gewissens g_i bei der Conscious SOM=Algorithmus.

γ_A ..... Gewichtungsexponent für die Fläche im Rahmen der Ressourceneffizienz.

γ_C ..... Exponentielle Gewichtung des Maßes zur Annäherung an ρ in Q. Im Rahmen dieser Arbeit gilt $\gamma_C = 1$.

γ_H ..... Exponentielle Gewichtung der Entropie in Q. Im Rahmen dieser Arbeit gilt $\gamma_H = 1$.

γ_K ..... Exponentielle Gewichtung des Klassifizierungsfehlers in Q. Im Rahmen dieser Arbeit gilt $\gamma_K = 2$.

γ_L ..... Gewichtungsexponent für die Zeit/Latenz im Rahmen der Ressourceneffizienz.

γ_P ..... Gewichtungsexponent für die Verlustleistung im Rahmen der Ressourceneffizienz.

γ_Q ..... Exponentielle Gewichtung des Quantisierungsfehlers in Q. Im Rahmen dieser Arbeit gilt $\gamma_Q = 1$.

γ_T ..... Exponentielle Gewichtung des topografischen Fehlers in Q. Im Rahmen dieser Arbeit gilt $\gamma_T = 1$.

γ_t ..... Gewichtungsexponent für das skalare Maß ϕ zur Bestimmung der Topografischen Verwerfungen.

$\Phi_M^X(\kappa)$ .... Die Topografische Funktion der Karte M und des Datensatzes X nach Villmann [144].

$\phi(\gamma_t)$ ..... Skalares Maß für die Topografieerhaltung, definiert über der Topografischen Funktion.

ρ ..... Der Vergrößerungsexponent.

$\tau\Delta t$ ..... Diskrete Zeitfunktion.

Lateinische Buchstaben

A ..... Die Fläche.

C ..... Die Menge der Klassen oder Cluster in einem Datensatz.

c_i ..... Die Klasse des Vektors $\mathbf{x}_i$.

$D(\cdot)$ ..... Diskretisierungs- oder Quantisierungsfunktion.

d_m ..... Die Dimension der Referenzvektoren, auch Anzahl der Komponenten genannt.

d_G ..... Die Dimension des SOM=Gitters, i.d.R. $d_G \in [1, 2, 3]$.

E_K ..... Der Klassifizierungsfehler.

E_Q ..... Der Quantisierungsfehler.

E_T ..... Der Topografische Fehler, auch Topografischer Index genannt.

F ..... Die Frequenz.

$F_{c,i}$ ..... Die Häufigkeit, mit der das Neuron i Erregungszentrum war, engl. winning frequency.

f_{Lern} ..... Lernregel eines neuronalen Algorithmus.

F_{SDRAMI} . Taktrate einer SDRAM-Schnittstelle.

f_T ..... Die Funktion f_T bewertet die Topografieerhaltenden Eigenschaften eines Neurons.

f_{Trans} .... Transferfunktion eines Neurons bzw. eines neuronalen Netzes.

$g_i(t)$ ..... Das *Gewissen* eines Neurons beim Conscious-SOM=Algorithmus.

GD ..... Graphische Darstellung einer SOM als Distanzmatrix.

GK ..... Graphische Darstellung einer SOM als Komponentenkarte.

H ..... Die Entropie.

$h_{c,i}(t)$ .... Die Nachbarschaftsfunktion, die in Verbindung mit α die Adaption der Referenzvektoren beeinflusst.

k ..... Die Anzahl der Vektoren in X.

C_L ..... Prozessparamter eines CMOS Prozess.

L ..... Die Latenz.

l ..... Die Anzahl der Neuronen in einem Künstlichen Neuronalen Netzwerk.

l_A ..... Die Anzahl der Neuronen, die in einem Lernschritt adaptiert werden, also Neuronen mit $\alpha(t)h_{c,i} \neq 0$.

L_a ..... Latenz eines Addiererbaumes.

L_α ..... Latenz der Verteilung der Adaptionswerte.

L_k ..... Latenz eines Komparatorbaumes.

L_l ..... Die Dauer eines Lernvorgangs, zusammengesetzt aus Bestimmung des Erregungszentrums und Adaption.

L_r ..... Die Dauer der Bestimmung des Erregungszentrum.

$\mathbf{m}_i$ ..... Der Gewichts- oder Referenzvektor des Neurons i.

M ..... Die Menge der Referenzvektoren.

$\mathbf{m}_{c,i}$ ..... Der Referenzvektor des Erregungszentrums zum Datenvektor i.

M_{ref} ..... Die Menge der Referenzvektoren einer Karte, die als Referenz dient. In der vorliegenden Arbeit ist dies eine Karte, die mit doppelter Präzision gelernt wurde..

$\tilde{M}_{ref}$ ..... Die Menge der Referenzvektoren einer Karte, die durch nachträgliche Quantisierung der Referenzvektoren der Karte M_{ref} erzeugt wurde..

M_X ..... Die Menge der Referenzvektoren, die durch einen Lernvorgang mit dem Datensatz X entstanden sind.

N ..... Die Menge der Neuronen.

n_i ..... Das Neuron i aus der Menge der Neuronen N. $n_i =< r_i, \mathbf{m}_i >$.

O ..... Anzahl der Ausgänge eines DUT.

P ..... Die Leistungsaufnahme.

$p(X)$ ..... Die Wahrscheinlichkeitsdichtefunktion von X.

$P_{C,l}$ ..... Die Lernleistung in Verbindungsänderungen pro Sekunde [CUPS].

$P_{C,r}$ ..... Die Abfrageleistung in Verbindungen pro Sekunde [CPS].

$P_{\varnothing}$ ..... Verlustleistung von Prozessorelementen, die Aufgrund einer ungünstigen Abbildung einer Karte auf den Beschleuniger nicht zur Berechnung des Ergebnisses beitragen..

PE_k ..... Anzahl der Prozessorelemente, die komponentenparallel arbeiten.

PE_n ..... Anzahl der Prozessorelemente(-gruppen), die neuronenparallel arbeiten.

PE_{nk} ..... Anzahl der Prozessorelemente(-gruppen), die komponenten- oder neuronenparallel arbeiten.

Q ..... Metrik für die Qualität einer Karte, basiert auf den in Abschnitt 1.3.1 definierten Qualitätsmaßen.

q ..... Quantisierungsschritt.

R ..... Abtastrate bei der Beobachtung eines Systems.

$\mathbf{r}_c$ ..... Die Position des Erregungszentrums im SOM=Gitter.

RE ..... Die Ressourceneffizienz.

r_h ..... Der effektive Radius der Nachbarschaftsfunktion, also das größte r, für das gilt $h_{c,i}(r) \geq 0$.

$\mathbf{r}_i$ ..... Die Position des Neurons i im SOM=Gitter.

S ..... Anzahl der Simulationsschritte.

s ..... Freier Parameter, mit dem sich der Vergrößerungsexponent ρ bei der BDH-SOM einstellen lässt.

T ..... Die Latenz.

U_{DD} ..... Versorgungsspannung von Transistoren.

U_{th} ..... Schwellspanung von Transistoren, engl. threshold voltage.

v ..... Anzahl der virtuellen Ebenen bei einer Karte, die von einer Hardwareimplementierung aufgrund ihrer Größe nicht in einem Schritt bearbeitet werden kann.

V_b ..... Datenvolumen bei Standardübertragung.

V_e ..... Datenvolumen bei Ereignis-basierter Übertragung.

w ..... Die Wortbreite eines Signals (in Bit).

w_{fest} ..... Anzahl der Bits für eine Festkommadarstellung.

w_{gleit} ..... Anzahl der Bits in einer Gleitkommadarstellung.

w_{krit} ..... Wortbreite, unterhalb derer ein gefordertes Qualitätskriterium nicht mehr erfüllt werden kann.

w_{SDRAMI} . Wortbreite einer SDRAM-Schnittstelle.

$\mathbf{X}$ ..... Der Eingabedatensatz.

$\mathbf{x}_i$ ..... Der Vektor i aus dem Datensatz X.

z_b ..... Basis in einer Gleitkommadarstellung.

z_e ..... Exponent in einer Gleitkommadarstellung.

z_m ..... Mantisse in einer Gleitkommadarstellung.

z_q ..... Punkt, an dem sich die Genauigkeiten einer Festkommadarstellung und einer Gleitkommadarstellung mit gleicher Anzahl an Bits überschneiden.

z_s ..... Vorzeichen in einer Gleitkommadarstellung.

Literaturverzeichnis

Eigene Veröffentlichungen

[1] Marc Franzmeier, Christopher Pohl, Mario Porrmann, and Ulrich Rückert. Hardware Accelerated Data Analysis. In *Proceedings for the 4th International Conference on Parallel Computing in Electrical Engineering (PARALEC 2004)*, pages 309–314, Dresden, Germany, September 7-10 2004. [zitiert auf S. 28]

[2] Paolo Roberto Grassi, Christopher Pohl, Mario Porrmann, Ulrich Rückert, and Mohammed Abdel-Wahab. Reconfiguration viewer. In *Design Automation and Test in Europe, DATE*, Nice, France, April, 20.-24. 2009. University Booth. [zitiert auf S. 129]

[3] Paolo Roberto Grassi, M. Santambrogio, Jens Hagemeyer, Christopher Pohl, and Mario Porrmann. Sillis: A simplified language for monitoring and debugging of reconfigurable systems. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '09)*, pages 174–180, Las Vegas, USA, July, 13.-16. 2009. [zitiert auf S. 129]

[4] Paolo Roberto Grassi, Marco Domenico Santambrogio, Christoph Puttmann, Christopher Pohl, and Mario Porrmann. A High Level Methodology for Monitoring Network-on-Chips. In *Digest of the 3rd DATE Workshop on Diagnostic Services in Network-on-Chips (DSNOC)*, pages 79–97, Nizza, Frankreich, 24. April 2009. [zitiert auf S. 129]

[5] Carlos Paiz, Christopher Pohl, and Mario Porrmann. Reconfigurable Hardware in-the-Loop Simulations for Digital Control Design. In *3rd International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 39–46, Setubal, Portugal, August 2006. [zitiert auf S. 104]

[6] Carlos Paiz, Christopher Pohl, and Mario Porrmann. Hardware-in-the-Loop Simulations for FPGA-Based Digital Control Design. In *Informatics in*

Control, Automation and Robotics, volume 15, pages 355–372. Springer-Verlag, May 2008. [zitiert auf S. 104]

[7] Carlos Paiz, Christopher Pohl, and Mario Porrmann. FPGA-in-the-Loop Simulations with CAMEL-View. In *In Proceedings of the 7th International Heinz Nixdorf Symposium*, February 2008. [zitiert auf S. 106]

[8] Carlos Paiz, Jens Hagemeyer, Christopher Pohl, Mario Porrmann, Ulrich Rückert, Bernd Schulz, Wilhelm Peters, and Joachim Böcker. Fpga-based realization of self-optimizing drive-controllers. In *the 35th Annual Conference of the IEEE Industrial Electronics Society (IECON 2009)*. IEEE, November, 3.-5. 2009. [zitiert auf S. 113]

[9] Christopher Pohl. Leistungs- und Ressourcenbewertung verschiedener Realisierungsalternativen eines Prozessorelements für die Simulation neuronaler Netze. Studienarbeit, Schaltungstechnik, Heinz Nixdorf Institut, Universität Paderborn, März 2003. [zitiert auf S. 28]

[10] Christopher Pohl. Integration des gNBX-Prozessors in die Rapid-Prototyping-Umgebung RAPTOR2000. Diplomarbeit, Schaltungstechnik, Heinz Nixdorf Institut, Universität Paderborn, Oktober 2003. [zitiert auf S. 28]

[11] Christopher Pohl, Marc Franzmeier, Mario Porrmann, and Ulrich Rückert. gNBX Reconfigurable Hardware Acceleration of Self-Organizing Maps. In *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT'04)*, pages 97–104, Brisbane, Australia, December 6-8 2004. [zitiert auf S. 28, 29, 134, 136]

[12] Christopher Pohl, Carlos Paiz, and Mario Porrmann. Hardware-in-the-Loop Entwicklungsumgebung für informationsverarbeitende Komponenten mechatronischer Systeme. In *5. Paderborner Workshop Entwurf mechatronischer Systeme*, pages 69–79, March 22-23 2007. [zitiert auf S. 104]

[13] Christopher Pohl, Carlos Paiz, and Mario Porrmann. vMAGIC – VHDL Manipulation and Automation for Reliable System Development. In *Proceedings of the 3rd International Workshop on Reconfigurable Computing Education*, April 2008. [zitiert auf S. 124]

[14] Christopher Pohl, Carlos Paiz, and Mario Porrmann. A Hardware-in-the-Loop Design Environment for FPGAs. *Design Automation and Test in Europe DATE, University Booth, Munich, Germany*, March 10-14 2008. [zitiert auf S. 105, 113]

[15] Christopher Pohl, Ralf Fuest, and Mario Porrmann. Manageable dynamic reconfiguration with eve – extendable vhdl editor. In *Design Automation*

and Test in Europe, DATE, Nice, France, April, 20.-24. 2009. University Booth. [zitiert auf S. 129]

[16] Christopher Pohl, Jens Hagemeyer, Mario Porrmann, and Ulrich Rückert. Using a reconfigurable compute cluster for the acceleration of neural networks. In *Proceedings of the 2009 International Conference on Field-Programmable Technology (FPT '09)*, Sydney, Australia, December, 9.-11. 2009. [zitiert auf S. 156]

[17] Christopher Pohl, Carlos Paiz, and Mario Porrmann. vMAGIC - Automatic Code Generation for VHDL. *International Journal of Reconfigurable Computing*, Selected Papers from ReCoSoc08:9, March 2009. URL `http://www.hindawi.com/journals/ijrc/2009/205149.html`. [zitiert auf S. 125]

[18] Mario Porrmann, Jens Hagemeyer, Christopher Pohl, Johannes Romoth, and Manuel Strugholtz. RAPTOR – A Scalable Platform for Rapid Prototyping and FPGA-based Cluster Computing. In *Proceedings of the International Conference on Parallel Computing (ParCo2009)*, Lyon, France, September 1-4 2009. [zitiert auf S. 122, 158]

[19] Madhura Purnaprajna, Christopher Pohl, Mario Porrmann, and Ulrich Rueckert. Using Run-time Reconfiguration for Energy Savings in Parallel Data Processing. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA'04, July 13-16, 2009, Las Vegas, Nevada, USA*, 2009. [zitiert auf S. 61, 144]

Literatur

[20] S.D. Abourida, C. Belanger, and J.T.Y.T. Arasawa. Hardware-In-the-Loop Simulation of Finite-Element Based Motor Drives with RT-LAB and JMAG. *Industrial Electronics, 2006 IEEE International Symposium on*, 3, 2006. [zitiert auf S. 102]

[21] C.C. Aggarwal, A. Hinneburg, and D. Keim. *On the Surprising Behavior of Distance Metrics in High Dimensional Space*. Springer, 2000. [zitiert auf S. 5, 57]

[22] A. Allan, D. Edenfeld, W. Joyner, A. Kahng, M. Rodgers, and Y. Zorian. International Technology Roadmap for Semiconductors. *IEEE Comput*, 2002. [zitiert auf S. 146]

[23] S. Anzali, J. Gasteiger, U. Holzgrabe, J. Polanski, J. Sadowski, A. Teckentrup, and M. Wagener. The use of self-organizing neural networks in

drug design. *Perspectives in Drug Discovery and Design*, 9:273–299, 1998. [zitiert auf S. 11]

[24] A. Asuncion and D.J. Newman. UCI machine learning repository. *School of Information and Computer Sciences. University of California, Irvine, California, USA*, 2007. [zitiert auf S. 182]

[25] M. Bartu, J. Tuckova, and J. Stastny. An Accelerator for Kohonen Self-Organizing Maps. *The 3rd International Workshop Digital Technologies 2006*, 2006. [zitiert auf S. 27]

[26] Hans-Ulrich Bauer and Klaus R. Pawelzik. Quantifying the neighborhood preservation of Self-Organizing Feature Maps. *IEEE Transactions on Neural Networks*, 3(4):570–579, 1992. [zitiert auf S. 15]

[27] H.U. Bauer, R. Der, and M. Herrmann. Controlling the magnification factor of self-organizing feature maps. *Neural Computation*, 8(4):757–771, 1996. [zitiert auf S. 20, 23, 24, 84]

[28] T. Beck and S. ETAS. Current trends in the design of automotive electronic systems. *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, 2001. [zitiert auf S. 102]

[29] E. Berglund and J. Sitte. The parameterless self-organizing map algorithm. *IEEE Transactions on Neural Networks*, 17(2):305–316, 2006. [zitiert auf S. 84]

[30] P. Bhatti and B. Hannaford. Single chip velocity measurement system for incremental optical encoders. In 83-105, editor, *IEEE Transactions on Control Systems Technology*, June 1997. [zitiert auf S. 102]

[31] D. Blank, D. Kumar, L. Meeden, and J.B. Marshall. Bringing up robot: Fundamental mechanisms for creating a self-motivated, self-organizing architecture. *Cybernetics and Systems*, 36(2):125–150, 2005. [zitiert auf S. 12]

[32] M. Bode, O. Freyd, J. Fischer, F.J. Niedernostheide, and H.J. Schulze. Hybrid hardware for a highly parallel search in the context of learning classifiers. *Artificial Intelligence*, 130(1):75–84, 2001. [zitiert auf S. 27]

[33] P. Buckhaults, Z. Zhang, Y.C. Chen, T.L. Wang, B. St Croix, S. Saha, A. Bardelli, P.J. Morin, K. Polyak, R.H. Hruban, et al. Identifying Tumor Origin Using a Gene Expression-based Classification Map 1, 2003. [zitiert auf S. 10]

[34] A. Campbell, E. Berglund, and A. Streit. Graphics Hardware Implementation of the Parameter-Less Self-Organising Map. *Proc. IDEAL*, pages 343–350, 2005. [zitiert auf S. 27]

[35] Davide Candiloro. Management and analysis od bitstream generators for Xilinx FPGAs. Master's thesis, University of Illinois at Chicago, 2008. [zitiert auf S. 121]

[36] D. Carrica, M.A. Funes, and S.A. Gonzalez. Novel stepper motor controller based on fpga hardware implementation. In *IEEE/ASME Transactions On Mechatronics*, volume 8, pages 120–124. IEEE/ASME, March 2003. [zitiert auf S. 102]

[37] G.J. Chappell and J.G. Taylor. The temporal Kohonen map. *Neural Networks*, 6(3):441–445, 1993. [zitiert auf S. 25]

[38] C.Y. Chen and R.C. Hwang. An Embedded Lens Controller for Passive Auto-Focusing Camera Device Based on SOM Neural Network. *TENCON 2006. 2006 IEEE Region 10 Conference*, pages 1–4, 2006. [zitiert auf S. 11]

[39] J. Cho, JC Principe, D. Erdogmus, and MA Motter. Modeling and inverse controller design for an unmanned aerial vehicle based on the self-organizing map. *Neural Networks, IEEE Transactions on*, 17(2):445–460, 2006. [zitiert auf S. 12]

[40] WJ Cody. Analysis of Proposals for the Floating-Point Standard. *Computer*, 14(3):63–68, 1981. [zitiert auf S. 33]

[41] M. Cottrell, JC Fort, and G. Pages. Two or three things that we know about the Kohonen algorithm. *Proc. ESANN*, 94:235–244, 1994. [zitiert auf S. 22, 34]

[42] C.H. Crawford, P. Henning, M. Kistler, and C. Wright. Accelerating computing with the cell broadband engine processor. In *Proceedings of the 5th conference on Computing frontiers*, pages 3–12. ACM New York, NY, USA, 2008. [zitiert auf S. 151]

[43] R. Crosbie, J. Zenor, R. Bednar, and D. Word. High-speed, scalable, real-time simulation using dsp arrays. In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)*, pages 52–59. IEEE Computer Society, 2004. [zitiert auf S. 101]

[44] L. Dagum, R. Menon, and S.G. Inc. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1):46–55, 1998. [zitiert auf S. 142]

[45] DR Dersch and P. Tavan. Asymptotic level density in topological feature maps. *Neural Networks, IEEE Transactions on*, 6(1):230–236, 1995. [zitiert auf S. 24]

[46] D. DeSieno, HNC Inc, and CA San Diego. Adding a conscience to competitive learning. *Neural Networks, 1988., IEEE International Conference on*, pages 117–124, 1988. [zitiert auf S. 23, 84]

[47] M. Egmont-Petersen, D. de Ridder, and H. Handels. Image processing with neural networks—a review. *Pattern Recognition*, 35(10):2279–2301, 2002. [zitiert auf S. 12]

[48] E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybernetics*, 67(1): 47–55, 1992. [zitiert auf S. 6, 34]

[49] E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: stationary states, metastability and convergence rate. *Biological Cybernetics*, 67(1): 35–45, 1992. [zitiert auf S. 6, 34, 187]

[50] NR Euliano and JC Principe. Spatio-temporal self-organizing feature maps. *Neural Networks, 1996., IEEE International Conference on*, 4, 1996. [zitiert auf S. 25]

[51] J. M. Fernandes, M. Adamski, and A. J. Proenca. VHDL generation from hierarchical petri net specifications of parallel controllers. In *IEE Proceedings-E Computers and Digital Techniques*, volume 144, pages 127–137, March 1997. [zitiert auf S. 102]

[52] JA Flanagan. Self-organization in the SOM with a decreasing neighborhood-function of any width. *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470)*, 1, 1999. [zitiert auf S. 21]

[53] JC Fort. SOM's mathematics. *Neural Netw*, 2006. [zitiert auf S. 6, 20]

[54] J.C. Fort, P. Letremy, and M. Cottrell. Advantages and drawbacks of the Batch Kohonen algorithm. *ESANN'2002*, pages 223–230, 2002. [zitiert auf S. 27, 32]

[55] D.J. Frank, R.H. Dennard, E. Nowak, P.M. Solomon, Y. Taur, and Hon-Sum Philip Wong. Device scaling limits of si mosfets and their application dependencies. *Proceedings of the IEEE*, 89(3):259–288, Mar 2001. [zitiert auf S. 94]

[56] B. Fritzke. Growing cell structures—a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994. [zitiert auf S. 25]

[57] B. Fritzke. Growing Grid—a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, 2(5): 9–13, 1995. [zitiert auf S. 25]

[58] U. Gerecke and N. Sharkey. Quick and dirty localization for a lost robot. *Computational Intelligence in Robotics and Automation, 1999. CIRA'99. Proceedings. 1999 IEEE International Symposium on*, pages 262–267, 1999. [zitiert auf S. 12]

[59] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1998. [zitiert auf S. 3]

[60] D.C. Hendry, A.A. Duncan, and N. Lightowler. Ip core implementation of a self-organizing neural network. *IEEE Transactions on Neural Networks*, 14(5):1085–1096, 2003. [zitiert auf S. 94]

[61] DC Hendry, AA Duncan, and N. Lightowler. IP core implementation of a self-organizing neural network. *Neural Networks, IEEE Transactions on*, 14 (5):1085–1096, 2003. [zitiert auf S. 29, 95, 96]

[62] A. Hernandez, J. Urena, J.J. Garcia, M. Mazo, D. Hernanz, J.P. Derutin, and J. Serot. Ultrasonic ranging sensor using simultaneous emissions from different transducers. In *IEEE Transactions On Ultrasonics, Ferroelectrics, And Frequency Control*, volume 51, pages 1660–1670, December 2004. [zitiert auf S. 102]

[63] H. Hikawa. FPGA implementation of self organizing map with digital phase locked loops. *Neural Netw*, 18(5-6):514–22, 2005. [zitiert auf S. 27]

[64] A. Hinneburg, C.C. Aggarwal, and D. Keim. *What is the Nearest Neighbor in High Dimensional Spaces?* IBM TJ Watson Research Center, 2000. [zitiert auf S. 5]

[65] A. Hoekstra and M.F.J. Drossaers. An extended Kohonen feature map for sentence recognition. *Proc. ICANN*, 93:404–407, 1993. [zitiert auf S. 25]

[66] Y. Hong-Tzong, L. Ming-Tzong, C. Yao-Ter, and Y. Kuo-Chin. Design and implementation of real-time nurbs interpolator using a FPGA-based motion controller. In *IEEE International Conference on Mechatronics (ICM2005)*, pages 56–61, Taipei, Taiwan, July 2005. [zitiert auf S. 102]

[67] F. C. Hoppensteadt and E. M. Izhikevich. Pattern recognition via synchronization in phase-locked loop neural networks. *IEEE-NN*, 11(3):734, May 2000. [zitiert auf S. 27]

[68] D. Hristu-Varsakelis and WS Levine. *Handbook of networked and embedded control systems*. Birkhauser, 2005. [zitiert auf S. 102]

[69] S. J. Huang, T.M. Yang, and J.T. Huang. Fpga realization of wavelet transform for detection of electric power system disturbances. In *IEEE*

Transactions On Power Delivery, volume 17, pages 388–394, April 2002. [zitiert auf S. 102]

[70] R. Isermann and N. Müller. Design of computer controlled combustion engines. In *Mechatrnoics*, volume 13, page 1067–1089. Elsevier, December 2003. [zitiert auf S. 101]

[71] R. Isermann, J. Schaffnit, and J. Sinsel. Hardware-in-the-loop simulation for the design and testing of engine-control systems. In *Control Engineering Practice*, volume 7, pages 643–653. Elsevier, May 1999. [zitiert auf S. 100]

[72] H. Kalte, M. Porrmann, and U. Rückert. Rapid prototyping system für dynamisch rekonfigurierbare hardwarestrukturen. In *Proceedings of AES2000*, pages 150–157, Karlsruhe, January 2000. [zitiert auf S. 133]

[73] J. Kangas. On the Analysis of Pattern Sequences by Self-Organizing Maps. *PhDThesis, Helsinki University of Technology*, 1994. [zitiert auf S. 24]

[74] K.B. Kim, S. Kim, and G.H. Kim. Vector quantizer of medical image using wavelet transform and enhanced SOM algorithm. *Neural Computing & Applications*, 15(3):245–251, 2006. [zitiert auf S. 12]

[75] V. Kodaganallur. Incorporating Language Processing into Java Applications: A JavaCC Tutorial. *IEEE SOFTWARE*, pages 70–77, 2004. [zitiert auf S. 124]

[76] T. Kohonen. *Contentaddressable Memories.* Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1987. [zitiert auf S. 64, 185]

[77] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences.* Springer, Berlin, Heidelberg, second edition, 1997. [zitiert auf S. 1, 20, 35, 186]

[78] Teuvo Kohonen. Comparison of som point densities based on different criteria. *Neural Comput.*, 11(8):2081–2095, 1999. ISSN 0899-7667. doi: http://dx.doi.org/10.1162/089976699300016098. [zitiert auf S. 26]

[79] K. Kopecz. Unsupervised learning of sequences on maps with lateral connectivity. *Proceedings of the International Conference on Artificial Neural Networks*, 1:431–436, 1995. [zitiert auf S. 25]

[80] T. Koskela. *Time Series Prediction Using Recurrent SOM with Local Linear Models.* Helsinki University of Technology, 1997. [zitiert auf S. 25]

[81] K. Krishna and M.N. Murty. Genetic K-Means Algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, 29(3):433, 1999. [zitiert auf S. 13]

[82] W. Kurdthongmee. A novel Kohonen SOM-based image compression architecture suitable for moderate density FPGAs. *Image and Vision Computing*, 2008. [zitiert auf S. 12, 28, 29, 134, 135, 136]

[83] J. Kwiatkowski, M. Pawlik, U. Markowska-Kaczmar, and D. Konieczny. Performance Evaluation of Different Kohenen Network Parallelization Techniques. *Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)-Volume 00*, pages 331–336, 2006. [zitiert auf S. 26]

[84] K. L. Lian and P. W. Lehn. Real-time simulation of voltage source converters based on time average method. In *IEEE Transactions On Power Systems*, volume 20, pages 110–118. IEEE, February 2005. [zitiert auf S. 101]

[85] N. Lightowler, CT Spracklen, and AR Allen. A Modular Approach to Implementation of the Self-Organising Map. In *Proceedings of WSOM*, volume 9, pages 130–135. Citeseer, 1997. [zitiert auf S. 56, 58]

[86] C. F. Lin, C.Y. Tseng, and T.W. Tseng. A hardware-in-the-loop dynamics simulator for motorcycle rapid controller prototyping. In *Control Engineering Practice*. Elsevier, 2006. [zitiert auf S. 101]

[87] S. Lloyd. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, 28:129–137, 1982. [zitiert auf S. 13]

[88] Z. Luo, H. Liu, and X. Wu. Artificial neural network computation on graphic process unit. *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, 1, 2005. [zitiert auf S. 27]

[89] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1:281–297, 1967. [zitiert auf S. 7]

[90] D.T. Marr, F. Binns, D.L. Hill, G. Hinton, D.A. Koufaty, J.A. Miller, and M. Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6:4–15, 2002. [zitiert auf S. 142]

[91] T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994. [zitiert auf S. 16]

[92] N.J. Medrano-Marques and B. Martin-del Brio. Topology preservation in SOFM: an euclidean versus manhattan distance comparison. *Foundations and Tools for Neural Modeling. International Work-Conference on Artificial and Natural Neural Networks, IWANN'99. Proceedings,(Lecture Notes in Computer Science Vol*, 1606:601–9, 1999. [zitiert auf S. 34, 57]

[93] E. Merényi. Precision Mining of High-Dimensional Patterns with Self-Organizing Maps: Interpretation of Hyperspectral Images. *Quo Vadis Computational Intelligence: New Trends and Approaches in Computational Intelligence, Studies in Fuzziness and Soft Computing*, 54, 2000. [zitiert auf S. 84, 86]

[94] E. Merényi and A. Jain. Forbidden magnification? II. *European Symposium on Artificial Neural Networks*, pages 57–62, 2004. [zitiert auf S. 23, 37]

[95] E. Merényi, A. Jain, and T. Villmann. Explicit Magnification Control of Self-Organizing Maps for Forbidden Data. *IEEE Transactions on Neural Networks*, 18(3):786, 2007. [zitiert auf S. 11, 23, 24]

[96] S.G. Narendra and A.P. Chandrakasan. *Leakage in nanometer CMOS technologies*. Springer-Verlag New York Inc, 2006. [zitiert auf S. 75]

[97] P. S. B. Nascimento, P. R. M. Pand Maciel, M. E. Lima, R. E. Sant'ana, and A. G. S. Filho. A partial reconfigurable architecture for controllers based on petri nets. In *SBCCI '04: Proceedings of the 17th symposium on Integrated circuits and system design*, pages 16–21, New York, NY, USA, 2004. ACM Press. [zitiert auf S. 102]

[98] K. Nose and T. Sakurai. Optimization of VDD and VTH for low-power and high speed applications. In *Proceedings of the 2000 conference on Asia South Pacific design automation*, pages 469–474. ACM New York, NY, USA, 2000. [zitiert auf S. 73]

[99] Carlos Paiz. *Dynamically Reconfigurable Hardware for Embedded Digital Control Design*. PhD thesis, Electrical Engineering, 2010. [zitiert auf S. 100]

[100] T.J. Parr and R.W. Quong. ANTLR: A Predicated-LL (k) Parser Generator. *Software - Practice and Experience*, 25:789–810, 1995. [zitiert auf S. 124]

[101] V. Peiris, B. Hochet, and M. Declercq. Implementation of a fully parallel Kohonen map: a mixed analog digital approach. *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, 4, 1994. [zitiert auf S. 27]

[102] J. Pena, M. Vanegas, and A. Valencia. Digital Hardware Architectures of Kohonen's Self Organizing Feature Maps with Exponential Neighboring Function. *Reconfigurable Computing and FPGA's, 2006. ReConFig 2006. IEEE International Conference on*, pages 1–8, 2006. [zitiert auf S. 28, 29, 134, 136]

[103] B. Pino, FJ Pelayo, J. Ortega, and A. Prieto. Design and evaluation of a reconfigurable digital architecture for self-organizing maps. *Microelectronics for Neural, Fuzzy and Bio-Inspired Systems, 1999. MicroNeuro'99. Proceedings of the Seventh International Conference on*, pages 395–402, 1999. [zitiert auf S. 29]

[104] M. Plöger, H. Schütte, and F. Ferrara. Automatisierter HIL-Test im Entwicklungsprozess vernetzter, automotiver Elektroniksysteme. *Beitrag zur Autoreg (VDI/VDE)*, 2:04, 2003. [zitiert auf S. 102]

[105] S.L. Pomeroy, P. Tamayo, M. Gaasenbeek, L.M. Sturla, M. Angelo, M.E. McLaughlin, J.Y.H. Kim, L.C. Goumnerova, P.M. Black, C. Lau, et al. Prediction of central nervous system embryonal tumour outcome based on gene expression. volume 415, pages 436–442, 2002. [zitiert auf S. 11]

[106] M. Porrmann. *Leistungsbewertung eingebetteter Neurocomputersysteme: Dissertation.* Heinz Nixdorf Institut, Universität Paderborn, 2002. [zitiert auf S. 28, 72, 93]

[107] M. Porrmann, U. Witkowski, and U. Rückert. A massively parallel architecture for self-organizing feature maps. *IEEE Transactions on Neural Networks*, 14:1110–1121, 2003. [zitiert auf S. 28, 29, 94, 95, 96]

[108] Lutz Prechelt. Proben1: A set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, 1994. URL `citeseer.nj.nec.com/prechelt94proben.html`. [zitiert auf S. 41, 182]

[109] A. Ramirez-Agundis, R. Gadea-Girones, and R. Colom-Palero. A hardware design of a massive-parallel, modular NN-based vector quantizer for real-time video coding. *Microprocessors and Microsystems*, 32:33–44, 2008. [zitiert auf S. 12, 29, 134, 136]

[110] U. Rathmann, G. Vermeulen, M. Bieber, R. Dennington, and J. Wilgen. Qwt-Qt Widgets for technical applications. [zitiert auf S. 121]

[111] U. Rückert and U. Witkowski. Silicon artificial neural networks. In L. Niklasson, M. Boden, and T. Ziemke, editors, *Proceedings of ICANN98, International Conference on Artificial Neural Networks*, pages 75–84, Skövde, Sweden, 1998. Springer. [zitiert auf S. 28]

[112] Stefan Reichör, Martina Zeinzinger, and Markus Pfaff. Connecting reality and simulation: Couple high speed fpgas with your hdl simulation. In *Proceedings of the 14th IP-SOC Conference*, Paris, France, December 2005. [zitiert auf S. 102]

[113] H. Ritter. Asymptotic level density for a class of vector quantization processes. *Neural Networks, IEEE Transactions on*, 2:173–175, 1991. [zitiert auf S. 20, 46]

[114] S. Rüping, K. Goser, and U. Rückert. A chip for self-organizing feature maps. In *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pages 26–33, Los Alamitos, CA, USA, 1994. IEEE Comput. Soc. Press. [zitiert auf S. 28]

[115] A.A. Sadeghi. Self-organization property of Kohonen's map with general type of stimuli distribution. *Neural Networks*, 11:1637–1643, 1998. [zitiert auf S. 21]

[116] A.L.I.A. SADEGHI. Convergence in distribution of the multi-dimensional kohonen algorithm. *Journal of Applied Probability*, 38:136–151, 2001. [zitiert auf S. 6, 21]

[117] T. Sakurai and AR Newton. Alpha-power law mosfet model and its applications to cmos inverterdelay and other formulas. *IEEE Journal of Solid-State Circuits*, 25:584–594, 1990. [zitiert auf S. 73]

[118] U. Seiffert. Artificial neural networks on massively parallel computer hardware. *Neurocomputing*, 57:135–150, 2004. [zitiert auf S. 26, 27]

[119] U. Seiffert and B. Michaelis. Multi-dimensional Self-Organizing Maps on massively parallel hardware. *Advances in Self-Organising Maps*, pages 160–6, 2001. [zitiert auf S. 27]

[120] J. Setoain, C. Tenllado, M. Prieto, D. Valencia, A. Plaza, and J. Plaza. Parallel Hyperspectral Image Processing on Commodity Graphics Hardware. *ICPPW'06: Proceedings of the 2006 International Conference on Parallel Processing*, pages 465–472, 2006. [zitiert auf S. 11]

[121] J. Setoain, M. Prieto, C. Tenllado, and F. Tirado. Real-Time Onboard Hyperspectral Image Processing Using Programmable Graphics Hardware. *High Performance Computing in Remote Sensing*, 2007. [zitiert auf S. 11]

[122] P.S. Shiakolas and D. Piyabongkarn. Development of a real-time digital control system with a hardware-in-the-loop magnetic levitation device for reinforcement of controls education. In *Education, IEEE Transactions on*, volume 46, pages 50–60. IEEE, February 2003. [zitiert auf S. 101]

[123] N. Sudha. An ASIC implementation of Kohonen's map based colour image compression. *Real-Time Imaging*, 10:31–39, 2004. [zitiert auf S. 12, 28, 29]

[124] N. Sudha, T. Srikanthan, and B. Mailachalam. A VLSI architecture for 3-D self-organizing map based color quantization and its FPGA implementation. *Journal of Systems Architecture*, 48:337–352, 2003. [zitiert auf S. 12, 94, 95, 96]

[125] D. Suzuki and O. Hammami. SOM on multi-FPGA ISA board-hardware aspects. *Electronics, Circuits and Systems, 1999. Proceedings of ICECS'99. The 6th IEEE International Conference on*, 3, 1999. [zitiert auf S. 29]

[126] *Synplify DSP*. Synopsys. URL `http://www.synplicity.com/`. [zitiert auf S. 124]

[127] H. Takizawa and H. Kobayashi. Hierarchical parallel processing of large scale data clustering on a PC cluster with GPU co-processing. *The Journal of Supercomputing*, 36:219–234, 2006. [zitiert auf S. 27]

[128] H. Tamukoh, T. Aso, K. Horio, and T. Yamakawa. Self-organizing map hardware accelerator system and its application to realtime image enlargement. *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, 4, 2004. [zitiert auf S. 29, 134, 136]

[129] K. Tasdemir and E. Merényi. Exploiting data topology in visualization and clustering of Self-Organizing Maps. *IEEE Transactions on Neural Networks*, 20(4):549–562, 2009. [zitiert auf S. 17]

[130] K. Tazi, E. Monmasson, and J. P. Louis. Description of an entirely reconfigurable architecture dedicated to the current vector control of a set of ac machines. In *IEEE International Conference on Industrial Electronics, Control, and Instrumentation*, volume 3, pages 1415–1420, Novembre 1999. [zitiert auf S. 102]

[131] L. Terfloth and J. Gasteiger. Neural networks and genetic algorithms in drug design. *Drug Discovery Today*, 6:102–108, 2001. [zitiert auf S. 11]

[132] P. Thiran and M. Hasler. Study of the Kohonen network with a discrete state space. *Mathematics and Computers in Simulation*, 38:189–197, 1995. [zitiert auf S. 22, 34]

[133] A. Ultsch. Maps for the Visualization of high-dimensional Data Spaces. *Proc. Workshop on Self organizing Maps*, pages 225–230, 2003. [zitiert auf S. 7, 8]

[134] A. Ultsch. U*-matrix: a tool to visualize clusters in high dimensional data. *Dept. of Computer Science University of Marburg, Research Report*, 36, 2003. [zitiert auf S. 8]

[135] Alfred Ultsch. Knowledge extraction from self-organizing neural networks. In O. Opitz, B. Lausen, and R. Klar, editors, *Information and Classification*, pages 301–306, London, UK, 1993. Springer. [zitiert auf S. 7, 8]

[136] E. van Keulen, S. Colak, H. Withagen, and H. Hegt. Neural network hardware performance criteria. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1885–1888, June 1994. [zitiert auf S. 19]

[137] G. Vane, R.O. Green, T.G. Chrien, H.T. Enmark, E.G. Hansen, and W.M. Porter. The Airborne Visible/Infrared Imaging Spectrometer (AVIRIS). *Remote Sensing of Environment*, 44(2-3):127–143, 1993. [zitiert auf S. 11]

[138] M. Verleysen and D. François. The curse of dimensionality in data mining and time series prediction. *Lecture Notes in Computer Science (J. Cabestany, A. Prieto, and FS Hernandez, eds.)*, 3512:758–770, 2005. [zitiert auf S. 5]

[139] J. Vesanto. Using the SOM and local models in time-series prediction. *Proc. of Workshop on Self-Organizing Maps*, pages 209–214, 1997. [zitiert auf S. 24]

[140] J. Vesanto. SOM-based data visualization methods. *Intelligent Data Analysis*, 3(2):111–126, 1999. [zitiert auf S. 8, 9]

[141] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas. Self-Organizing Map in Matlab: the SOM Toolbox. *Proceedings of the Matlab DSP Conference*, pages 35–40, 1999. [zitiert auf S. 6, 10, 26, 134]

[142] J. Vesanto, J. Himberg, E. Alhoniemi, J. Parhankangas, S.O.M.T. Team, and L. Oy. Som toolbox for matlab. *Techn. Ber., Helsinki University of Technology*, 2000. [zitiert auf S. 35]

[143] C. Vijayakumar, G. Damayanti, R. Pant, and CM Sreedhar. Segmentation and grading of brain tumors on apparent diffusion coefficient images using self-organizing maps. *Computerized Medical Imaging and Graphics*, 31(7): 473–484, 2007. [zitiert auf S. 11]

[144] T. Villmann, R. Der, M. Herrmann, and TM Martinetz. Topology preservation in self-organizing feature maps: exact definition and measurement. *Neural Networks, IEEE Transactions on*, 8(2):256–266, 1997. [zitiert auf S. 15, 21, 160]

[145] T. Villmann, E. Merényi, and B. Hammer. Neural maps in remote sensing image analysis. *Neural Networks*, 16(3-4):389–403, 2003. [zitiert auf S. 11]

[146] P.M. Visser, M.A. Groothuis, and J.F. Broenink. FPGAs as versatile configurable IO devices in Hardware-in-the-Loop Simulation. *5th PROGRESS Symposium on Embedded Systems*, pages 41–44, 2004. [zitiert auf S. 102]

[147] P.M. Visser, M.A. Groothuis, and J.F. Broenink. Multi-Disciplinary Design Support using Hardware-in-the-Loop Simulation). *5th PROGRESS Symposium on Embedded Systems*, pages 206–213, 2004. [zitiert auf S. 102]

[148] T. Voegtlin. Recursive self-organizing maps. *Neural Networks*, 15(8-9):979–991, 2002. [zitiert auf S. 25]

[149] J. Wang, J. Delabie, H.C. Aasheim, E. Smeland, and O. Myklebost. Clustering of the SOM easily reveals distinct gene expression patterns: results of a reanalysis of lymphoma study. *BMC Bioinformatics*, 3, 2002. [zitiert auf S. 11]

[150] Felix Werner, Joaquin Sitte, and Frederic Maire. On the Induction of Topological Maps from Sequences of Colour Histograms. *Digital Image Computing Techniques and Applications, 9th Biennial Conference of the Australian Pattern Recognition Society on*, pages 167–174, 2007. [zitiert auf S. 12]

[151] S. White, J. O'Madadhain, D. Fisher, and Y.B. Boey. JUNG: Java Universal Network/Graph Framework, 2004. [zitiert auf S. 129]

[152] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick. The potential of the cell processor for scientific computing. In *Proceedings of the 3rd Conference on Computing Frontiers*, pages 9–20. ACM New York, NY, USA, 2006. [zitiert auf S. 151]

[153] Ulf Witkowski. *Einbettung selbstorganisierender Karten in Minirobotern.* PhD thesis, Universität Paderborn, Fachgebiet Schaltungstechnik, 2001. Wird veröffentlicht. [zitiert auf S. 28]

[154] W. Woo, M.D. Miller, and J. S. Kenney. A hybrid digital RF envelope predistortion linearization system for power amplifiers. In *IEEE Transactions On Microwave Theory And Techniques*, volume 53, pages 229–237, January 2005. [zitiert auf S. 102]

[155] R. Wu, B. Zhang, and M. Hsu. Clustering billions of data points using GPUs. In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, pages 1–6. ACM New York, NY, USA, 2009. [zitiert auf S. 151]

[156] *System Generator for DSP.* Xilinx. URL `http://www.xilinx.com/ise/optional_prod/system_generator.htm`. [zitiert auf S. 124]

[157] T. Yamakawa, K. Horio, and J.U.N.I. Takatori. Hardware Design of Self-Organizing Maps Using Pulse Width Modulation Based Signal Processing. *Faji Shisutemu Shinpojiumu Koen Ronbunshu*, 18:337–340, 2002. [zitiert auf S. 27]

[158] Y. Yankova, G. Kuzmanov, K. Bertels, G. Gaydadjiev, Y. Lu, and S. Vassiliadis. DWARV: Delftworkbench Automated Reconfigurable VHDL Generator. *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 697–701, 2007. [zitiert auf S. 124]

[159] X. Yue, D. M. Vilathgamuwa, and K.J. Tseng. Robust adaptive control of a three-axis motion simulator with state observers. In *IEEE/ASME Transactions On Mechatronics*, volume 10, pages 437–448. IEEE, August 2005. [zitiert auf S. 101]

[160] L. Zhang and E. Merényi. Weighted differential topographic function: A refinement of the topographic function. In *Proc. 14th European Symposium on Artificial Neural Networks (ESANN'2006)*, pages 13–18, 2006. [zitiert auf S. 17]

Anhang A

Weitere Untersuchungen SOM-Algorithmus

A.1 Automatische Klassifizierung von Datensätzen

Das Identifizieren von Clustern innerhalb eines Datensatzes ist das Haupteinsatzgebiet der SOM, und wird deshalb in Kapitel 2.1 als wichtigstes Maß für die Bewertung unterschiedlicher Implementierungen verwandt; dabei ist man in der Regel auf einen Datensatz angewiesen, dessen Cluster a-priori bekannt sind. Diese Vorgehensweise birgt zwei Probleme: Zunächst wird mit diesem Ansatz das Verhalten der SOM als Algorithmus bewertet, d.h. es wird überprüft, inwieweit die SOM die ursprüngliche Clusterung wiedergibt. Da diese u.U. mit Hilfe der SOM nicht wiedergegeben werden kann, kommt es zu sehr großen Fehlern, bei denen die eigentlich durch die hardwarespezifischen Anpassungen hervorgerufenen zusätzlichen Fehler im Rauschen untergehen. Die zweite Schwierigkeit liegt in der Verfügbarkeit von Datensätzen mit vorgegebener Clusterung. Stattdessen wird hier eine Vorgehensweise erläutert, die auf Basis einer SOM Referenzimplementierung eine Zuordnung zwischen Datenpunkten und Clustern erzeugt:

1. Der Datensatz wird in einen Trainings- und einen Validierungsdatensatz aufgeteilt.

2. Eine Referenz-SOM mit *double*-Präzision wird angelernt; anschließend werden die Referenzvektoren mit Hilfe eines k-Means Algorithmus in bis zu n Klassen segmentiert, n wird durch den Benutzer vorgegeben,

3. Für jede Segmentierung wird der Davies-Bouldin Clusterindex berechnet und so die beste Segmentierung ausgewählt

4. Die Referenzvektoren werden entsprechend der besten k-Means-Segmentierung bezeichnet; in einer anschließenden Abrufphase werden den Datenvektoren die Bezeichnungen ihrer Erregungszentren zugewiesen.

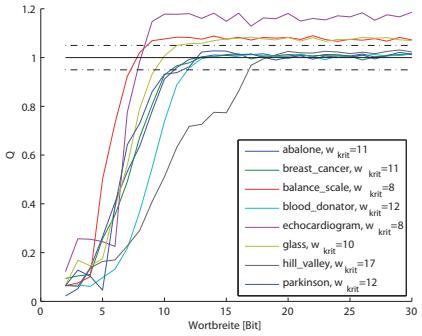
Auf diese Weise entsteht eine Klassifizierung unabhängig von externen Vorgaben, die zudem eine höhere Aussagekraft für den Vergleich unterschiedlicher Implementierungen besitzt. Es kann anstelle des Davies-Bouldin-Index auch ein anderes Klassenvaliditätsmaß verwendet werden.

A.2 Eigenschaften der verwendeten Datensätze

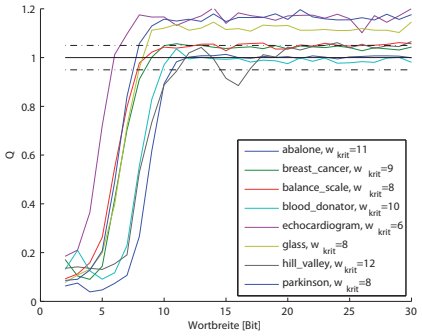
Bei den hier aufgelisteten Werten für die Anzahl der Vektoren k und deren Dimension d_m handelt es sich um die im Begleitmaterial zu den jeweiligen Datensätzen beschriebenen Werte. Die Anzahl der Cluster wurde wie im vorangegangenen Abschnitt beschrieben aus einer Referenzkarte entnommen.

Name	k	d_m	$\|C\|$	l
[24] Abalone	4177	8	3	19×17
[24] Balance Scale	625	4	2	14×9
[24] Blood Donator	748	4	2	17×8
[24] Breast Cancer	683	9	2	19×7
[108] Card	345	14	5	10×9
[108] Diabetes	384	8	3	9×11
[24] Echocardiogramm	61	10	5	8×5
[108] Flare	533	24	5	12×10
[108] Gene	1585	120	5	17×12
[24] Glass	214	10	3	12×6
[108] Heart	460	35	4	11×10
[24] Hill-Valley	1212	100	2	13×13
[108] Horse	182	58	2	7×10
[108] Mushroom	4062	125	4	19×17
[24] Parkinson	195	22	4	6×12
[108] Soybean	340	82	2	13×7

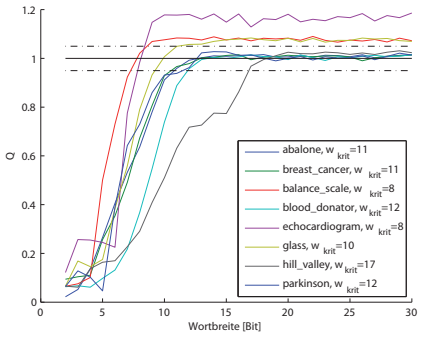
A.3 Untersuchung weiterer hardwarespezifischer Anpassungen



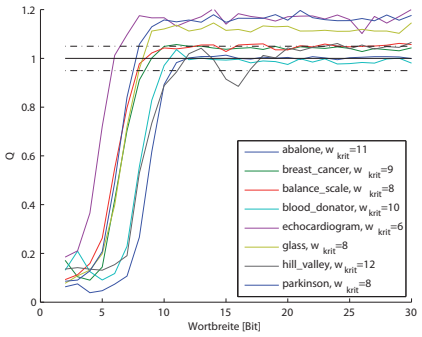
(a) KSOM mit $p1$-Norm für $|\vec{w} - \vec{m}|$



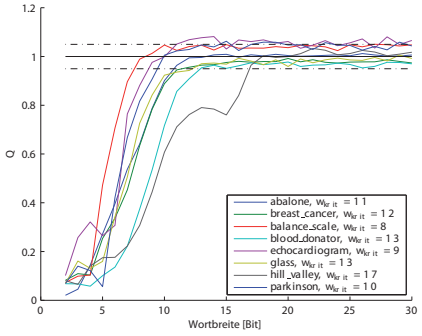
(b) CSOM mit $p1$-Norm für $|\vec{w} - \vec{m}|$



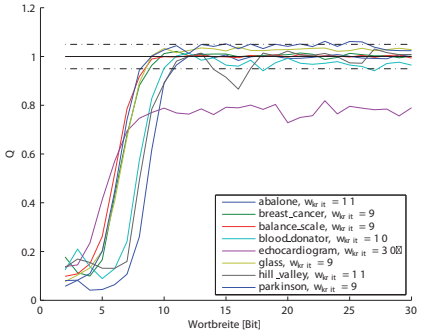
(c) KSOM mit wiederkehrender Reihenfolge



(d) CSOM mit wiederkehrender Reihenfolge



(e) KSOM mit quadratischer Adaptionsfunktion



(f) CSOM mit quadratischer Adaptionsfunktion

Abbildung A.1: Ergebnisse für die in Abschnitt 2.6 beschriebenen Variationen

Anhang B

Weitere Untersuchungen SOM-Implementierung

B.1 Modifizierte WTA Schaltung

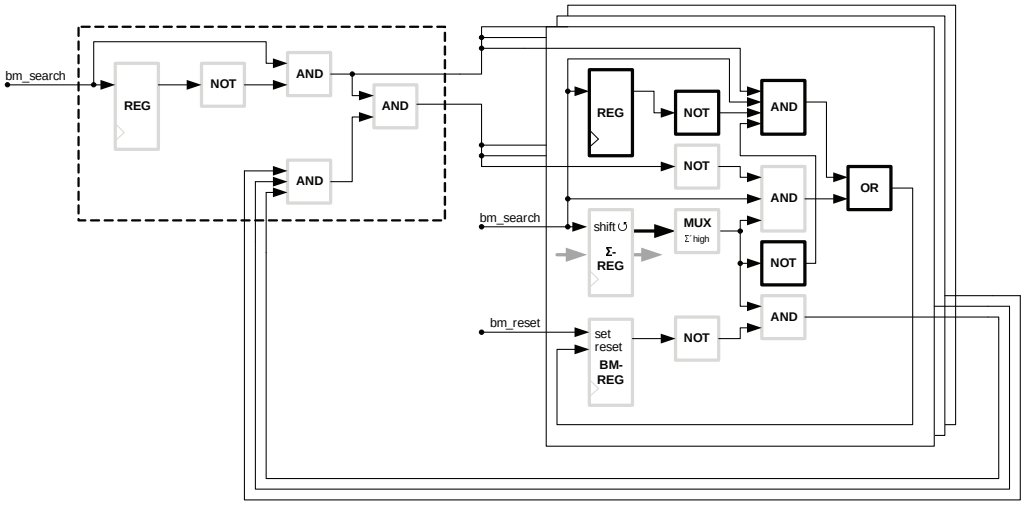
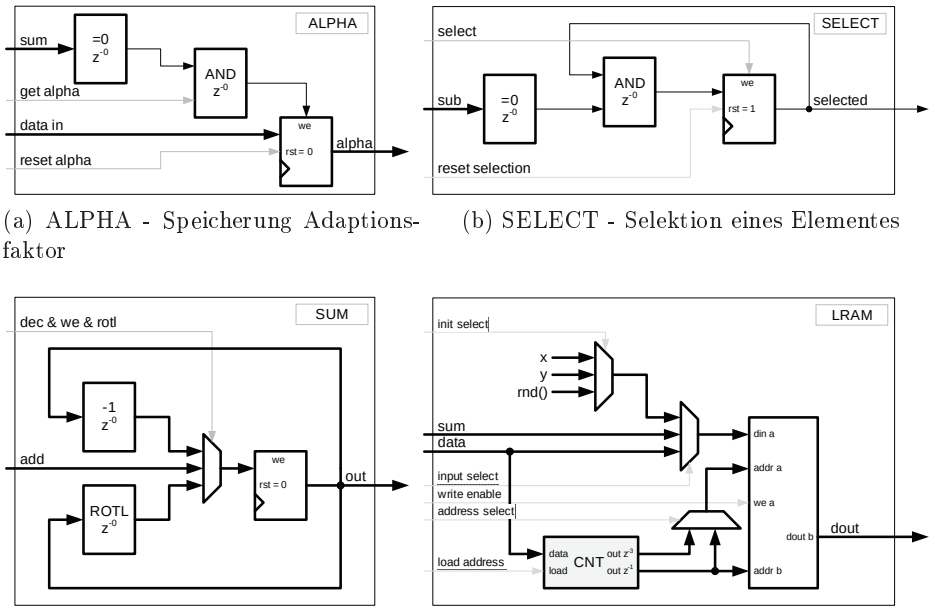


Abbildung B.1: Modifizierte WTA-Schaltung, links der globale, rechts die Neuronen-lokalen Teile

Die in [76] beschriebene Schaltung zur Suche nach dem Erregungszentrum ist ausschließlich für die Behandlung positiver Zahlen geeignet. Da bei der Berechnung der Conscious-SOM auch negative Zahlen berücksichtigt werden müssen, wurde eine modifizierte Variante der WTA Schaltung eingesetzt. In Abbildung B.1 wurden die Komponenten grau umrandet, die für die herkömmliche WTA-



(a) ALPHA - Speicherung Adaptionsfaktor

(b) SELECT - Selektion eines Elementes

(c) SUM - Summierung

(d) LRAM - Speicherung der Referenzvektoren

Abbildung B.2: Elemente des gNBX-Datenpfades

Schaltung benötigt werden. Da in diesem Fall die UND3 Komponente durch eine UND2 Komponente ersetzt werden kann, ergibt sich ein Mehraufwand von *einem 1-Bit Register, zwei 1-Bit Negierer und drei 1-Bit UND pro Neuron*. Zusätzlich verlängert sich der interne kritische Pfad um zwei Logikebenen. Da der interne kritische Pfad aber auch in diesem Fall deutlich kleiner ist, als die Pfade von den Neuronen zur Steuerung, fällt dieses nicht ins Gewicht.

B.2 Berechnung hexagonaler Gitter

Alle bisherigen Betrachtungen basieren auf der Berechnung selbst-organisierender Karten, denen eine quadratische Gitterstruktur zugrunde liegt. Dies ist allerdings keine Notwendigkeit, denn durch eine geeignete Transformation der Gitterkoordinaten kann auch eine hexagonale Struktur sehr effizient berechnet werden. Dies kommt nach Kohonen [77] dem Prozess der Visualisierung zu Gute, da im hexagonalen Fall die horizontalen und vertikalen Richtungen nicht bevorzugt werden.

Um eine hexagonale Struktur zu erzeugen, kann ein quadratisches Gitter als Ausgangsposition verwendet werden. Wenn für die Koordinaten auch Gleitkommawerte verwendet werden können, so werden die ursprünglichen Gitterkoordinaten wie folgt transformiert:

$$\hat{y} = \sqrt{0,75}y \text{ und } \hat{x} = \begin{cases} x & \text{falls } x \bmod 2 = 0 \\ x + 0,5 & \text{sonst} \end{cases} \tag{B.1}$$

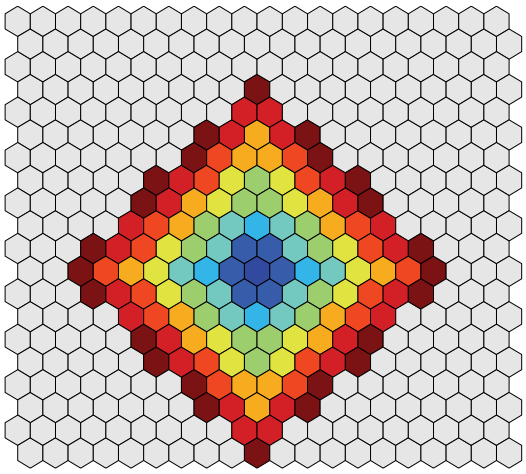


Abbildung B.3: Adaption in einem hexagonalen Gitter nach Gleichung B.2

Bei Verwendung dieser Transformation entsteht ein Gitter, bei dem benachbarte Neuronen immer einen euklidischen Abstand von 1 haben. Da in der oben beschriebenen Hardwareimplementierung keine Gleitkommawerte zur Verfügung stehen, muss eine Transformation verwendet werden, die sich auf ganzzahlige Werte beschränkt. Dazu wird die Transformation aus Gleichung B.1 äquivalent umgeformt in

$$\hat{y} = 8\sqrt{0,75}y \approx 7y \text{ und } \hat{x} = \begin{cases} 8x & \text{falls } x \bmod 2 = 0 \\ 8x+4 & \text{sonst} \end{cases} \tag{B.2}$$

Die Verzerrung durch die Approximation für $\hat{y}$ verursacht einen relativen Fehler von etwa 1,03% gegenüber der exakten Lösung.

In Bild B.3 wird eine hexagonale Karte dargestellt, die nach Gleichung B.2 aus einer 20×20 Neuronen großen rechteckigen Karte erstellt wurde. Das Element mit den Koordinaten $[84, 84]$ wurde als Erregungszentrum angenommen, und die Adaptionskoeffizienten mit $c_{dec} = 3$ auf die Neuronen verteilt. Die Neuronen wurden in Abhängigkeit von ihrem Adaptionskoeffizienten eingefärbt, so dass Elemente mit gleichen Adaptionsfaktoren gut zu erkennen sind. Daraus ergibt sich für die Anzahl der mit n Werten adaptierten Neurone N_{adapt}:

$$N_{adapt} \leq \frac{c_{dec}}{8}(1 + 4\frac{n(n+1)}{2} - 4n) \tag{B.3}$$

Die Anzahl der pro Wert adaptierten Neurone ist also abhängig von c_{dec}, allerdings kann für $c_{dec} > 1$ nicht mehr garantiert werden, dass die effektive Adaptionsfunktion konvex ist, siehe [49].

Der Aufwand für die Berechnung von selbst-organisierenden Karten mit hexagonaler Gitterstruktur beschränkt sich damit auf die korrekte Zuweisung von Koordinaten bei der Initialisierung, alle anderen Bereiche der Hardware bleiben unberührt. Zusätzlich muss seitens des Anwenders sichergestellt werden, dass die Adaptionskoeffizienten an die Abstände im Gitter angepasst werden.

Abbildungsverzeichnis

Tabellenverzeichnis