Zur Validierung der Spezifikationen von videobildverarbeitenden Komponenten unter realen Anwendungsbedingungen

Dissertation

zur Erlangung des Doktorgrades
am Fachbereich Informatik der Universität Hamburg
vorgelegt von

Lars Larsson

April 2000

Genehmigt vom Fachbereich Informatik der Universität Hamburg

auf Antrag von Prof. Dr.-Ing. K. Lagemann (Referent)

und Prof. Dr.-Ing. D. P. F. Möller (Korreferent)

Hamburg, den 26. April 2000 (Datum der Disputation)

Prof. Dr. Leonie Dreschler-Fischer
Die Dekanin des Fachbereichs Informatik

Dem Andenken meines Vaters.

Kurzfassung

Die Abbildung einer originären Spezifikation in Form einer Verhaltensbeschreibung auf eine Zieltechnologie durch Logiksynthese ist mittlerweile weitgehend automatisiert. Leistungsfähige Logiksimulatoren stehen seit vielen Jahren zur Verfügung. Prototypen funktionieren aber trotz Validierung durch Simulation häufig nicht in einer realen Systemumgebung [Keating und Bricaud, 1998]. Mit formalen Verifikationsmethoden kann Äquivalenz zwischen Implementierungen oder verschiedenen Repräsentationen einer Implementierung nachgewiesen werden [Kropf, 1998]. Ob jedoch eine originäre Spezifikation das vom Entwerfer beabsichtigte Verhalten in realen Anwendungsumgebungen repräsentiert, läßt sich damit jedoch nicht zeigen.

Bei neuronalen Netzwerken und Komponenten zur digitalen Videosignalverarbeitung besteht zudem das Problem, daß geringfügig unterschiedliche Dateneingaben, wie sie in realen Anwendungsumgebungen zwangsläufig auftreten, zu denselben Ausgaben führen sollen. Betreffende Implementierungen sollen in realen Anwendungsumgebungen robustes Verhalten aufweisen. Außerdem werden große Datenmengen verarbeitet, die zur Validierung in geeigneter Weise aufzubereiten sind. Es sind also neue Methoden erforderlich, um originäre Spezifikationen von Komponenten, die direkt mit realen Anwendungsumgebungen interagieren, zu validieren.

Inspiriert durch die visomotorische Fahrzeugsteuerung mit einem neuronalen Netzwerk namens *ALVINN* [Pomerleau, 1989, 1993], das auf dem Backpropagation-Algorithmus [Rumelhart *et al.*, 1986a,b] basiert, wurde ein Architekturkonzept zur Echtzeitverarbeitung von Videobildern mit einem Backpropagation-Netzwerk unter dem Namen *NeNEB* eingeführt [Larsson *et al.*, 1996, 1997a,b] und als Machbarkeitsstudie im Rahmen einer Diplomarbeit implementiert [Krol, 1996]. Dieses Architekturkonzept sieht die vollständig digitale Implementierung auf einem Chip vor und umfaßt auch digitale Videosignalvorverarbeitung, um den möglichen Flaschenhals der Übertragung und Zwischenspeicherung von Bilddaten zu vermeiden.

Der neue Ansatz der vorliegenden Arbeit ist die Berücksichtigung der realen Anwendungsumgebung bei der Validierung originärer Spezifikationen einerseits von neuronalen Netzwerken und andererseits von videosignalverarbeitenden Komponenten. Dazu wird jeweils das Verhalten spezifizierter Komponenten in realen Anwendungsumgebungen beobachtet und mit erwartetem Verhalten verglichen. Vor diesem Hintergrund wurde im Rahmen der vorliegenden Arbeit eine neue Methode zur Visualisierung des Lernvorgangs von Backpropagation-Netzwerken unter dem Namen *Lerntrajektorien* eingeführt [Larsson, 1997, 1999a]. Zur Entwicklung und Validierung digitaler, videosignalverarbeitender Komponenten wurde eine bidirektionale Schnittstelle namens *VidTrans* als Brücke zwischen Simulationen und Realsystemen geschaffen [Larsson, 1996]. Die als Fallbeispiel realisierte Hardware-Implementierung eines digitalen PAL-Farbvideosignal-Encoders [Larsson, 1999b] namens *PalCo* zeigt durch Vergleich mit einer zuvor realisierten Software-Implementierung eines digitalen PAL-Encoders die Notwendigkeit der Berücksichtigung realer Anwendungsumgebungen bei der Validierung originärer Spezifikationen auf.

Der wissenschaftliche Beitrag der vorliegenden Arbeit liegt u.a. in der Bereitstellung von neuen Methoden zur Parametrierung und Validierung von Backpropagation-Netzwerken sowie digitalen, videosignalverarbeitenden Komponenten zur Echtzeitverarbeitung von Videobildern unter Berücksichtigung realer Anwendungsumgebungen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Es ist eine der zentralen Aufgaben der Informatik, Methoden bereit zu stellen, welche die Handhabung komplexer informationsverarbeitender Systeme ermöglichen oder erleichtern und darüber hinaus neue Methoden für den Entwurf immer komplexerer Systeme zu erarbeiten. Die treibende Kraft für die steigende Komplexität informationsverarbeitender Systeme ist der technologische Fortschritt der Mikroelektronik in Bezug auf Verarbeitungsgeschwindigkeit und Integrationsdichte.

Beim Entwerfen mikroelektronischer Systeme tritt aber anstelle der effizienten Nutzung technologischer Ressourcen die effizientere Gestaltung des Entwurfsvorgangs, auch auf Kosten technologischer Ressourcen, da die Vergrößerung von Entwurfsteams nicht zu einer beliebigen Steigerung der Entwurfsperformanz für den Entwurf von Chips mit vielen Millionen Logikgattern führt [Keating und Bricaud, 1998]. Die Steigerung der Entwurfsperformanz, z.B. durch Wiederverwendbarkeit und Wartbarkeit von validierten Schaltungsmodulen, ist erforderlich, um komplexe Chips für neue Anwendungsbereiche in Zukunft überhaupt noch in vertretbarer Zeit entwerfen zu können – *Time-to-Market*.

Die Mikroelektronik ermöglicht für manche Anwendungen durch Hardware-Implementierung neben der Miniaturisierung informationsverarbeitender Systeme eine Steigerung der Verarbeitungsgeschwindigkeit gegenüber einer Software-Implementierung um mehrere Größenordnungen. Die Steigerung der Verarbeitungsgeschwindigkeit wird dabei durch Implementierung anwendungsspezifisch, parallel arbeitender Komponenten erreicht. Dadurch wird für bestimmte Algorithmen die Echtzeitverarbeitung erst ermöglicht. Eine effiziente Hardware-Implementierung erfordert in der Regel einen Übergang von Gleitkommazahlendarstellung zu Festkommazahlendarstellung, und sie ist unter Umständen zwar kompatibel zur Software-Implementierung, aber nicht auch äquivalent. Manche Algorithmen sind schon von vornherein für eine Hardware-Implementierung geeignet. Ein Beispiel dafür ist der DES-Verschlüsselungs-Algorithmus, für den durch Hardware-Implementierung eine Steigerung der Verarbeitungsgeschwindigkeit um viele Größenordnungen erreicht wurde [Foundation, 1998].

Die Umsetzung einer *originären Spezifikation* in Form einer abstrakten Verhaltensbeschreibung in eine geeignete Struktur, die das Verhalten der originären Spezifikation repräsentiert, ist dank leistungsfähiger Entwurfswerkzeuge weitgehend automatisiert. Formale Verifikationsmethoden ermöglichen den maschinellen Nachweis der Korrektheit eines Entwurfs in Bezug auf

eine gegebene Spezifikation [Kropf, 1998]. Durch den technologischen Fortschritt der Mikroelektronik werden jedoch Realisierungen mikroelektronischer Systeme ermöglicht, die direkt mit realen Anwendungsumgebungen interagieren. Die direkte Interaktion von mikroelektronischen Systemen mit realen Systemkomponenten erfordert jedoch neue Methoden zur Validierung, da sich reale Systemkomponenten einer realen Anwendungsumgebung allenfalls unvollständig spezifizieren lassen.

Der neue Ansatz der vorliegenden Arbeit ist die *systemnahe Validierung originärer Spezifikationen* durch Beobachtung des Verhaltens implementierter Systemkomponenten auf der Systemebene im Kontext des Entwurfs digitaler Schaltungen. Ausgehend von einem Architekturkonzept zur Echtzeitverarbeitung von Videobildern mit neuronalen Netzwerken [Larsson *et al.*, 1996, 1997a,b] wurde eine neue Methode zur Visualisierung des Lernvorgangs von Backpropagation-Netzwerken unter dem Namen *Lerntrajektorien* eingeführt [Larsson, 1997, 1999a]. Zur Entwicklung und Validierung digitaler, videosignalverarbeitender Komponenten wurde eine bidirektionale Schnittstelle als Brücke zwischen Simulationen und Realsystemen geschaffen [Larsson, 1996]. Der Erfolg *systemnaher Validierung* wurde abschließend mit einem Fallbeispiel durch Implementierung eines digitalen PAL-Farbvideosignal-Encoders praktisch demonstriert [Larsson, 1999b]. Damit wurde die Grundlage für den Entwurf und die Validierung von Systemen zur Echtzeitverarbeitung von Videobildern mit neuronalen Netzwerken geschaffen.

1.1 Aufgabenstellung

Für effiziente Hardware-Implementierung von neuronalen Netzwerken oder von Algorithmen der digitalen Signalverarbeitung ist in der Regel Festkommadarstellung oder Ganzzahldarstellung erforderlich, da die Hardware-Implementierung von Gleitkommarechenwerken, wie sie in Prozessoren implementiert sind, sehr aufwendig ist. Algorithmen, die auf Gleitkommazahlendarstellung basieren, sind daher zur Hardware-Implementierung geeignet zu ändern. Der Übergang von einer Gleitkommazahlendarstellung zur Festkommadarstellung oder zur Ganzzahldarstellung verändert jedoch das Verhalten des betreffenden Algorithmus. Software-Implementierung und Hardware-Implementierung sind dann nicht mehr äquivalent, sondern sie zeigen nur noch *funktional ähnliches Verhalten*, was aber für den praktischen Einsatz einer Hardware-Implementierung tolerabel ist, wenn diese in der realen Anwendungsumgebung robustes Verhalten zeigt. Hardware-Implementierung und Software-Implementierung müssen dann nicht äquivalent sein, sondern es genügt, wenn die Implementierungen auf den verschiedenen Plattformen kompatibel sind.

Neuronale Netzwerke und videosignalverarbeitende Komponenten zeichnen sich gerade dadurch aus, daß sich die Informationsverarbeitung bei der Anwendung in realen Systemumgebungen über viele Verarbeitungsschritte erstreckt. Bei Simulationen können Zwischenergebnisse von solchem Umfang entstehen, daß eine manuelle Auswertung von Detailergebnissen nicht mit vertretbarem Aufwand realisierbar ist. Die maschinelle Auswertung von Zwischenergebnissen ist dabei allenfalls für eine abschließende Validierung eines funktionsfähigen Systems geeignet.

Daher sind Methoden erforderlich, um die Kompatibilität von Implementierungen auf unterschiedlichen Plattformen – Software und Hardware – zu zeigen. Wesentlich ist dabei für Imple-

mentierungen, die direkt mit realen Systemkomponenten interagieren, daß die Kompatibilität zur Anwendungsumgebung und deren Komponenten gezeigt werden kann.

1.2 Bestehende Methoden

Schaltungsbeschreibungssprachen wie VHDL (und Verilog) erlauben die Realisierung komplexer Testumgebungen zur prozeduralen Erzeugung von Stimuli, aber auch zur Auswertung von Zwischenergebnissen während einer laufenden Simulation. Ein einfaches Beispiel dafür ist die Überwachung der Einhaltung von Zeitbedingungen. Dennoch funktionieren Prototypen trotz Validierung durch Simulation häufig nicht in einer realen Systemumgebung [Keating und Bricaud, 1998], da bei der Simulation bestimmte Annahmen gemacht werden und Simulationen sich meist nur über relativ kurze Realzeitintervalle erstrecken. Weisen bestehende Systemkomponenten ein definiertes Verhalten außerhalb bestimmter Spezifikationen auf, so bleibt dieses Verhalten beim Schaltungsentwurf mit großer Wahrscheinlichkeit unberücksichtigt und tritt – wenn überhaupt – erst bei umfangreichen Simulationen in Erscheinung, aber nur wenn Simulationsergebnisse in geeigneter Weise bewertet werden können. So erfordert die Simulation komplexer Systemkomponenten und die Bewertung umfangreicher Simulationsergebnisse besondere Methoden.

Zur Bewertung von Simulationsergebnissen stehen verschiedene Methoden zur Verfügung. Impulsdiagramme sind zur Fehlersuche und für die Validierung von Einzelkomponenten geeignet. Die graphische Darstellung von Signalverläufen und Spektren ist zur Bewertung von Simulationsergebnissen auch bis zu einem gewissen Grad geeignet. Für die Bewertung des Verhaltens einer Implementierung in einer realen Anwendungsumgebung sind diese Methoden jedoch unzureichend, insbesondere dann, wenn die Anwendungsumgebung reale Komponenten enthält, die sich einer vollständigen Spezifikation entziehen.

Mit formalen Verifikationsmethoden kann Äquivalenz zwischen Implementierungen oder verschiedenen Repräsentationen einer Implementierung nachgewiesen werden. Ob jedoch eine originäre Spezifikation das vom Entwerfer beabsichtigte Verhalten repräsentiert, läßt sich damit jedoch nicht zeigen. Die testweise Implementierung in Form eines Prototypen ist dafür zwar geeignet, setzt aber eine zumindest teilweise funktionsfähige Implementierung voraus.

1.3 Ansatz der Arbeit

Inspiriert durch die visomotorische Fahrzeugsteuerung mit einem neuronalen Netzwerk namens ALVINN [Pomerleau, 1989, 1993] werden im Rahmen der Arbeit Verfahren zur Validierung von Systemen zur Echtzeitverarbeitung von Videobildern mit neuronalen Netzwerken eingeführt. Im Rahmen einer Diplomarbeit [Krol, 1996] wurde ein neues Architekturkonzept [Larsson *et al.*, 1996, 1997a,b] für ein Backpropagation-Netzwerk [Rumelhart *et al.*, 1986a,b; Rumelhart und McClelland, 1986; McClelland und Rumelhart, 1986] zur Echtzeitklassifizierung von Bildern in einen VLSI-Schaltungsentwurf umgesetzt. An dieses Architekturkonzept wurden verschiedene Anforderungen gestellt, die sich aus technischen Randbedingungen des Chip-Entwurfs ergeben. Die Videosignalverarbeitung ist Teil des Architekturkonzepts. Denn erst durch Integration vi-

deosignalverarbeitender Einheiten kann der Flaschenhals zwischen Datentransfer und der Datenspeicherung vermieden werden.

Das System wurde rein digital konzipiert, da der Entwurf analoger Komponenten auf einem Chip aufwendiger ist, als der Entwurf digitaler Komponenten. Bei der Integration eines Systems auf einem einzigen Chip – System-on-a-Chip (SoC) – ist die Zahl der Anschlüsse (Pads) zur Außenwelt limitiert. Chip-Gehäuse mit vielen Pads sind teurer als solche mit weniger Pads. Außerdem ist die Handhabung von Chip-Gehäusen mit vielen Pads aufwendiger. Für eine effiziente Hardware-Implementierung ist der Übergang von der Gleitkommazahlendarstellung zur Festkommadarstellung erforderlich. Die Implementierung von Gleitkommarechenwerken erfordert mehr Ressourcen [Omondi, 1994].

Bei der Konzipierung und Implementierung von Komponenten des Systems zur Echtzeitverarbeitung von Videobildern wurde die Notwendigkeit von neuen Methoden zur Validierung deutlich. Im Rahmen der vorliegenden Arbeit wird eine neue Methode zur Visualisierung des Lernprozesses unter dem Namen *Lerntrajektorien* [Larsson, 1997, 1999a] eingeführt. Die Visualisierung durch Lerntrajektorien unterstützt die Parametrierung und die systemnahe Validierung von Backpropagation-Netzwerken. Über den Rahmen der Arbeit hinaus scheint diese Methode allgemein für die Visualisierung von hochdimensionalen Prozessen geeignet zu sein, z.B. für Parametrierung, Erforschung und Weiterentwicklung von Optimierungsalgorithmen.

Zum Architekturkonzept gehört auch eine digitale Vorverarbeitung von Videosignalen. Um deren Komponenten unter Berücksichtigung realer Systemumgebungen entwickeln zu können, bedarf es einer bidirektionalen Schnittstelle zwischen realen analogen Videosignalen und einem Computer-Simulationssystem. Eine solche Schnittstelle [Larsson, 1996] wurde später unter dem Namen *VidTrans* auf verschiedenen internationalen Fachmessen (*Hannover-Messe Industrie'97* und *Elektronica'98* in München) ausgestellt und auch bei verschiedenen regionalen Veranstaltungen (*Handelskammer Hamburg* im Juni'97 und bei der Auftaktveranstaltung der *Hamburger Mikroelektronik Initiative* der TU-TECH GmbH (TU-Hamburg-Harburg) im September'97) unter dem Namen *VidTrans* $\approx$ (*Vid*eosignal-*Trans*ienten-Rekorder) praktisch demonstriert.

1.4 Kapitelübersicht

Kapitel 2 : Grundlagen

Nach einer allgemein gehaltenen Einführung in das Gebiet der neuronalen Netzwerke erfolgt die mathematische Beschreibung des für die Arbeit zentralen Backpropagation-Netzwerks und Backpropagation-Lernalgorithmus. Bestehende Implementierungen werden beschrieben und offene Fragen werden aufgezeigt. Im folgenden Abschnitt sind die Verfahren der digitalen Signalverarbeitung zusammengetragen, die für die vorliegende Arbeit wichtig sind. Daran schließt ein Abschnitt zur Videosignalverarbeitung an. Der darauf folgende Abschnitt zum Entwurf digitaler Schaltungen führt in die abschließende Betrachtung über, in der offene Fragen aufgezeigt werden.

Kapitel 3 : Gewählter Ansatz

Dieses Kapitel bildet die Brücke zwischen den offenen Fragen und den im Rahmen der vorliegenden Arbeit neu eingeführten Verfahren. Der Ansatz der *systemnahen Validierung originärer Spezifikationen* wird skizziert und in die Technische Informatik eingeordnet. Anschließend wird der Rahmen und Hintergrund der Arbeit spezifiziert, und die Architektur des Rahmensystems wird kurz beschrieben.

Kapitel 4 : Entwickelte Verfahren

In diesem Kapitel werden die eigenen, im Rahmen der vorliegenden Arbeit entwickelten, Verfahren und Erweiterungen bestehender Verfahren beschrieben. Nach einer Einleitung in den Systemkontext erfolgt zunächst die detaillierte, mathematische Beschreibung der *Lerntrajektorien* mit daran anschließenden Anwendungsbeispielen, um die Semantik der *Lerntrajektorien* zu veranschaulichen. Dabei wird schrittweise der Bezug zum Backpropagation-Lernen und dem digitalen Schaltungsentwurf hergestellt und durch weitere Anwendungsbeispiele illustriert. Schlußfolgerungen und ein Ausblick schließen den Abschnitt zu *Lerntrajektorien* ab. Anschließend wird der Videosignal-Transienten-Rekorder *VidTrans* eingeführt, und anhand verschiedener Implementierungen digitaler, videosignalverarbeitender Komponenten werden Unterschiede zu bestehenden Visualisierungsmethoden aufgezeigt. Der Abschnitt zu *VidTrans* wird ebenfalls mit Schlußfolgerungen und einem kurzen Ausblick abgeschlossen.

Kapitel 5 : Implementierungen

Implementierungsdetails, die im vorangegangenen Kapitel ”Entwickelte Verfahren” von den wesentlichen wissenschaftlichen Inhalten ablenken könnten, werden in diesem Kapitel beschrieben.

Anhang

Verwendete Abkürzungen sind im Anhang ab Seite 161 aufgeführt. Außerdem befinden sich im Anhang einige verwendete Formeln und mathematische Umformungen, um gegebenenfalls ein Nachvollziehen von mathematischen Beschreibungen im Hauptteil der Arbeit zu erleichtern. Referenzen auf Gleichungen im laufenden Text haben die Form $(k.n)$, wobei k die Kapitelnummer der Gleichung und n die laufende Nummer der Gleichung im Kapitel k ist.

Kapitel 2

Grundlagen

Nach einer Einführung in das Gebiet der neuronalen Netzwerke im Abschnitt 2.1 erfolgt im Anschnitt 2.2 eine mathematische Beschreibung des für die vorliegende Arbeit zentralen *Backpropagation-Netzwerks* mit dem *Backpropagation-Algorithmus*. Daran anschließend werden bestehende Ansätze zur Implementierung beschrieben und offene Fragen aufgezeigt. Da die Vorverarbeitung von Information für den praktischen Einsatz neuronaler Netzwerke von großer Bedeutung ist, beinhaltet dieses Kapitel im Abschnitt 2.3 auch eine zusammenfassende Darstellung der im Rahmen dieser Arbeit implementierten Verfahren der *digitalen Signalverarbeitung*. Im Abschnitt 2.4 werden die Grundlagen der Videosignalverarbeitung skizziert, die für die vorliegende Arbeit von Bedeutung sind. Anschließend erfolgt in Abschnitt 2.5 eine kurze Beschreibung des Entwurfs digitaler, integrierter Schaltungen. Dabei wird auf die durch Schaltungsimplementierung von Algorithmen induzierten Randbedingungen und die dafür spezifischen Methoden zur Bewertung von Ergebnissen der Simulation komplexer Schaltungen eingegangen. Abschließend werden offene Fragen aufgezeigt. In diesem Kontext stellen sowohl die Rahmen der Arbeit eingeführten *Lerntrajektorien* [Larsson, 1997, 1999a] als auch die mit Hilfe von programmierbaren Logikbausteinen (EPLDs) realisierte Schaltung (*VidTrans*) zum Austausch von (analogen) Videosignalen mit Simulationen [Larsson, 1996] besondere Methoden zur Bewertung von Simulationsergebnissen dar.

2.1 Neuronale Netzwerke

Die Modellierung neuronaler Netzwerke erfolgt, abhängig vom wissenschaftlichen Kontext, in unterschiedlicher Weise und mit unterschiedlichen Zielen. Die Biologie und Neurophysiologie modellieren künstliche neuronale Netzwerke mit dem primären Ziel, biologische Zusammenhänge zu beschreiben. Die kognitive Psychologie versucht, kognitive Vorgänge zu modellieren, um diese zu verstehen und erklären zu können. Die Physik zeigt Analogien zu biologischen neuronalen Netzwerken auf und modelliert sie mit physikalischen Methoden. Die Philosophie verfolgt erkenntnistheoretische Ansätze. Ingenieurwissenschaften verwenden (künstliche) neuronale Netzwerke als Paradigmen für die Realisierung technischer Systeme, die mit konventionellen Methoden nur schwer zu handhaben sind. Für die Informatik sind neuronale Netzwer-

ke eine Erscheinungsform informationsverarbeitender Systeme mit bestimmten Eigenschaften [Hecht-Nielsen, 1990].

Künstliche neuronale Netze stellen, an biologischen Erkenntnissen angelehnte, mathematische Modelle biologischer Nervenzellen und Nervenzellenverbände dar (Abb. 2.1). Das bedeutet, daß empirisch gewonnene Erkenntnisse über mikrobiologische Mechanismen elektrochemischer Verarbeitung und Übertragung von Information zwischen Nervenzellen zu mathematischen Modellen künstlicher neuronaler Netzwerke geführt haben, die bezüglich ihres Verhaltens Analogien mit biologischen neuronalen Netzwerken aufweisen, obwohl derzeit nicht einmal alle mikrobiologischen Mechanismen vollständig verstanden sind. Das gilt insbesondere für höhere mentale Fähigkeiten des Menschen.

Die etablierten Netzwerkmodelle unterscheiden sich sehr stark dahingehend, bis zu welchem Grad sie biologische Vorbilder nachbilden und ob sie eher mathematischen oder technischen Randbedingungen angepaßt sind. Systeme künstlicher neuronaler Netze können daher auch als konnektivistische Systeme bezeichnet werden. Aus technischer Sicht stellen neuronale Netzwerke, bezogen auf eine bestimmte Anwendung, modellfreie, technische Systeme dar, für die eine Modellbildung nicht durch imperative Vorgabe eines Algorithmus erfolgt, sondern in Analogie mit biologischen Systemen, durch Erlernen eines gewünschten Verhaltens anhand von Beispielen. Das Lernen wird durch netzwerkspezifische Lernalgorithmen gesteuert, die mehr oder weniger physiologisch plausibel sind. Im Zusammenhang mit der Realisierung technischer Systeme dienen neuronale Netzwerke unter Umständen zunächst nur als vorläufiger Ansatz, etwa im Sinne von Machbarkeitsstudien, um technische Lösungsansätze zu validieren, oder zur Realisierung von Prototypen, um später gegebenenfalls ein System oder Teile eines Systems mit konventionellen, algorithmischen Verfahren zu implementieren.

Ein Grundproblem bei der Verwendung neuronaler Netzwerke besteht darin, daß Informationsverarbeitung *verteilt* in einer Weise erfolgt, die es nicht zuläßt, ein spezielles Ergebnis neuronaler Informationsverarbeitung lokal, etwa einer Folge von Befehlen, zuzuordnen. Die-

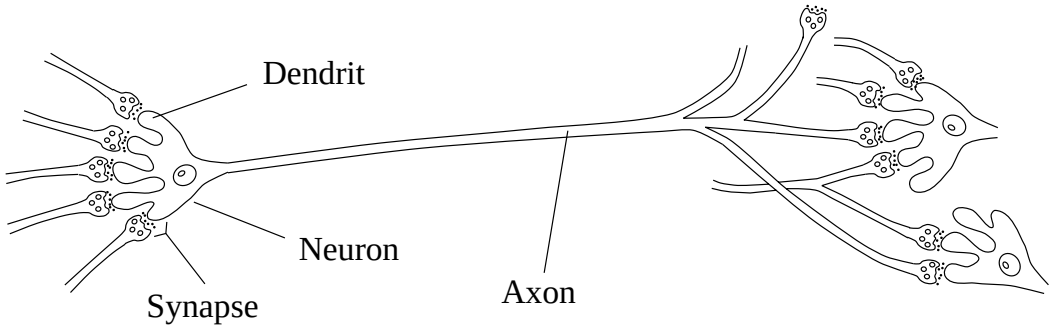


Abbildung 2.1: **Neuron, Axon, Dendriten, Synapsen** *biologischer neuronaler Netze (nach [McClelland und Rumelhart, 1986]). Diese physiologischen Einheiten sind als mathematisch modellierte funktionale Elemente in künstlichen neuronalen Netzwerken wiederzufinden (Abb. 2.4 auf S. 18).*

ser Umstand erschwert insbesondere den industriellen Einsatz neuronaler Netzwerke. Das Erlernen eines gewünschten Verhaltens, gesteuert durch den Backpropagation-Algorithmus, stellt aus mathematischer Sicht die numerische Lösung eines nicht-linearen Gleichungssystems dar. Das Verhalten eines Backpropagation-Netzwerks wird durch einen Satz von Parametern – die synaptischen Gewichte – bestimmt. Ein Satz von synaptischen Gewichten, der bezüglich eines Satzes von sogenannten Lernmustern gewünschtes Verhalten zeigt, stellt zusammen mit der durch ein Backpropagation-Netzwerk bestimmten Abbildungsvorschrift (s. Seite 15, Gleichungen (2.4), (2.5) und (2.3)) ein nicht-lineares Gleichungssystem dar. Der Backpropagation-Algorithmus versucht beim Lernen numerisch einen Satz synaptischer Gewichte zu finden, so daß eine vorgegebene Fehlerfunktion (2.6) bezüglich eines Satzes von M sogenannten *Lernmustern* minimal wird. Dazu werden die synaptischen Gewichte in Richtung des negativen Gradienten dieser Fehlerfunktion schrittweise verändert.

Dabei weist der Lernvorgang auch schon bei vergleichsweise einfachen Problemen einen hochdimensionalen Charakter auf. So erfordert schon das Erlernen der einfachen Boole'schen Funktion XOR: $y(x_1, x_2) = x_1 \oplus x_2 = \overline{x_1}x_2 \vee x_1\overline{x_2}$ ein Backpropagation-Netzwerk mit neun Parametern [Rumelhart und McClelland, 1986; Rojas, 1993]. Es sei vorgreifend an dieser Stelle bemerkt, daß gerade dem XOR-Problem im Zusammenhang mit neuronalen Netzwerken eine besondere historische Bedeutung zukommt.

2.1.1 Historischer Abriß

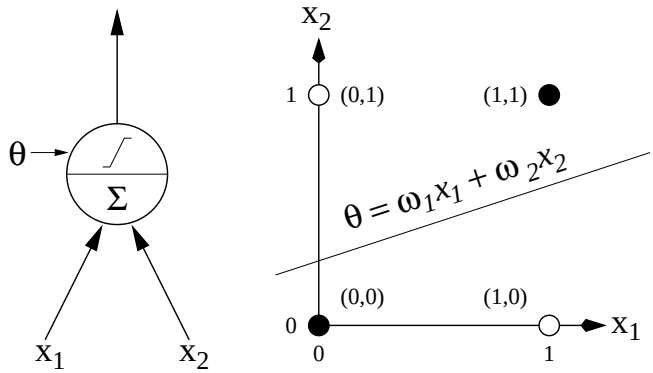
Im Laufe der Zeit wurden viele verschiedene Modelle künstlicher neuronaler Netzwerke entwickelt. Die Gemeinsamkeit der unterschiedlichen Modelle besteht im Aufbau aus relativ einfachen Rechenelementen, die untereinander vernetzt sind. Kurze historische Übersichten sind beispielsweise in [Hecht-Nielsen, 1990; Hertz *et al.*, 1991] zu finden. Viele der historisch wichtigen Originalarbeiten bezüglich der Entwicklung künstlicher neuronaler Netzwerke sind in der umfassenden Kollektion [Anderson und Rosenfeld, 1988] in chronologischer Reihenfolge zusammengetragen.

Gemeinhin wird die Publikation von Warren McCulloch und Walter Pitts aus dem Jahr 1943 als Grundlage der künstlichen neuronalen Netzwerke angesehen [McCulloch und Pitts, 1943], in der ein einfaches biologisch motiviertes Modell neuronaler Strukturen vorgestellt wird, das später nach ihnen benannte *McCulloch-Pitts-Neuron*. Im Jahr 1949 formulierte Donald O. Hebb in seinem Buch mit dem Titel *The Organization of Behavior* [Hebb, 1949] eine neurophysiologisch motivierte Hypothese bezüglich des Lernens, die *Hebb'sche Lernregel*. Dieser Algorithmus weist einen lokalen und daher physiologisch zumindest plausiblen Charakter auf. Das als *Perceptron* bezeichnete neuronale Netzwerk wurde 1958 von F. Rosenblatt vorgestellt [Rosenblatt, 1958]. Es stellt ein sogenanntes vorwärtsgerichtetes, schichtenorientiertes neuronales Netzwerk mit binären Schwellwertelementen, ohne eine verdeckte Neuronenschicht (*engl.* Hidden Layer) wie beim später eingeführten Backpropagation-Netzwerk, dar. Das Perceptron baut auf physiologischen Grundlagen auf. Wie Minsky und Papert später zeigten, kann es jedoch nur *linear teilbare* Muster (s. Abb 2.2) klassifizieren, und es versagt somit schon beim XOR-Problem [Minsky und Papert, 1969].

Ein neues Modell eines neuronalen Rechenelements namens *Adaline (ADAptive LINear Element)* wurde 1960 von Bernard Widrow und Marcian E. Hoff zusammen mit einer geeigneten Lernregel publiziert [Widrow und Hoff, 1960], die später in der Literatur als *Delta-Regel* oder als *Widrow-Hoff-Regel* bezeichnet wurde. Mit *Adaline* wurde neben einem *Fehlersignal* auch erstmals eine konstante Komponente *(Bias)* eingeführt, was für die geschlossene Formulierung des über zwei Jahrzehnte später eingeführten mehrschichtigen Backpropagation-Algorithmus, der eine verallgemeinerte Form der Widrow-Hoff-Regel darstellt, von Bedeutung ist (s. Abschnitt 2.2.1). Bernard Widrow gründete 1962 die erste Neurocomputerfirma (Memistor Corporation).

Auf die erste Phase der Euphorie folgte eine Phase der Ernüchterung, die im Jahr 1969 durch Minsky und Papert eingeleitet wurde, indem sie in ihrem Buch [Minsky und Papert, 1969] mathematisch mit einem Widerspruchsbeweis darlegten, daß perzeptronartige Netzwerke *ohne* verdeckte Schicht, selbst eine so einfache Funktion wie das XOR oder das Erkennen einfacher geometrischer Figuren grundsätzlich *nicht* lernen können. Das Interesse an neuronalen Netzwerken (Perceptrons) verlagerte sich statt dessen auf adaptive Signalverarbeitung, Mustererkennung und biologische Modellierung neuronaler Strukturen [Hecht-Nielsen, 1990]. Außerdem entstanden Arbeiten zu selbstorganisierenden Systemen. Daß die Modellierung neuronaler Netzwerke disziplinübergreifend ist, zeigen zwei unabhängig voneinander im Jahr 1972 veröffentlichte Arbeiten, die zu demselben Modell eines Assoziativspeichers gelangen [Anderson, 1972] und [Kohonen, 1972]. James A. Anderson entwickelte das Modell als Neurophysiologe und Teuvo Kohonen als Elektroingenieur. In den Jahren 1982 und 1984 veröffentlichte J. J. Hopfield erste Arbeiten, die eine Verbindung zwischen neuronalen Netzwerken und physikalischen Systemen herstellten [Hopfield, 1982, 1984] und großes wissenschaftliches Interesse weckten.

Mit der Veröffentlichung der beiden Werke *Parallel Distributed Processing, Volume I & II* [Rumelhart und McClelland, 1986; McClelland und Rumelhart, 1986] und einem Artikel in *Nature* [Rumelhart *et al.*, 1986c] im Jahr 1986 wurde wieder starkes Interesse an künstlichen neu-



x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

XOR	
(0,0)	●
(0,1)	○
(1,0)	○
(1,1)	●

Abbildung 2.2: **XOR-Problem** *Die XOR-Funktion ist nicht linear teilbar. Es ist nicht möglich, eine von den zwei Eingängen x_1 und x_2 aufgespannte Ebene durch eine einzige Gerade $\theta = \omega_1 x_1 + \omega_2 x_2$ so in zwei Halbebenen zu teilen, daß die Einspunkte (○) der XOR-Funktion in der einen Halbebene liegen und die Nullpunkte (●) der XOR-Funktion in der anderen.*

ronalen Netzwerken geweckt. Insbesondere nehmen die Autoren in den Publikationen direkten Bezug auf *Perceptrons* [Minsky und Papert, 1969] und präsentieren darin das *Backpropagation-Netzwerk* und den *Backpropagation-Algorithmus*, die es ermöglichen, das XOR zu repräsentieren und zu lernen. In der Literatur wird die Dissertation [Webros, 1974] als erste Arbeit angesehen, die grundsätzlich den Backpropagation-Algorithmus, wenn auch nicht unter diesem Namen, eingeführt hat. Als weitere, unabhängige Arbeiten werden in diesem Zusammenhang [Parker, 1985] und [Le Cun, 1985] angegeben. Im Jahr 1987 findet die erste *IEEE International Conference on Neural Networks* statt. Im Jahr 1989 erscheint die erste Ausgabe des Journals *Neural Computation* und 1990 erscheint die erste Ausgabe von *IEEE Transactions on Neural Networks*.

Das letztlich vor allem durch die Einführung des Backpropagation-Netzwerks erneut hervorgerufene, große wissenschaftliche Interesse an künstlichen neuronalen Netzwerken mag auch gerade der Grund für die große Popularität der Backpropagation-Netzwerke sein. So wurde das Backpropagation-Netz erfolgreich auf viele technische Probleme angewandt, wie etwa das Aussprechen geschriebener Worte mit NetTalk [Hecht-Nielsen, 1990; Hertz *et al.*, 1991; Rojas, 1993]. Das Grundkonzept von NetTalk basierte auf dem damals von der Firma *Digital Equipment Corporation* vorgestellten kommerziellen System zur Sprachsynthese namens DECTalk, welches nicht auf neuronalen Netzwerken basierte, sondern auf algorithmisch formulierten Regeln. NetTalk wurde mit Hilfe von DECTalk trainiert und lernte es nachzubilden. Am Ende lieferte NetTalk vergleichbare Ergebnisse wie DECTalk. Weitere Anwendungsbeispiele sind das Balancieren eines Stabes (inverses Pendel) [Hecht-Nielsen, 1990; Schöneburg *et al.*, 1990] und die visomotorische Fahrzeugsteuerung durch ALVINN [Pomerleau, 1989, 1993]. Ein kommerzielles System zur Implementierung neuronaler Algorithmen namens NNetView [Maris und Fürth, 1997] erlaubt die Verarbeitung von Bildern in ähnlich direkter Weise wie bei ALVINN. Weitere Anwendungen sind in [Schöneburg, 1993; Chauvin und Rumelhart, 1995; Patterson, 1996; Taylor, 1996] beschrieben. Seitdem wurden Varianten des Backpropagation-Netzwerks und Backpropagation-Algorithmus entwickelt und mit Methoden der *Fuzzy-Logik* kombiniert [Lin und Lee, 1996], um neue Anwendungen zu erschließen oder effizientere Implementierungen zu ermöglichen. Zur Unterstützung solcher Entwicklungen gibt es Ansätze zur Visualisierung des hochdimensionalen Lernvorgangs [Pratt und Nicodemus, 1994; Rojas, 1994], die über die graphische Darstellung des Gesamtfehlers während des Lernvorgangs hinausgehen. Die im Rahmen der vorliegenden Arbeit entwickelte Methode der *Lerntrajektorien* ordnet sich in diese Entwicklungen ein.

2.1.2 Modelle neuronaler Netzwerke

Eine Taxonomie neuronaler Netzwerke kann nach ganz unterschiedlichen Netzwerkparametern erfolgen. Jede spiegelt eine andere Perspektive neuronaler Netzwerke wider. Eine Übersicht über einige alternative Taxonomien neuronaler Netzwerke ist in [Patterson, 1996] zu finden. Als übergeordnete Bezeichnungen dienen dabei: *Lernstrategie* (supervised, reinforcement, unsupervised), *Lernregel* (fehlerkorrigierend, Hebb'sch, wettbewerbend, stochastisch), *Anwendungstyp* (Assoziativspeicher, Optimierung, Klassifizierung, Mustererkennung, Abbildung allgemein, Vorhersage), *Architektur* (einlagig, mehrlagig, rückgekoppelt). So könnte beispielsweise auch der Bezug zu physiologischen Modellen, zu physikalischen Modellen oder zu technischen Randbedingungen als Kriterium dienen.

Eine detaillierte Beschreibung erfolgt in diesem Kapitel nur für das *Backpropagation-Netz*, da dieses für die vorliegende Arbeit eine zentrale Rolle spielt. Andere wichtige Modelle neuronaler Netzwerke werden in der Literatur [Hecht-Nielsen, 1990; Hertz *et al.*, 1991; Rojas, 1993; Patterson, 1996; Lin und Lee, 1996] detailliert beschrieben und diskutiert.

Technisches Paradigma

Die Entwicklung von mathematischen Modellen neuronaler Netzwerke war ursprünglich eher physiologisch motiviert, auch wenn dabei nicht nur das Verständnis biologischer neuronaler Netzwerke als Motivation diente, sondern auch der Wunsch, durch Nachbildung biologischer Systeme neue Methoden für technische Anwendungen zu erarbeiten.

Anwendungsbereiche

Neuronale Netzwerke kommen in unterschiedlichen Anwendungsbereichen zum Einsatz. Dies sind im wesentlichen: assoziative Speicherung, Optimierung, Klassifizierung, Mustererkennung und Vorhersage [Patterson, 1996]. Neuronale Netzwerke können immer dann in einem technischen Kontext eingesetzt werden, wenn es gelingt, eine vorliegende Aufgabenstellung in einer Weise zu formulieren, in der ein spezielles neuronales Netzwerk direkt als eine Komponente – in Form eines Moduls mit definierten Eingängen und Ausgängen – in Erscheinung tritt. Sind die Schnittstellen eines neuronalen Netzwerks zu einer Systemumgebung definiert, so bestehen meist noch hinsichtlich verschiedener Netzwerkparameter viele Freiheitsgrade. Soll ein neuronales Netzwerk ein bestimmtes Verhalten anhand von Beispielen erlernen, so ist die Auswahl von geeigneten Lernbeispielen sowie deren Umfang für eine erfolgreiche Anwendung von großer Bedeutung. Außerdem spielt die Vorverarbeitung und Repräsentation der Information, die durch ein neuronales Netzwerk weiterverarbeitet werden sollen, eine wesentliche Rolle. Je besser die Information zur Weiterverarbeitung durch ein neuronales Netzwerk aufbereitet wird, um so größer sind die Erfolgsaussichten, eine bestimmte Aufgabe mit einem neuronalen Netzwerk in gewünschter Weise zu lösen. Die gewählte Codierung von Information bestimmt beim Lernen mit dem Backpropagation-Algorithmus den Schwierigkeitsgrad, beim Lernen eine Lösung zu finden, weil die Codierung Einfluß auf den Suchraum beim Gradientenabstieg hat, was in [Bakker *et al.*, 1993] kompakt und deutlich dargestellt wird. Die Fähigkeit zur Generalisierung ist eine wichtige Eigenschaft neuronaler Netzwerke [Hertz *et al.*, 1991]. Generalisierung bedeutet, auf ähnliche Eingaben mit *ähnlichen* Ausgaben zu reagieren. Handelt es sich bei Eingaben und Ausgaben etwa um Bitvektoren, so würde mit dem Attribut *ähnlich* beispielsweise *geringer Hamming-Abstand* umschrieben werden, für Vektoren des $\mathbb{R}^n$ würde das Attribut *ähnlich* beispielsweise geringem *euklidischen Abstand* zugeordnet werden.

Anwendungsbeispiele für Backpropagation-Netze

Wie schon im Abschnitt 2.1.1 erwähnt, verdankt das Backpropagation-Netzwerk möglicherweise dem Zeitpunkt seiner Einführung seine große Popularität und damit verbunden die – verglichen mit anderen Netzwerktypen – relativ große Zahl von Anwendungen.

Zwei bekannte Anwendungsbeispiele für Backpropagation-Netzwerke wurden schon im historischen Abriß (Abschnitt 2.1.1) erwähnt: NETtalk und das inverse Pendel. Bei beiden Beispielen erlernt ein geeignetes Backpropagation-Netzwerk, das Verhalten eines Systems anhand von Beispielen nachzubilden. Zu beiden neuronalen Lösungen gibt es algorithmische Lösungen, in Form von Regeln wie bei DECTalk oder von Differentialgleichungen für das inverse Pendel. Ein weiteres praktisches, nicht triviales Beispiel ist das Rückwärtseinparken eines Lastwagens mit einem Anhänger [Nguyen und Widrow, 1989]. Ein Vergleich mit einer auf Fuzzy-Logik basierenden Lösung ist in einem eigenen Kapitel in [Kosko, 1992a] beschrieben. Der rückwärts einparkende Lastwagen mit Anhänger ist ein Beispiel für ein System, das sich nicht oder nur unzureichend, mit den bekannten analytischen, naturwissenschaftlichen Methoden, durch das Aufstellen von exakten Regeln und Gleichungen in algorithmischer Form beschreiben läßt. Oft ist gerade dann die Modellierung mit einem neuronalen Netzwerk hilfreich, um durch eine praktische Machbarkeitsstudie Erkenntnisse bezüglich der *grundsätzlichen* Machbarkeit eines Vorhabens zu gewinnen. Hat ein neuronales Netzwerk die Nachbildung des Verhaltens eines komplexen Systems gelernt, so ist es unter Umständen möglich, ein derart trainiertes neuronales Netzwerk durch Hardware-Implementierung für eine Echtzeitanwendung zu verwenden. Ein Beispiel dafür ist die Realisierung eines neuronales Netzwerks zur Steuerung aktiver Wirbelreduzierung turbulenter Strömungen auf Oberflächen [Babcock *et al.*, 1996], um etwa den Treibstoffverbrauch von Flugzeugen reduzieren zu können. In diesem Zusammenhang wurde ein neuronales Netzwerk trainiert, analytisch bestimmte Gesetzmäßigkeiten nachzubilden. Die Arbeit wurde biologisch von der Mikrostruktur der Haut des Haies inspiriert, die mit Methoden der Mikrosystemtechnik nachgebildet wurde.

Ein weiterer Anwendungsbereich sind Überwachungssysteme. Dabei erlernt ein neuronales Netzwerk das Verhalten eines Systems oder eines Teils davon, um anschließend dieses System teilweise emulieren zu können. Beispiele hierfür die Verschleißüberwachung von Hubschrauberrotorblättern [Vella *et al.*, 1996] oder ein Chip [Tryba, 1996] zur Überwachung bestimmter Parameter, die Aufschluß über Wasserverschmutzung geben, so daß nur bei Bedarf umfangreiche (zeitaufwendige und teure) Wasseranalysen eingeleitet werden müssen. Ein weiterer Anwendungsbereich neuronaler Netzwerke ist die Vorhersage des Verhaltens analytisch nur unvollständig beschreibbarer Systeme. Beispiele dafür sind die Vorhersage von Kursen im Finanzgeschäft [Pasquale *et al.*, 1996] oder die Bedarfsvorhersage von Wasser, Energie oder Verkehrsaufkommen [Ding *et al.*, 1996]. Wesentlich bei solchen Vorhersagen ist die Annahme, daß es Abhängigkeiten zwischen meßbaren Größen und dem Systemverhalten gibt, die grundsätzlich auch analytisch beschreibbar sind. Ein geeignetes neuronales Netz kann anhand von passenden Beispielen lernen, das Verhalten eines solchen Systems nachzubilden.

2.2 Backpropagation-Algorithmus

Der Backpropagation-Algorithmus (BPG) zum Trainieren von mehrschichtigen, rückkopplungsfreien neuronalen Netzwerken wurde im Jahr 1986 von D. E. Rumelhart, J. L. McCelland, G. E. Hinton, und R. J. Williams [Rumelhart *et al.*, 1986a], [Rumelhart *et al.*, 1986b] vorgestellt. Die biologischen und mathematischen Grundlagen sind in [Rumelhart und McClelland, 1986] und

[McClelland und Rumelhart, 1986] eingehend beschrieben. Beim Backpropagation-Algorithmus handelt es sich grundsätzlich um eine Abwandlung des Gradientenverfahrens zur Lösung nichtlinearer Gleichungssysteme. Eine skalare Schreibweise zur Darstellung des Backpropagation-Algorithmus ist für die Implementierung des Algorithmus auf sequentiellen Architekturen gut geeignet. Für die Implementierung des Backpropagation-Algorithmus auf (hoch-)parallelen Architekturen bietet sich jedoch eher die Matrixschreibweise an, die eine sehr kompakte Darstellung des Algorithmus erlaubt. Die Autoren von [Rumelhart und McClelland, 1986] führen beide Schreibweisen ein. Im folgenden Abschnitt erfolgt zunächst einmal eine kurze Beschreibung des Backpropagation-Algorithmus.

2.2.1 Neuron

Biologische Neuronen erzeugen elektrische Impulse, die auf elektrochemische Weise zu anderen Neuronen eines Netzwerks übertragen und verarbeitet werden [McClelland und Rumelhart, 1986]. Die Amplitude der Ausgangssignale von Neuronen ist nahezu konstant, die Information ist in der Frequenz der Signale codiert. Die Synapsen (Abb. 2.1) bestimmen die Kopplungsstärken (synaptische Gewichte) der Eingangssignale von Neuronen. Funktional wird zwischen erregenden (exzitatorischen) und hemmenden (inhibitatorischen) Synapsen unterschieden. Zur mathematischen Modellierung werden die Neuronensignale (Aktivitäten) und die synaptischen Gewichte zunächst als reelle Zahlen innerhalb endlicher Intervalle definiert.

Jedes Neuron i bzw. j berechnet hierbei die gewichtete Summe seiner Eingänge h_i bzw. x_j mit anschließender Anwendung einer nicht-linearen *Aktivierungsfunktion* (2.3). Die Ausgabe y_i des Neurons i mit der Ausgabe h_j des Hidden-Neurons j ist gegeben durch

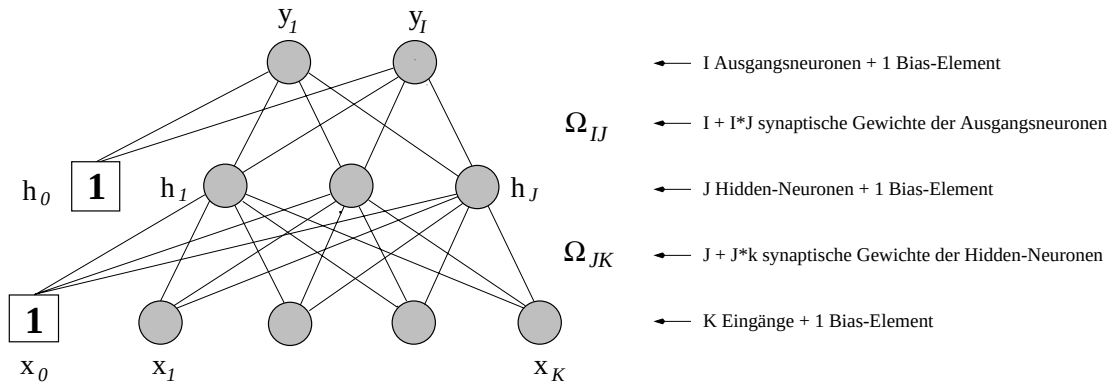


Abbildung 2.3: **Backpropagation-Netzwerk mit einer versteckten Schicht (Hidden-Layer).** *Es stellt ein [4-3-2] Netzwerk dar (K=4 Eingänge, J=3 Hidden-Neuronen, I=2 Ausgangs-Neuronen). Die synaptischen Gewichte des Netzwerks sind durch Kopplungsmatrizen Ω_{IJ} und Ω_{JK} spezifiziert.*

$$y_i = g(\theta_i + \sum_{j=1}^{J} \omega_{ij} \cdot h_j) \qquad \text{mit} \qquad h_i = g(\theta_j + \sum_{k=1}^{K} \omega_{jk} \cdot x_k). \tag{2.1}$$

Dabei sind ω_{ij} die *synaptischen Gewichte* zwischen den I Ausgangsneuronen und den J Hidden-Neuronen h_j. Die ω_{jk} sind dabei die *synaptischen Gewichte* zwischen den J Hidden-Neuronen h_j und den K Eingängen x_k. Die spezifische Schwelle θ_i eines jeden Neurons i bzw. Hidden-Neurons j kann durch zusätzliche Eingänge $x_0 := 1$ bzw. $h_0 := 1$ mit konstanter Aktivität modelliert werden. Damit erhält man zusätzliche Gewichte $\omega_{i0} = \theta_i$ bzw. $\omega_{j0} = \theta_j$, welche als *Bias* bezeichnet werden [Rumelhart *et al.*, 1986a] und grundsätzlich auf [Widrow und Hoff, 1960] zurückgehen. Damit wird Gleichung (2.1) zu

$$y_i = g(\sum_{j=0}^{j} \omega_{ij} \cdot h_j) \quad \text{mit} \quad h_0 = 1 \qquad \text{und} \qquad h_j = g(\sum_{k=0}^{K} \omega_{jk} \cdot x_k) \quad \text{mit} \quad x_0 = 1. \tag{2.2}$$

Die Einführung des *Bias* erlaubt eine vereinfachte und kompakte Formulierung des Backpropagation-Algorithmus, da keine zusätzliche Lernregel für die Schwellen θ_i und θ_j zu formulieren ist – die Schwellen können durch die Einführung des *Bias* wie synaptische Gewichte behandelt werden. In der folgenden Beschreibung des Backpropagation-Algorithmus wird daher auf die Schwelle θ verzichtet. Die Funktion

$$g(a) = \frac{1}{1 + e^{-a/\tau}} \tag{2.3}$$

in (2.2) stellt die biologisch motivierte, sigmoide Aktivierungsfunktion dar. Außer der Aktivierungsfunktion (2.3) werden auch andere Funktionen verwendet, wie etwa $g(a) = \tanh(a)$. Der Parameter τ bestimmt das Steigungsverhalten dieser sigmoiden Aktivierungsfunktion (2.3). Ein typischer Bereich für diese auch als *Temperaturparameter* bezeichnete Größe ist $0.1 < \tau < 10$. Im biologischen Kontext ist τ ein temperaturabhängiger Parameter. Die sigmoide Aktivierungsfunktion kann aus Messungen an biologischen Nervenzellen gewonnen werden [Netter, 1987].

2.2.2 Netzwerk

Ein Backpropagation-Netzwerk (wie in Abbildung 2.3) mit einer versteckten Schicht (Hidden-Layer) mit K Eingängen, J versteckten Neuronen und I Ausgangsneuronen ([K-J-I] Netzwerk) bildet einen Eingangsvektor $\vec{x}$ auf einen Ausgangsvektor $\vec{y}$ ab

$$\vec{y} = F(\Omega_{IJ}, \Omega_{JK}, \vec{x}), \tag{2.4}$$

wobei Ω_{IJ} und Ω_{JK} die synaptischen Kopplungsmatrizen darstellen und das Abbildungsverhalten des Backpropagation-Netzwerks bestimmen. Ein Netzwerk mit einer derartigen Schichtenstruktur ist grundsätzlich in der Lage, jede beliebige stetige Funktion $\vec{y} = F(\vec{x})$ von K Variablen x_k zu repäsentieren [Hecht-Nielsen, 1990; Rojas, 1993]. Die synaptischen Gewichte werden durch *Lernen* bestimmt. Für ein Netz mit einer verdeckten Schicht ist die Abbildungsvorschrift (2.4) in folgender Weise gegeben

$$\begin{aligned} \vec{h} &= \Omega_{JK} \cdot \vec{x}, \quad \vec{\gamma} = g(\vec{h}), \\ \vec{o} &= \Omega_{IJ} \cdot \vec{\gamma}, \quad \vec{y} = g(\vec{o}). \end{aligned} \tag{2.5}$$

Dabei sind $\vec{x}$, $\vec{h}$, $\vec{\gamma}$, $\vec{o}$ und $\vec{y}$ Spaltenvektoren.

2.2.3 Lernalgorithmus

Das Trainieren eines Backpropagation-Netzwerks bedeutet die Modifikation der Matrixelemente ω_{ij} der Matrix Ω_{IJ} und ω_{jk} der Matrix Ω_{JK}. Vor Beginn des Trainings werden die synaptischen Gewichte mit zufälligen Werten (typisch $\omega_{init} \in \{-0,5 \ldots +0,5\}$) vorbesetzt. Ziel des Trainings ist die Minimierung des Fehlers

$$E = \frac{1}{2} \sum_M (\vec{y} - \vec{t})_M^2 , \tag{2.6}$$

wobei M der Index über die Lernmuster darstellt. Ein Lernmuster $(\vec{x};\vec{t})_M$ ist dabei ein Paar aus einem Eingangsvektor $\vec{x}$, welcher mit einem Ausgangsvektor $\vec{t}$ (*target*) korreliert ist. Das Backpropagation-Netzwerk erlernt durch das Training die Korrelation solcher Vektorpaare. Während des Trainings wird Lernmuster (M) für Lernmuster ($M = 1, \ldots, M_{max}$) trainiert. Dazu wird ein Eingangsvektor $\vec{x}_M$ gemäß Gleichung (2.4) durch das Netzwerk auf einen Ausgangsvektor $\vec{y}$ abgebildet (Vorwärtspfad). Ausgehend vom Ausgangsvektor $\vec{y}$ und dem Zielvektor $\vec{t}_M$ des Lernmusters $(\vec{x};\vec{t})_M$ wird ein Fehler $\vec{d}$ berechnet, welcher dann rückwärts durch das Netz propagiert wird (Rückwärtspfad).

$$\begin{aligned} \vec{d} &= \vec{y} - \vec{t}, & \vec{\delta} &= g'(\vec{o}) \\ \vec{e} &=^t \Omega_{IJ} \cdot (\vec{d} \odot \vec{\delta}), & \vec{\epsilon} &= g'(\vec{e}), \end{aligned} \tag{2.7}$$

wobei ${}^t\Omega_{IJ}$ die transponierte Matrix der Matrix Ω_{IJ} ist. Der Operator $\odot$ steht hier für ein komponentenweises Vektor-Vektor-Produkt $\vec{c}$ zweier Vektoren $\vec{a}$ und $\vec{b}$ mit Komponenten $(a_1, \ldots, a_n)$ und $(b_1, \ldots, b_n)$, wobei

$$\vec{c} = \vec{a} \odot \vec{b} \Leftrightarrow c_i = a_i \cdot b_i \quad \text{für} \quad i = 1, \ldots, n \tag{2.8}$$

ist. Die Ableitung (2.9) der Sigmoidfunktion kann selbst wieder durch die Sigmoidfunktion (2.3) ausgedrückt werden (s. Anhang, S. 163)

$$g'(a) = \frac{1}{\tau} \cdot g(a) \cdot g(-a). \tag{2.9}$$

Mit den im Vorwärtspfad und Rückwärtspfad eines Lernschritts berechneten Vektoren werden die Matrizen der synaptischen Gewichte in folgender Weise modifiziert

$$\begin{aligned} \Omega_{IJ} &:= \Omega_{IJ} - \eta \cdot \vec{\delta} \cdot^t \vec{\gamma} \\ \Omega_{JK} &:= \Omega_{JK} - \eta \cdot \vec{\epsilon} \cdot^t \vec{x}. \end{aligned} \tag{2.10}$$

Dabei stellt η die *Lernrate* dar (typischer Wertebereich ist $0.001 < \eta < 0.5$). Die Modifikation der synaptischen Gewichte erfolgt in Richtung des negativen Gradienten des Fehlers (2.6)

$$\Delta\omega_{ij} = -\eta \cdot \frac{\partial E}{\partial \omega_{ij}} = -\eta \cdot \delta_i \cdot h_j \quad \text{und} \quad \Delta\omega_{jk} = -\eta \cdot \frac{\partial E}{\partial \omega_{jk}} = -\eta \cdot \epsilon_j \cdot x_k. \tag{2.11}$$

Um Oszillationen zu unterdrücken und die Konvergenz zu beschleunigen, dient der *Impulsterm* in (2.12) mit dem Impulsparameter α (typischer Wertebereich $0.1 < \alpha < 0.9$), um den die Gleichung (2.11) erweitert werden kann. Damit ergibt sich im Lernschritt n für die Änderung der synaptischen Gewichte

$$\Delta\Omega_n := \Delta\Omega + \underbrace{\alpha \cdot \Delta\Omega_{n-1}}_{\text{Impulsterm}}, \tag{2.12}$$

wobei die Matrixelemente $\Delta\Omega_{n-1}$ die Änderungen der synaptischen Gewichte Ω beim vorangegangenen Lernschritt $(n-1)$ und $\Delta\Omega$ die durch Fehlerrückpropagieren im aktuellen Lernschritt (n) berechnete und durchzuführende Änderung darstellen. In der vorliegenden Arbeit entspricht *ein* Lernschritt der Abbildung (2.4) mitsamt anschließender Veränderung der synaptischen Gewichtsmatrizen Ω_{IJ} und Ω_{JK} gemäß (2.11) in Abhänigkeit vom berechneten Fehler für *ein* Lernmuster $(\vec{x};\vec{t})_M$. Als Lernzyklus wird ein Satz von M Lernschritten bezeichnet.

Performanzangaben (cps & cups)

Die Performanz einer Implementierung des Backpropagation-Algorithmus wird üblicherweise in Einheiten $cps \approx$ *connections per second* für die Vorwärtsphase und in $cups \approx$ *connection updates per second* für die Lernphase angegeben. Ein Backpropagation-Netzwerk mit K Eingängen, einer verdeckten Neuronen Schicht mit J Hidden-Neuronen und I Ausgängen hat insgesamt

$$N = \underbrace{I \cdot (J+1)}_{\text{Zahl der synaptischen Gewichte } \Omega_{IJ}} + \underbrace{J \cdot (K+1)}_{\text{Zahl der synaptischen Gewichte } \Omega_{JK}} \tag{2.13}$$

synaptische Gewichte. Die additive Konstante $+1$ resultiert vom Bias. Um die Abbildung eines Eingangsvektors $\vec{x}$ auf einen Ausgangsvektor $\vec{y}$ gemäß (2.4) zu berechnen, sind daher N Multiplikation-Akkumulation-Operationen ($\sum \omega \cdot x$) gemäß (2.2) zu berechnen. Die Zahl derartiger Operationen pro Sekunde wird in cps angegeben. Die Zahl der Gewichtsänderungen beim Lernen, die pro Sekunde erfolgen, wird in $cups$ angegeben. Da Backpropagation-Lernen das Vorwärtspropagieren gemäß (2.5) und das Rückwärtspropagieren gemäß (2.7) – gegebenenfalls zusammen mit der Handhabung des Impulsterms (2.12) – erfordert sowie Speicherzugriffe für die eigentliche Änderung der synaptischen Gewichte, liegt der Wert für die $cups$ im allgemeinen um einen Faktor 2-3 unter dem Wert der cps.

2.2.4 Hardware-Implementierung

Die wesentlichen Vorteile einer Schaltungsimplementierung neuronaler Netzwerke sind hohe Verarbeitungsgeschwindigkeiten und die sich daraus ergebende Möglichkeit, größere Netze in kürzerer Zeit zu simulieren, wodurch die Untersuchung unterschiedlicher Netzwerkkonfigurationen erleichtert wird. Verglichen mit der Implementierung auf sequentiellen Rechnern kann die VLSI-Schaltungsimplementierung neuronaler Netzwerke eine Steigerung der Verarbeitungsgeschwindigkeit um Faktoren 100-1000 mit digitalen Schaltungen sowie Faktoren von 10000 und mehr mit analogen Schaltungen erreichen [Glesner und Pöchmüller, 1994].

Hohe Verarbeitungsgeschwindigkeit ist für Echtzeitanwendungen von Bedeutung, und die Realisierung größerer Netzwerke ist für bestimmte Aufgaben wie etwa Bildverarbeitung von Interesse (s. Abb. 2.5). Die unterschiedlichen Netzwerk-Algorithmen implizieren spezifische Randbedingungen, die bei der Software-Implementierung des Algorithmus zu berücksichtigen sind. Der VLSI-Entwurf impliziert dagegen andere Randbedingungen.

Bei der Schaltungsimplementierung neuronaler Netzwerke muß daher ein Kompromiß zwischen den algorithmischen und den VLSI-bestimmten Randbedingungen gefunden werden. Die-

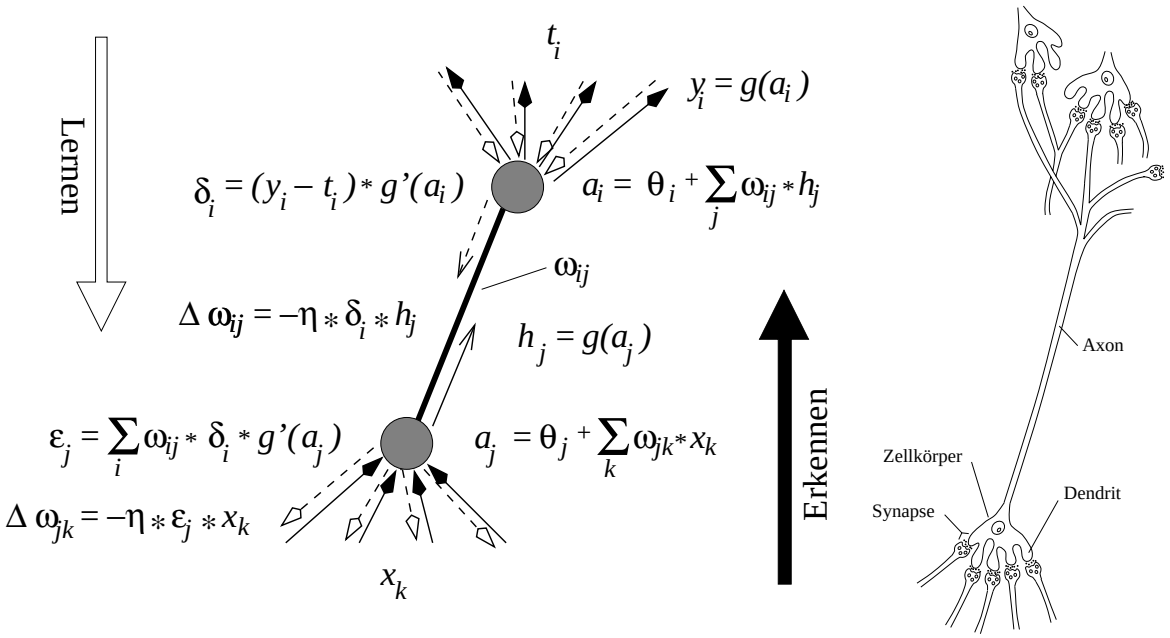


Abbildung 2.4: **Operationen des Backpropagation-Algorithmus.** *Die Operationen der Vorwärtsrichtung beim Erkennen von Mustern sind bezüglich der Informationsverarbeitung und aufgrund ihres lokalen Charakters und der Informationsflußrichtung physiologisch plausibel. Dem gegenüber ist das Backpropagation-Lernen ist dieser Weise physiologisch nicht plausibel. Die elementaren Operationen des Algorithmus sind für eine Hardware-Implementierung durch Schaltkreise nachzubilden.*

ser Kompromiß wird im allgemeinen durch geeignete Modifikation der betreffenden Algorithmen in eine VLSI-gerechte Form erreicht. Neuronale Netzwerke lassen sich in der Regel mit einfachen arithmetischen Operationen realisieren, die teilweise hochgradig parallel ausgeführt werden können. Um große Netzwerke untersuchen und einsetzen zu können, bedarf es zusätzlich der Möglichkeit, große Netzwerke quasi interaktiv bezüglich der Eignung für bestimmte Aufgaben untersuchen zu können. Dazu sind Lernzeiten von maximal einigen Stunden bis zu Tagen tolerabel. Die benötigte Netzwerkgröße wie auch die Verarbeitungsgeschwindigkeit ist anwendungsabhängig (vgl. Abb. 2.5).

Numerische Näherung

In Hinblick auf Hardware-Implementierung ist die Auswirkung von numerischen Näherungen auf das Verhalten des Backpropagation-Algorithmus entscheidend [Stevenson *et al.*, 1990]. Auf Rechnern, die über einen Prozessor mit integrierter *Floating Point Unit* (FPU) verfügen (Sparc, UltraSparc, 5x86), werden die Multiplikation-Akkumulation-Operationen des Backpropagation-Algorithmus bei Gleitkommadarstellung ebenso schnell (oder sogar schneller) ausgeführt wie

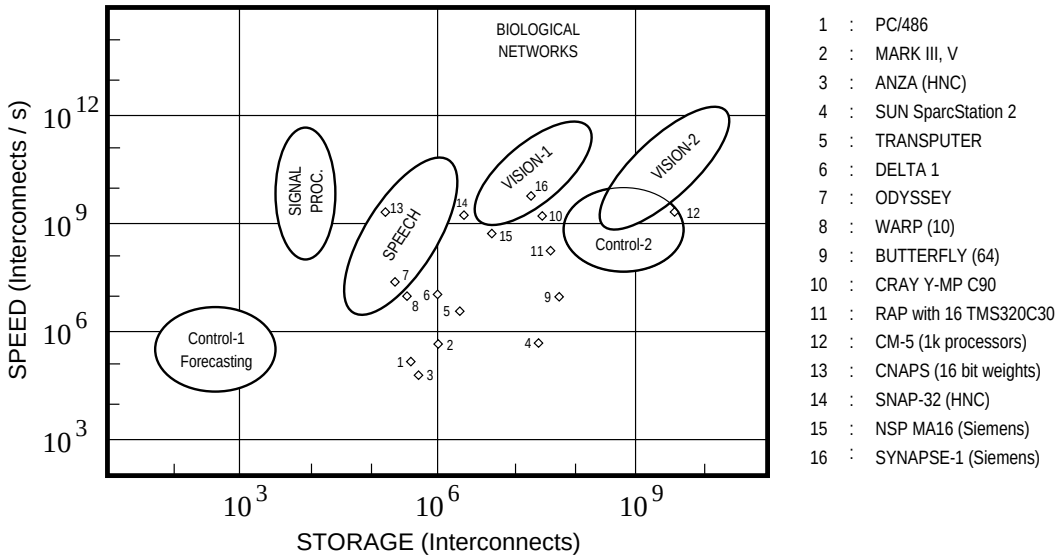


Abbildung 2.5: **Leistungsbedarf und Anwendungsfelder neuronaler Netzwerke,** *nach [Ramacher* et al.*, 1994]. Auf der X-Achse ist die, für bestimmte Klassen von Aufgaben benötigte, Zahl synaptischer Gewichte (INTERCONNECTS) aufgetragen. Auf der Y-Achse ist die erforderliche Verarbeitungsgeschwindigkeit (SPEED) aufgetragen. Die verschiedenen Bereiche weisen stark unterschiedliche Leistungsanforderungen auf, die teilweise nur durch sehr leistungsfähige Computer oder erst durch Hardware-Implementierung erreichbar sind.*

bei ganzzahliger Darstellung. Da die Gleitkommadarstellung darüber hinaus eine bessere Darstellungsdynamik als ganzzahlige Darstellungen aufweist [Hennessy und Patterson, 1996], ist die Gleitkommadarstellung bei Implementierung auf Rechnern mit FPU gegenüber ganzzahligen Darstellungen vorzuziehen.

Die Implementierung von Gleitkommarechenwerken ist in der Regel gegenüber Ganzzahlrechenwerken aufwendiger, da bei halblogarithmischer Zahlendarstellung Mantisse *und* Exponent gesondert verarbeitet werden müssen [Omondi, 1994; Hennessy und Patterson, 1996]. Gleitkommaarithmetik kann jedoch dann günstiger sein als Festkommaarithmetik, wenn beispielsweise sehr kleine und sehr große Beträge zu verarbeiten sind und Verarbeitung mit sehr geringen Datenwortbreiten für Mantisse und Exponent brauchbare Ergebnisse liefert und eine Festkommazahlendarstellung sehr viele Stellen erfordert [Cloutier und Simrad, 1994; Wüst *et al.*, 1998]. Das ist jedoch im Einzelfall zu untersuchen. Meist ist jedoch für eine Hardware-Implementierung von Algorithmen ganzzahlige Arithmetik vorzuziehen. Näherungen erfordern unter Umständen gravierende Modifikationen an den zu implementierenden Algorithmen. Sind Näherungen nicht oder nur in geringem Umfang möglich, so ist die Software-Implementierung auf einem Standardrechner einer speziellen Hardware-Implementierung vorzuziehen, da eine anwendungsspezifische Implementierung von Gleitkommarechenwerken zu aufwendig ist.

Beim Backpropagation-Algorithmus sind die Anforderungen an die numerische Genauigkeit in der Lernphase höher als in der Vorwärtsphase. In der einschlägigen Literatur [Glesner und Pöchmüller, 1994] werden Werte von 8 Bit- bis zu 32 Bit-Floating-Point für den Lernalgorithmus angegeben. Die tatsächlich benötigte numerische Auflösung ist problemabhängig und kann zudem noch in extremer Weise von der Codierung der zu lernenden Muster abhängen, da sie Einfluß auf die die Fehlerfunktion (2.6) hat und damit auf den Gradientenabstieg, wie in [Bakker *et al.*, 1993] anhand eines N-2-N-Encoders mit einem [n-2-n]-Netzwerk durch besondere Mustercodierung gezeigt wird. Das Encoder-Problem (N-M-N-Encoder mit $M = \log_2(N)$) ist ein übliches Testproblem für den Leistungsvergleich von Implementierungen des Backpropagation-Algorithmus [Hertz *et al.*, 1991; Patterson, 1996]. Die Lernfreudigkeit in Abhängigkeit von gewählter Mustercodierung (Binärcode vs. Thermometercode) wird in [Gallant, 1993] diskutiert. In der Literatur finden sich daher verschiedene Arbeiten, die die Auswirkungen unterschiedlicher numerischer Darstellungen auf das Backpropagation-Lernen untersuchen. So wird in [Debenham und Grath, 1989] für die Abrufphase 8 Bit und für die Lernphase 16 Bit benötigte Genauigkeit angegeben. In [Hollis *et al.*, 1990] wird eine Mindestauflösung beim Lernen von 12 Bit angegeben. Zur Repräsentation der Parität geben die Autoren in [Hoehfeld und Fahlman, 1992] zum Lernen 12 Bit und zum Erkennen 7 Bit an. Bei analogen Implementierungen ist die gegenüber digitalen Implementierungen des Backpropagation-Algorithmus geringere erreichbare numerische Darstellungsgenauigkeit für das Lernen besonders problematisch [Frye *et al.*, 1991].

Modifikation

Die Beschleunigung des Backpropagation-Lernalgorithmus ist für die Bestimmung verschiedener Netzwerkparameter von Bedeutung. Die Lerngeschwindigkeit wird durch die Lernrate η bestimmt. Eine zu groß gewählte Lernrate stört jedoch das Konvergenzverhalten des Backpropagation-Algorithmus und kann sogar die Konvergenz verhindern. Modifikationen des

ursprünglichen Lernalgorithmus versuchen daher, das Konvergenzverhalten zu verbessern, um das Lernen zu beschleunigen. Die erste dahingehende Modifikation war die Einführung des Impulsterms (2.12).

Die Fähigkeit zur Generalisierung eines Backpropagation-Netzwerks hängt direkt von der Anzahl der Hidden-Neuronen ab. Jedes Neuron trennt den Eingaberaum durch eine Hyperebene in zwei Teilräume. Die Zuordnung solcher Teilräume zu bestimmten Musterklassen erfolgt durch die Neuronen der Ausgabeschicht (s. Abb. 2.6). Je mehr Hidden-Neuronen ein Netz aufweist, um so genauer kann es eine bestimmte Form durch Hyperebenen approximieren. Die Festlegung einer für ein bestimmtes Problem günstigen Anzahl von Neuronen erfolgt meist empirisch. Dazu ist in der Regel das Lernen mit unterschiedlichen Netwerkkonfigurationen notwendig. Es gibt jedoch auch algorithmische Ansätze, die Netzwerkgröße iterativ festzulegen (Purning) wie beispielsweise [Karnin, 1990].

Beim Backpropagation-Algorithmus erfolgt die Änderung der synaptischen Gewichte in Richtung des negativen Gradienten. Bei der als *Quickprop* bezeichneten Variante des Backpropagation-Algorithmus wird zur Beschleunigung des Lernens neben der Steigung (erste Ableitung) auch die Krümmung (zweite Ableitung) der Fehlerfunktion E berücksichtigt [Rojas, 1993]. Weitere Beispiele für die Erweiterung des Backpropagation-Algorithmus und Implementierungen mit verschiedenen numerischen Näherungen sind [Fukumi und Omatu, 1991; Wawrzynek *et al.*, 1993; Zhang *et al.*, 1993; An, 1996].

Eine Möglichkeit, den Backpropagation-Algorithmus zu beschleunigen, ist die Implementierung auf Parallelrechnern. Auch mit digitalen Signalprozessoren ist hohe Performanz erreichbar [Muller *et al.*, 1992]. In [Zell, 1994] wird die mit Parallelrechnern erreichbare Performanz in der Größenordnung von $20 \ldots 400\ Mcps$ angegeben, in [Singer, 1990] wird die Größenordnung von $Gcps$ angegeben. Wenn auf einem Parallelrechner keine leistungsfähigen Gleitkommaein-

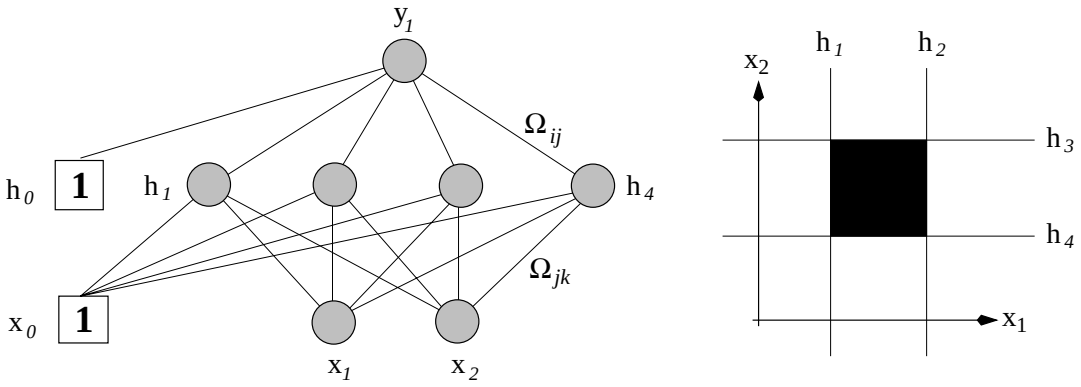


Abbildung 2.6: **Näherung einer Fläche** *durch ein [2-4-1] Backpropagation-Netzwerk. Bei diesem Beispiel sei $y_1 = 0$, wenn (x_1, x_2) einen Punkt innerhalb des schwarzen Quadrats ist, und $y_1 = 1$ für Punkte außerhalb des Quadrats. Die Seiten des Quadrats werden durch vier Geraden h_1, h_2, h_3 und h_4 definiert, die durch vier Hidden-Neuronen repräsentiert werden. Das Ausgangsneuron y_1 verknüpft die vier Ausgänge der Hidden-Neuronen, also die Halbebenen, in geeigneter Weise.*

heiten zur Verfügung stehen, sind unter Umständen wie bei der VLSI-Implementierung geeignete numerische Näherungen notwendig, um den Gewinn an Performanz durch die Parallelverarbeitung ausschöpfen zu können. Dieser Umstand wird beispielsweise bei einer Implementierung [Watanabe *et al.*, 1989] des Backpropagation-Algorithmus auf einem SIMD-Rechner mit 65536 1-Bit-Prozessoren deutlich, bei der eine Lernperformanz von nur $18\ Mcups$ erreicht wurde. Dagegen wird für den moderat parallelen MA-16-Neuroprozessor der Firma Siemens eine Spitzenperformanz von $640\ Mcps$ (bei $40\ MHz$ Taktfrequenz) für rechenintensive Operationen neuronaler Algorithmen angegeben [Ramacher *et al.*, 1993].

2.2.5 Bestehende Neuroprozessoren

Die VLSI-Implementierung neuronaler Netzwerke kann grundsätzlich digital, analog oder optisch sowie auch in Mischformen erfolgen [Glesner und Pöchmüller, 1994; Choi und Sheu, 1995; König, 1995]. Die digitale Implementierung neuronaler Netzwerke hat die Vorteile der hohen Reproduzierbarkeit und frei wählbaren Rechengenauigkeit, was für die Implementierung der Lernphase besonders wichtig ist. Die analoge Implementierung ermöglicht verglichen mit der digitalen Implementierung viel größere Verarbeitungsgeschwindigkeit bei geringerem Energieverbrauch. Die Implementierung mit Hilfe der Optik erlaubt noch größere Verarbeitungsgeschwindigkeiten und bietet darüber hinaus den Vorteil der räumlichen Verschaltung von Neuronen. Nachteil optischer Implementierung ist jedoch das Fehlen geeigneter Speicher und die schwierige Handhabung, wenn man etwa an Erschütterungsempfindlichkeit optischer Bänke denkt.

Es gibt einige industriell entworfene Neuroprozessoren [Choi und Sheu, 1995], die als Koprozessoren für die besonders rechenintensiven arithmetischen Matrixoperationen, wie zur Implementierung neuronaler Netzwerke benötigt, dienen. Als Koprozessor benötigen sie in der Regel einen Steuerrechner, auf dem Teile der Algorithmen ausgeführt werden. Reine Spezialprozessoren sind für die Implementierung neuronaler Netzwerke dann von Interesse, wenn sie etwa in kleinen, mobilen Systemen zum Einsatz kommen sollen. Der ETANN-Chip ($2\ Gcps$) ist ein analoger Neurochip der Firma Intel [Holler *et al.*, 1989]. Der Chip verfügt über analoge Speicherzellen für die synaptischen Gewichte. Das CNAPS System (bestehend aus CNAPS-1064 Chips, die jeweils 64 Prozessorelemente beinhalten) der Firma Adaptive Solutions Inc. ist ein massiv parallel arbeitender Rechner zur Simulation neuronaler Netzwerke. Ein CNAPS System mit 256 Prozessorelementen erreicht $5\ Gcps$ und $1\ Gcups$ beim Backpropagation-Lernen [Glesner und Pöchmüller, 1994]. Der MA-16 (bis 800 Mcps bei 50 MHz Taktfrequenz) der Firma Siemens beinhaltet 4 Multiplizierer und 3 Addierer für effiziente 4×4-Matrixmanipulationen. Bei der Angabe der Spitzenperformanz bleibt der Flaschenhals bezüglich der Kommunikation mit der Außenwelt in der Regel unberücksichtigt. Die an einem Synapse•2-PC-Board mit einem MA-16 und einem mitgelieferten Backpropagation-Programm gemessene Performanz liegt deutlich unter der in der Literatur [Ramacher *et al.*, 1993] angegebenen Spitzenperformanz, was auf einen systembedingten Kommunikationsflaschenhals zurückzuführen sein dürfte.

Digital

Die enge Verbindung zwischen den, für eine effiziente Hardware-Implementierung neuronaler Netzwerke notwendigen, numerischen Näherungen und der resultierenden Implementierung wird bei verschiedenen Publikationen deutlich. In [Lehmann *et al.*, 1993] wird ein systolisches Array [Kung, 1981] mit Festkommaarithmetik (8-Bit-Ergebnisse) und einer Spitzenperformanz von $100\ Mcps$ vorgestellt. In [Watanabe *et al.*, 1993] wird ein digitaler Ein-Chip-Neuroprozessor mit Genauigkeiten von 8 Bit für Ergebnisse und Speicherung der synaptischen Gewichte beschrieben, der eine Spitzenperformanz von $1,4\ Gcps$ erreicht. Ein dem MA-16 der Firma Siemens vergleichbarer Neurochip ist der SAND [Kock und Becher, 1997], mit einer angegebenen Spitzenperformanz von $200\ Mcps$ bei einer Taktfrequenz von $50MHz$. Ein direkter Vergleich zwischen Gleitkommaarithmetik und Festkommaarithmetik und die Auswirkung auf das Verhalten der Regelung eines inversen Pendels werden in [Costa *et al.*, 1997] exemplarisch anhand einer mit FPGAs (Xilinx) realisierten Hardware-Implementierung beschrieben. Die Modifikation eines neuronalen Netzwerks vom Typ Hopfield-Gardner für eine besonders effiziente Hardware-Implementierung durch Übergang von kontinuierlichen synaptischen Kopplungen zu binären Kopplungen zusammen mit der Einführung geeigneter Algorithmen ist in [Hendrich, 1996a,b] beschrieben.

Pulse-Stream

Neuronale Pulse-Stream-Netzwerke weisen bezüglich der Informationsrepräsentation direkte Analogien mit biologischen neuronalen Netzwerken auf. Es handelt sich dabei um digital arbeitende Systeme. Sie sind wissenschaftlich interessant, um biologisch plausible künstliche neuronale Netzwerke zu modellieren. Die Hardware-Implementierung ist hinsichtlich der benötigten Schaltungskomponenten vergleichsweise effizient. Der stochastische Charakters der Informationsrepräsentation erfordert jedoch hochwertige Zufallszahlgeneratoren sowie Mittelung über längere Zeitintervalle und scheint damit für zeitkritische Anwendungen weniger geeignet zu sein. Übersichten über Pulse-Stream-Netzwerke sind etwa in [Hamilton *et al.*, 1992] und in [Kondo und Sawada, 1992] zu finden. Eine für Hardware-Implementierung geeignete Modifikation durch Modulation von Impulsweite und Impulshöhe mit entsprechender Arithmetik ist in [Meador *et al.*, 1991] beschrieben. Die direkte Hardware-Implementierung des Backpropagation-Algorithmus mittels stochastischer Rechenwerke ist in [Köllmann *et al.*, 1996] beschrieben. Die Hardware-Implementierung eines Hopfield-Netzwerks als Pulse-Stream-Netzwerk ist in [Murray *et al.*, 1991] dargestellt. Eine weitere Implementierung mit Hilfe von Pulse-Stream-Arithmetik ist in [Moon *et al.*, 1992] beschrieben. Eine biologisch motivierte Implementierung auf Basis von Signalprozessoren ist in [Wolff *et al.*, 1999] zu finden.

Analog

Die hohe Verarbeitungsgeschwindigkeit analoger neuronaler Netzwerke resultiert aus der direkten Abbildung benötigter elementarer Rechenoperationen auf entsprechende elektrotechnische Systeme. Die Addition kann durch Summierung elektrischer Ströme erfolgen. Die Multiplikation kann mit Hilfe von Widerständen und dem Ohmschen Gesetz erfolgen. Aktivierungs-

funktionen lassen sich mit nicht-linearen Verstärkern realisieren. Verschiedene analoge VLSI-Implementierungen biologischer neuronaler Strukturen sind in [Mead, 1989; Mead und Ismail, 1989; Mead *et al.*, 1991; Andreou *et al.*, 1991] beschrieben. Einige Beispiele für analoge Neurochips sind [Satyanarayana *et al.*, 1989; Valle *et al.*, 1992; Lansner und Lehmann, 1993]. Weitere Implementierungen sind in [MicroNeuro'96, 1996] zu finden.

Wegen der geringen mit analoger Schaltungstechnik erreichbaren Rechengenauigkeit ist Analogtechnik zur vollständigen Implementierung des Lernens mit dem Backpropagation-Algorithmus weniger gut geeignet. Ein Beispiel für die Implementierung von Backpropagation-Lernen mit analoger Schaltungstechnik ist in [Lont und Guggenbühl, 1992] beschrieben. Dabei werden jedoch die synaptischen Gewichte in der Art initialisiert, daß sich das Netzwerk zu Beginn der Lernens schon in der Nähe eines Minimums der Fehlerfunktion befindet.

Hybrid

Die analoge Speicherung synaptischer Gewichte [Durfee und Shoucair, 1992; Lont und Guggenbühl, 1992] mit hinreichender Genauigkeit ist, verglichen mit der digitalen Speicherung, nur schwer zu erreichen. Erfolgt die Speicherung digital, so kann mit Hilfe von Digital-Analog-Wandlern die Brücke zur analogen Verarbeitung hergestellt werden, um den Vorteil der besonders hohen Verarbeitungsgeschwindigkeit analoger neuronaler Netzwerke zu nutzen. Derartige Ansätze sind in [Moopenn *et al.*, 1989; Fisher *et al.*, 1991; Massengill und Mundie, 1992] beschrieben. Ein digital programmierbarer und analog verarbeitender Neurochip – auf Basis eines Backpropagation-Netzwerks – zur Echtzeitklassifizierung von Detektordaten in der Hochenergiephysik [Schiek und Schmidt, 1995], die mit mindestens 10^6 Mustern pro Sekunde zu verarbeiten sind, ist in [Masa *et al.*, 1993; Masa, 1995] beschrieben. Die grundsätzliche Machbarkeit mit einem Backpropagation-Netzwerk wurde im Rahmen einer Diplomarbeit [Larsson, 1991] gezeigt und mit einem algorithmischen (nicht-neuronalen) Ansatz verglichen.

Optisch

Optische Implementierungen neuronaler Netzwerke erreichen Verarbeitungsgeschwindigkeiten, die noch um einige Größenordnungen über mit analogen Implementierungen erreichbaren Verarbeitungsgeschwindigkeiten liegen. Aufgrund der räumlichen Verschaltung [Krishnamoorthy *et al.*, 1992] der Neuronen lassen darüber hinaus mit optischen Implementierungen neuronale Netzwerke mit sehr vielen synaptischen Gewichten realisieren. Problematisch ist jedoch auch bei optischen Implementierungen die geringe erreichbare Darstellungsgenauigkeit der synaptischen Gewichte sowie die optische Speicherung mit zur Implementierung von Lernalgorithmen notwendigen Änderungsmöglichkeiten. Übersichten über Techniken der optischen beziehungsweise opto-elektronischen Implementierung neuronaler Funktionen sind in [Krishnamoorthy *et al.*, 1992], [Kosko, 1992b] und in [Choi und Sheu, 1995] beschrieben.

2.3 Digitale Signalverarbeitung

Zur Vorverarbeitung analoger (Farb-)Videosignale wurden im Rahmen der vorliegenden Arbeit verschiedene Methoden und Algorithmen der digitalen Signalverarbeitung implementiert, die hier kurz zusammenfassend dargestellt werden. Eine geschlossene Darstellung scheint an dieser Stelle notwendig, da in der Literatur sowohl Begriffe als auch Details wie Normierungsfaktoren sehr unterschiedlich verwendet werden, was zwar mathematisch unerheblich ist, sich jedoch für eine konkrete Implementierung als sehr störend bemerkbar macht.

Die *Fourier-Transformation* stellt zusammen mit dem *Faltungssatz* die mathematische Grundlage für die Realisierung spezieller digitaler Filter dar. Die *Quadraturamplitudenmodulation* (QAM) ist ein Verfahren, welches unter anderem bei der Übertragung von Farbvideosignalen nach dem PAL-Verfahren (und auch NTSC-Verfahren) verwendet wird [Mäusel, 1995; Jack, 1996], aber auch zur Datenübertragung mit Modems über Fernsprechleitungen [Paul, 1995]. Für das Verfahren ist die Rekonstruktion eines sinusförmigen Signals (Farbträgersignal bei PAL und NTSC) hinsichtlich Phase und Frequenz notwendig, was mit einer *Phasenregelschleife* (PLL $\approx$ Phase Locked Loop) erfolgen kann. Außerdem werden *Filter* benötigt.

Fourier-Transformation

Mit der *Fourier-Transformation* wird eine Funktion $f(t)$ vom Zeitbereich in eine äquivalente Darstellung einer Spektralfunktion $\hat{\mathrm{F}}(\omega)$ im Frequenzbereich transformiert [Bronstein und Semendjajew, 1981]

$$\hat{\mathrm{F}}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(t) \cdot e^{i\omega t}\, dt. \tag{2.14}$$

Die *inverse Fourier-Transformation* transformiert eine Spektralfunktion $\hat{\mathrm{F}}(\omega)$ vom Frequenzbereich (zurück) in eine äquivalente Darstellung einer Funktion im Zeitbereich

$$f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} \hat{\mathrm{F}}(\omega) \cdot e^{-i\omega t}\, d\omega. \tag{2.15}$$

Die Funktionen $f(t)$ und $\hat{\mathrm{F}}(\omega)$ sind zwei gleichwertige Repräsentationen einer Funktion [Press *et al.*, 1992].

Bezeichnungen

Die Gleichungen der Form (2.14) und (2.15) werden zusammengenommen auch als *Fourier-Transformationsgleichungen* oder als *Fourier-Transformationspaar* bezeichnet. Die Zuordnung der Bezeichnungen *Fourier-Transformation* und *inverse Fourier-Transformation* zu den Gleichungen (2.14) und (2.15) ist in der Literatur leider nicht einheitlich. So steht beispielsweise der Begriff *Fourier-Transformation* in [Bronstein und Semendjajew, 1981; Press *et al.*, 1992; Hess, 1993] für die Form (2.14) wogegen in [Hamming, 1983; Rost, 1983; Jähne, 1993; Gerdsen, 1996] dafür der Begriff *inverse Fourier-Transformation* verwendet wird. Im Rahmen der vorliegenden Arbeit wird dieselbe Vereinbarung zur Definition der Fourier-Transformation wie

in *Mathematica* [Wolfram, 1997] verwendet, die in den Naturwissenschaften üblich ist (speziell in der Elektrotechnik hat der Exponent jedoch oft das umgekehrte Vorzeichen).

Normierung

In der Literatur werden unterschiedliche Normierungsfaktoren verwendet (1, $\frac{1}{2\pi}$, $\frac{1}{\sqrt{2\pi}}$, ...), was dann besonders zu beachten ist, wenn Gleichungen von verschiedenen Quellen kombiniert werden. Bezüglich der Integrationsgrenzen ist folgendes zu bemerken: Für periodische Funktionen genügt es, über genau eine Periode zu integrieren, beispielsweise von $0 \ldots 2\pi$, wenn die Funktion mit 2π periodisch ist. Durch geeignete Skalierung des Funktionsarguments einer periodischen Funktion ist eine Periodizität von 2π immer zu erreichen.

Abtasttheorem

Ein Signal $f(t)$ heißt *bandbegrenzt*, wenn sein Spektrum $\hat{\mathrm{F}}(\omega)$ oberhalb einer Kreisgrenzfrequenz $\omega_g = 2\pi \cdot f_g$ beziehungsweise oberhalb einer Grenzfrequenz $f_g = \frac{\omega_g}{2\pi}$ verschwindet

$$\hat{\mathrm{F}}(\omega) = 0 \quad \text{für} \quad |\omega| > \omega_g. \tag{2.16}$$

Wird ein kontinuierliches Signal periodisch mit einer Abtastkreisfrequenz $\omega_a = 2\pi \cdot f_a$ abgetastet, so wird das Spektrum $\hat{\mathrm{F}}(\omega)$ des Signals $f(t)$ periodisch bei ganzzahligen Vielfachen der Abtastkreisfrequenz $\omega_a = 2\pi/T$ wiederholt, d.h. es ist $\hat{\mathrm{F}}(\omega + n \cdot \omega_a) = \hat{\mathrm{F}}(\omega)\ \forall\ n \in \{0, \mathbb{N}\}$. Ist die Bedingung

$$\omega_a > 2 \cdot \omega_g \tag{2.17}$$

erfüllt, so kann das kontinuierliche, abgetastete Signal mit einem Tiefpaßfilter mit einer Grenzkreisfrequenz ω_g vollständig wiederhergestellt werden. Die halbe Abtastfrequenz $f_g = \omega_g/2\pi$ wird als *Nyquist-Frequenz* $f_n = \frac{1}{2} \cdot f_g$ bezeichnet. Eine ausführliche Abhandlung der Abtastung und des Abtasttheorems ist etwa in [Press *et al.*, 1992; Tietze und Schenk, 1993; Mildenberger, 1995; Girod *et al.*, 1997] zu finden.

Signalrauschverhältnis

Ein Merkmal zur Charakterisierung von Signalen ist der als Signalrauschverhältnis bezeichnete Quotient aus Effektivwert U_S eines Nutzsignals und Effektivwert U_R des dem Nutzsignal überlagerten Rauschens $S = U_S/U_R$. Anstelle des Signalrauschverhältnisses wird auch der *Signalrauschabstand* in *Dezibel* angegeben

$$S[\mathrm{dB}] = 20 \cdot \log_{10}\left(\frac{U_S}{U_R}\right). \tag{2.18}$$

Der mit Analog-Digital-Wandlern und Digital-Analog-Wandlern maximal erreichbare Signalrauschabstand hängt direkt von deren Wortbreite ab. Mit jedem zusätzlichen Bit mehr verdoppelt

sich der Dynamikumfang der Signaldarstellung. Da $6 \approx 20 \cdot \log_{10}(2)$ ist, vergrößert sich der erreichbare Signalrauschabstand mit jedem Bit um 6dB. Der durch *Quantisierungsrauschen* bestimmte Signalrauschabstand eines N-Bit-Umsetzers beträgt für sinusförmige Signale bei Vollaussteuerung [Tietze und Schenk, 1993]

$$S_N \approx N \cdot 6\text{dB}. \tag{2.19}$$

Aufgrund von Nichtlinearitäten von Wandlern sowie analogen Rauschquellen ist der tatsächlich erreichte Signalrauschabstand meist geringer. Stellvertretend für die beiden Begriffe Signalrauschverhältnis *und* Signalrauschabstand wird oft nur der Begriff Signalrauschverhältnis (*engl. SNR ≈ Signal Noise Ratio*) verwendet, was jedoch gegebenenfalls durch die Angabe der Einheit dB eindeutig wird.

Faltung

Die Faltung ist zusammen mit dem Faltungssatz [Bronstein und Semendjajew, 1981] die mathematische Grundlage zur Implementierung linearphasiger Filter zur digitalen Filterung im Zeitbereich [Press *et al.*, 1992]. Die Faltung (f gefaltet g) zweier Funktionen $f(t)$ und $g(t)$ ist

$$(f \circ g)(t) = \int_{-\infty}^{+\infty} f(\tau) \cdot g(t-\tau)\, d\tau. \tag{2.20}$$

Faltungssatz

Die Fourier-Transformierte der Faltung $(f \circ g)(t)$ zweier Funktionen $f(t)$ und $g(t)$ ist bis auf einen Faktor gleich dem Produkt der Fourier-Transformierten $\hat{\mathrm{F}}(\omega)$ und $\hat{\mathrm{G}}(\omega)$ der beiden Funktionen $f(t)$ und $g(t)$

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} (f \circ g)(t) \cdot e^{i\omega t}\, dt = \sqrt{2\pi} \cdot \hat{\mathrm{F}}(\omega) \cdot \hat{\mathrm{G}}(\omega). \tag{2.21}$$

Anders ausgedrückt entspricht also die Multiplikation zweier Spektren $\hat{\mathrm{F}}(\omega)$ und $\hat{\mathrm{G}}(\omega)$ im Frequenzraum einer Faltung $(f \circ g)(t)$ der inversen Fourier-Transformierten $f(t)$ und $g(t)$ im Zeitraum und umgekehrt. Diese Zusammenhänge zwischen Multiplikation und Faltung werden auch als *Multiplikationstheorem* oder *Modulationstheorem* bezeichnet [Lüke, 1995]. Die Vorfaktoren ($\frac{1}{\sqrt{2\pi}}$ und $\sqrt{2\pi}$) in (2.21) sind abhängig von der Wahl der Normierungsfaktoren des Fourier-Transformationspaars (2.14) und (2.15).

2.3.1 Digitale Filter

Zur rein digitalen Verarbeitung analoger Videosignale werden Tiefpaßfilter benötigt. Im einfachsten Fall kann mit Hilfe eines Tiefpaßfilters der Farbsignalanteil von einem PAL-Videosignal (oder auch NTSC-Videosignal) unterdrückt werden, um nur den Graubildanteil zu extrahieren. Zur Demodulation des Farbsignalanteil eines PAL-Videosignals werden weitere Tiefpaßfilter

benötigt. Ebenso erfordert die Demodulation des Farbsignals eine *Phase Locked Loop* (PLL), die ihrerseits ein Tiefpaßfilter benötigt. Hochpaßfilter oder Bandpaßfilter, die sich auf Tiefpaßfilter zurückführen lassen [Hamming, 1983; Hess, 1993; Tietze und Schenk, 1993], werden zur digitalen Verarbeitung des PAL-Videosignals nicht benötigt.

Auch wenn nur Graustufenbilder verarbeitet werden sollen, so ist die Unterdrückung der Farbvideosignalkomponente durch geeignete Filterung von Bedeutung, da handelsübliche (und preiswerte) Videokameras in der Regel über einen PAL-Farbvideosignalausgang verfügen.

FIR-Filter und IIR-Filter haben unterschiedliche Vorteile und Nachteile bezüglich ihres Verhaltens, der Handhabung und des Realisierungsaufwands [Bowen und Brown, 1982]. FIR-Filter (*Finite Impulse Response Filter*) haben eine endliche Impulsantwort, daher der Name. Ein großer Vorteil der FIR-Filter ist ihr lineares Phasenverhalten: Die Phasenverschiebung ist *linear* von der Signalfrequenz abhängig. Dadurch sind sie einfach zu handhaben. Anders ist das bei IIR-Filtern (*Infinite Impulse Response*), bei denen die Phasenverschiebung *nicht-linear* von der Signalfrequenz abhängt. Darüber hinaus neigen IIR-Filter zu Instabilitäten, was insbesondere beim Übergang zu ganzzahliger Arithmetik zu berücksichtigen ist [Hess, 1993]. FIR-Filter und IIR-Filter sind in [Hamming, 1983] ausführlich beschrieben.

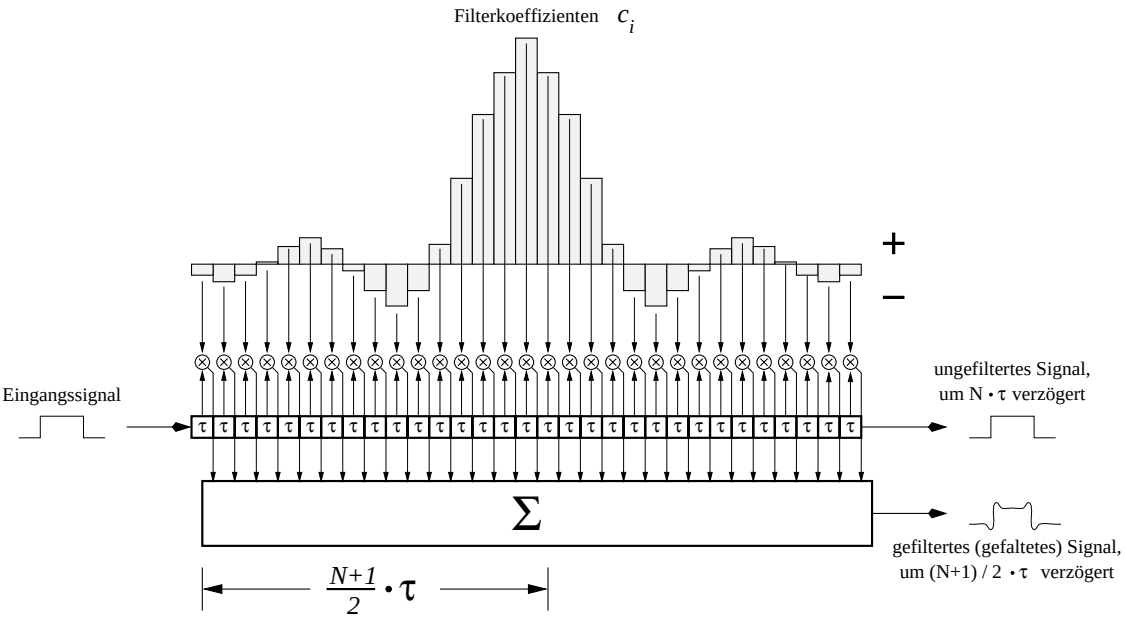


Abbildung 2.7: **FIR-Filter Prinzip.** *Wird ein Signal mit einer Abtastfrequenz $f_s = 1/\tau$ in ein Filter mit N Filterkoeffizienten c_i getaktet, so erscheint das gefilterte Signal bei symmetrischer Anordnung der Filterkoeffizienten c_i am Ausgang nach einer Verzögerungszeit $t = \frac{1}{2} \cdot (N+1) \cdot \tau$. Dazu wird in jedem Schritt die Summe Σ der Filterkoeffizienten mit den verzögerten Signalwerten berechnet. Eine solche Struktur erlaubt die Berechnung einer diskreten Faltung entsprechend der Gleichung (2.20).*

Filterkoeffizienten

Filterung ist die Multiplikation eines gewünschten Spektrums (Frequenzgangs) mit dem Spektrum eines Signals. Da Signale im technischen Kontext in der Regel im Zeitbereich vorliegen, bietet sich die Filterung mit Hilfe der Faltung an. Dazu ist eine Funktion $g(t)$ zu bestimmen, mit welcher ein Signal $f(t)$ gefaltet wird. Der Funktion $g(t)$ entsprechen im zeitdiskreten Fall die *Filterkoeffizienten*.

Ein Filter kann grundsätzlich auch mit Hilfe der Fourier-Transformation (speziell der Fast-Fourier-Transformation (FFT)) und Multiplikation von Spektren implementiert werden [Press *et al.*, 1992]. Der Vorteil der Faltung ist, daß nur einfache arithmetische Operationen zu implementieren sind. Außerdem ermöglicht die Faltung eine einfache Implementierung mit einer Pipeline-Struktur, so daß bei jedem Zeitschritt eine neuer, gefilterter Wert am Ausgang erscheint (s. Abb. 2.8).

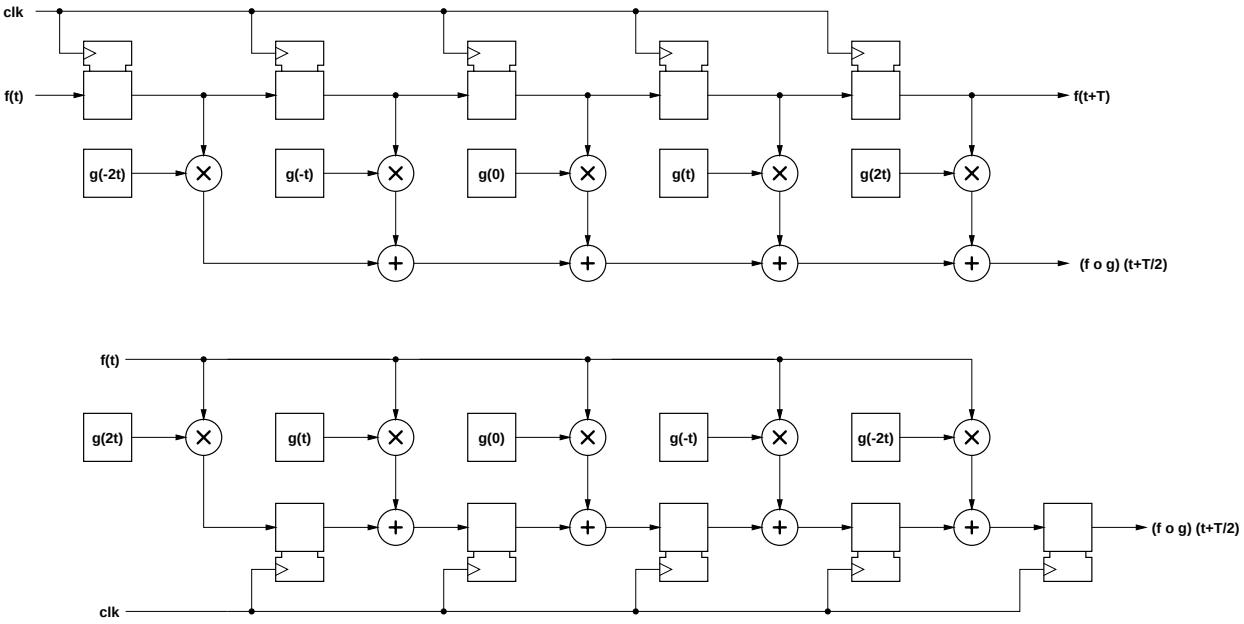


Abbildung 2.8: **FIR-Filter** *nach [Tietze und Schenk, 1993]. Ein Signal $f(t)$ wird mit den Filterkoeffizienten gefaltet. Bei der oben dargestellten Implementierungsvariante wird die maximal mögliche Taktfrequenz in ungünstiger Weise von der Summe der Laufzeiten durch die Kette der Addierer $\oplus$ bestimmt. Durch einfaches Umordnen ergibt sich eine Pipeline-Struktur in der unten dargestellten Weise, bei der die Verarbeitungsgeschwindigkeit nur noch von der Signallaufzeit einer Multiplizierer-Addierer-Stufe $\otimes \longrightarrow \oplus$ bestimmt wird und somit eine wesentlich höhere Verarbeitungsgeschwindigkeit erlaubt.*

Berechnung von Filterkoeffizienten

In der Literatur [Hamming, 1983; Kammeyer und Kroschel, 1992; Hess, 1993] lassen sich zahlreiche Verfahren zur Konstruktion von Filtern und zur Bestimmung von Filterkoeffizienten finden. Bei den verschiedenen Verfahren sind technische Randbedingungen in unterschiedlicher Weise berücksichtigt. Für die digitale Filterung durch Faltung (2.20) kann die Berechnung der dafür notwendigen Impulsantwortfunktion $g(t)$ grundsätzlich direkt durch inverse Fourier-Transformation eines gewünschten Frequenzgangs $\hat{\mathrm{G}}(\omega)$ erfolgen. Für einen idealen Tiefpaßfilter (d.h. mit rechteckigem Frequenzgang) mit einer Grenzkreisfrequenz ω_g (Grenzfrequenz $f_g = \omega_g/2\pi$) ergibt sich ein Ausdruck der Form

$$g(t) = \frac{1}{\pi} \cdot \frac{\sin(\omega_g \cdot t)}{t} \qquad \text{mit} \qquad \lim_{t \to 0} g(t) = \frac{\omega_g}{\pi} = 2 f_g . \tag{2.22}$$

Bei einer Abtastfrequenz $f_{clk} = \omega_{clk}/2\pi$ ergibt sich eine Zeitdiskretisierung von $\Delta t = 1/f_{clk}$, womit sich die Filterkoeffizienten (d.h. die Funktionswerte von (2.22) an den diskreten Stellen $t = k \cdot \Delta t$ mit $k = \ldots -2, -1, 0, +1, +2, \ldots$) errechnen lassen.

Diskrete Faltung

Für große Beträge von t geht (2.22) gegen null. Bei der Implementierung eines auf der Faltung beruhenden Filters erfolgt die Berechnung des Faltungsintegrals (2.20) näherungsweise durch Berechnung einer Summe der Form

$$y(n) = \sum_{k=-N}^{+N} c_k \cdot x_{n-k} \tag{2.23}$$

wobei

$$c_k = g(t) \qquad \text{mit} \qquad t = k \cdot \frac{1}{f_{clk}} \quad \text{für} \quad k = \{-N, \ldots, -1, 0, +1, \ldots, +N\} \tag{2.24}$$

die Filterkoeffizienten sind. Um Artefakte zu reduzieren, die bei der diskreten Faltung durch die endliche Zahl der Filterkoeffizienten auftreten, werden die Filterkoeffizienten häufig mit geeigneten *Fensterfunktionen* multipliziert.

Fensterfunktion

Die diskrete Faltung (2.23) über einen *endlichen Zeitbereich* (Zeitfenster) ist gleichbedeutend mit einer Multiplikation einer zu filternden Funktion im Zeitraum mit einem rechteckigen Zeitfenster der Breite $T = (2 \cdot N + 1)/f_{clk}$, was der Faltung der Fourier-Transformierten im Frequenzraum entspricht und eine Verbreiterung der Spektrallinien sowie eine Verzerrung des gewünschten Frequenzgangs des Filters zur Folge hat. Durch Wichtung der Filterkoeffizienten mit einer geeigneten Fensterfunktion kann diese Störung verringert werden [Kammeyer und Kroschel, 1992; Hess, 1993]. Im Rahmen der Arbeit wurde eine Fensterfunktion der Form

$$\hat{c}_k = c_k \cdot \frac{1}{2} \cdot \left(1 - \cos(\frac{2\pi \cdot (k+N)}{2N+1})\right) \quad \text{für} \quad k = -N, \ldots, -1, 0, +1, \ldots, +N \qquad (2.25)$$

verwendet, die in der Literatur als *von Hann Funktion* oder als *Hanning-Fenster* [Press *et al.*, 1992] bezeichnet wird. Die Filterkoeffizienten c_k werden also einfach durch gewichtete Filterkoeffizienten $\hat{c}_k$ ersetzt. Wie sich jedoch anhand von Simulationen zeigte, waren Abweichungen vom Frequenzgang des idealen Tiefpaßfilters für die digitale Verarbeitung analoger Farbvideosignale, d.h. für die Qualität extrahierter Bilder, von untergeordneter Bedeutung.

Filterlänge

Neben der Bestimmung der Filterkoeffizienten selbst, ist die Festlegung der Zahl n ($n = 2N + 1$ in Gleichung (2.23)) der Filterkoeffizienten (Ordnung $n - 1$ bzw. $2N$ in Gleichung (2.23)) des betreffenden Filters für das Filterverhalten – den Frequenzgang eines Filters – notwendig. Bei einem FIR-Filter wird das Faltungsintegral (2.20), welches sich über ein unendliches Zahlenintervall erstreckt, durch die diskrete Faltung (2.23), die sich über ein endliches Zahlenintervall, mit endlich vielen Filterkoeffizienten c_i erstreckt, angenähert. Die Menge aller Frequenzen, die ein Filter durchläßt, wird als *Durchlaßbereich* bezeichnet, und die Menge aller Frequenzen, die ein Filter nicht durchläßt, wird als *Sperrbereich* bezeichnet [Hess, 1993] – bei realen Filtern befindet sich zwischen Durchlaßbereich und Sperrbereich ein stetiger *Übergangsbereich*.

Konstruktive Entwurfsverfahren für FIR-Filter sind in [Fliege, 1993; Hess, 1993] beschrieben, die iterativ einen Satz von Filterkoeffizienten für ein zu spezifizierendes Toleranzschema des gewünschten Frequenzgangs bestimmen. Wesentliche Parameter sind dabei die Welligkeit des Frequenzgangs im Durchlaßbereich, die relative Übergangsbreite zwischen Durchlaßbereich und Sperrbereich sowie die Dämpfung im Sperrbereich.

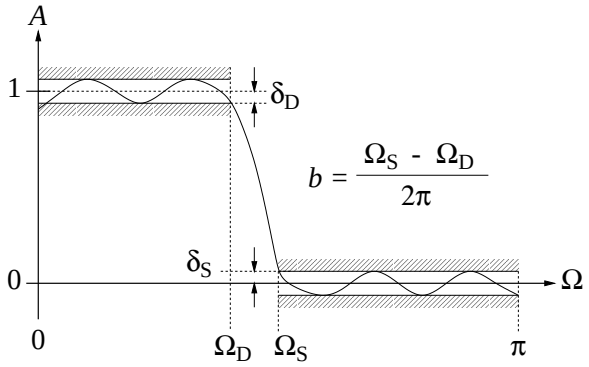


Abbildung 2.9: **Toleranzbereich** *nach [Fliege, 1993]. Die Größe δ_D spezifiziert den sog.* Ripple *des Durchlaßbereichs, die Toleranzbreite δ_S spezifiziert den Ripple des Sperrbereichs. Die Größe δ_S repräsentiert die Signalunterdrückung des Filters im Sperrbereich. Die Toleranzbreite $b = (\Omega_S - \Omega_D)/2\pi$, die als* relative Übergangsbreite *bezeichnet wird, spezifiziert den Übergangsbereich. In der Abbildung repräsentiert π die Nyquist-Frequenz und 2π repräsentiert die Abtastfrequenz.*

Tiefpaßfilter, die für die digitale Verarbeitung analoger PAL-Farbvideosignale benötigt werden, lassen sich durch ein Toleranzschema, wie in Abbildung 2.9 dargestellt, charakterisieren. Der gewünschte Frequenzgang soll dabei in dem vorgegebenen Toleranzschema verlaufen. Die Zahl der Filterkoeffizienten sei im folgenden Abschnitt mit N bezeichnet. Die Abschätzung nach *Bellanger* [Bellanger, 1984]

$$N \approx \frac{2}{3} \cdot \log_{10}\left(\frac{1}{10 \cdot \delta_D \cdot \delta_S}\right) \cdot \frac{1}{b} \tag{2.26}$$

oder die Abschätzung nach *Kaiser*, veröffentlicht in [Rabiner und J.H. McCellan, 1975],

$$N \approx \frac{-20 \cdot \log_{10}\sqrt{\delta_D \cdot \delta_S} - 13}{14,6 \cdot b} + 1 \tag{2.27}$$

führen zu ähnlichen Abschätzungen für die Zahl N der Filterkoeffizienten [Fliege, 1993] bei vorgegebenen Ripple. Bei gleichem Ripple des Durchlaßbereichs δ_D und des Sperrbereichs δ_S, was sich durch Wahl geeigneter Parameter beim Filterentwurf erreichen läßt [Fliege, 1993], vereinfacht sich (2.26) mit $\delta := \delta_D = \delta_S$ zu

$$N \approx \frac{2}{3} \cdot \log_{10}\left(\frac{1}{10 \cdot \delta^2}\right) \cdot \frac{1}{b} \tag{2.28}$$

und (2.27) zu

$$N \approx \frac{-20 \cdot \log_{10}(\delta) - 13}{14,6 \cdot b} + 1. \tag{2.29}$$

Mit (2.18) ergibt sich nach einigen elementaren Umformungen (s. Abschnitt B.4, S. 166 und Abschnitt B.4, S. 167) für (2.26) der Ausdruck

$$\delta[\text{dB}] \approx -15 \cdot N \cdot b - 10 \tag{2.30}$$

und für (2.29) ergibt sich

$$\delta[\text{dB}] \approx -14,6 \cdot (N-1) \cdot b - 13. \tag{2.31}$$

Dabei wird ein linearer Zusammenhang zwischen dem Produkt $N \cdot b$ und dem Ripple in logarithmischer Darstellung – in Dezibel – deutlich. Außerdem wird dabei die Ähnlichkeit der beiden Abschätzungen (2.26) und (2.27) deutlich.

2.3.2 Quadraturamplitudenmodulation

Diese Methode ist wesentlicher Bestandteil der Farbinformationsübertragung des PAL-Verfahrens (und des NTSC-Verfahrens). Die Quadraturamplitudenmodulation (QAM) kommt bei Modemanwendungen (Trägerfrequente Übertragung digitaler Signale über analoge Leitungen) in sehr unterschiedlichen Frequenzbändern (Datenübertragung über Telefonleitungen (Kbps), Übertragung digitaler Videosignale in Kabelnetzen (Mbps)) [Hewlett-Packard, 1997; Reimers, 1997] zur Anwendung.

QAM – Modulation

Bei der Modulation werden zwei in geeigneter Weise bandbegrenzte Signale $u(t)$ und $v(t)$ auf zwei gegeneinander um 90^o phasenverschobene Trägersignale $\cos(wt)$ und $\sin(wt)$ gemäß

$$c(t) = u(t) \cdot \cos(\omega t) + v(t) \cdot \sin(\omega t) \tag{2.32}$$

amplitudenmoduliert. Die Signale $u(t)$ und $v(t)$ lassen sich wieder aus $c(t)$ zurückgewinnen, da $\cos(\omega t)$ und $\sin(\omega t)$ orthogonal sind. Wenn $\cos(\omega t)$ einen Maximalwert annimmt, ist $\sin(\omega t)$ gleich null und umgekehrt, so daß zu den betreffenden Zeitpunkten $c(t) = u(t)$ beziehungsweise $c(t) = v(t)$ ist.

QAM – Demodulation

Die Rückgewinnung der beiden Signale $u(t)$ und $v(t)$ aus dem zusammengesetzten Signal $c(t)$ könnte grundsätzlich durch Abtastung bei Nulldurchgängen der Sinus- bzw. Cosinus-Komponente erfolgen. Dieses Vorgehen würde jedoch ein äußerst genaues Abtasten dieser Zeitpunkte (d.h. eine sehr hohe Abtastfrequenz, mit einem Vielfachen der Nyquist-Frequenz) erfordern und wäre zudem gegenüber verrauschten Signalen äußerst fehleranfällig. Daher erfolgt die Rückgewinnung der beiden Signale $u(t)$ und $v(t)$ aus dem zusammengesetzten Signal $c(t)$ durch Multiplikation mit den Trägersignalen

$$\begin{array}{rcccl} c_{cos}(t) & = & \frac{1}{2}u(t) + \frac{1}{2}u(t) \cdot \cos(2\omega t) & + & \frac{1}{2}v(t) \cdot \sin(2\omega t) \\ c_{sin}(t) & = & \frac{1}{2}u(t) \cdot \sin(2\omega t) & + & \frac{1}{2}v(t) - \frac{1}{2}v(t) \cdot \cos(2\omega t) \end{array} \tag{2.33}$$

mit anschließender Tiefpaßfilterung (TP[]) der Signale $c_{cos}(t)$ und $c_{sin}(t)$ mit einer Grenzkreisfrequenz ω_g, die kleiner als die Trägerkeisfrequenz ω ist. Durch die Tiefpaßfilterung verschwin-

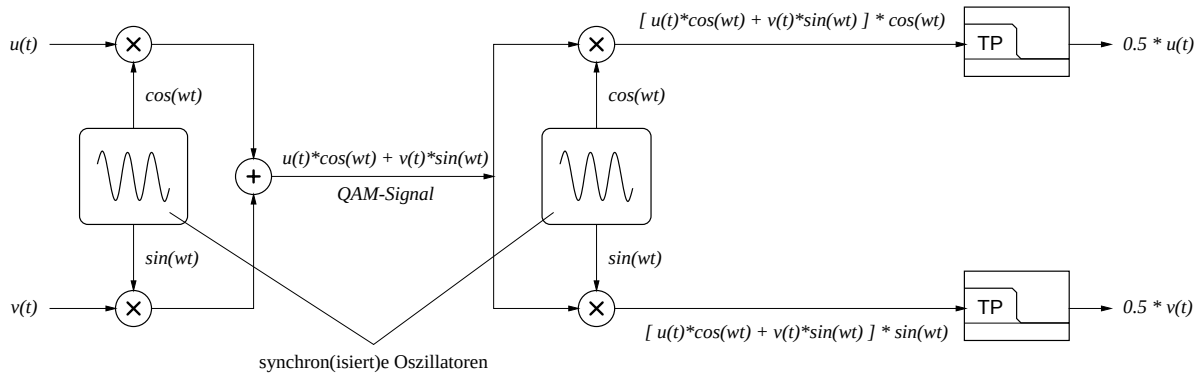


Abbildung 2.10: **QAM** *Modulator und Demodulator.*

den genau die Terme, die einen Faktor $\sin(2\omega t)$ oder $\cos(2\omega t)$ enthalten, so daß die demodulierten Signale

$$\begin{aligned} u_d(t) &= 2 \cdot \mathrm{TP}[c(t) \cdot \cos(\omega t)] \\ v_d(t) &= 2 \cdot \mathrm{TP}[c(t) \cdot \sin(\omega t)] \end{aligned} \tag{2.34}$$

zur Verfügung stehen. Diese Methode ist aufgrund ihres zeitlich ausgedehnten Charakters weit weniger empfindlich gegenüber Rauschen. Zudem genügt die Abtastung des Signals $c(t)$ mit einer Abtastfrequenz, die mit dem Abtasttheorem konform ist.

Zur Demodulation ist die Kenntnis der Frequenz und Phase des bei der QAM-Modulation verwendeten Trägers notwendig. Es sind also in Frequenz und Phase der zur Modulation verwendeten Trägersignale äquivalente Signale $\sin(\omega t)$ und $\cos(\omega t)$ zu erzeugen, was mit einer Phasenregelschleife erreicht werden kann.

2.3.3 Phasenregelschleife

Eine Phasenregelschleife (*engl.* ≈ *Phase Locked Loop*, PLL) ermöglicht es, ein Signal der Form $c(t) = \sin(\omega t)$ mit einem zugeführten Signal der Form $c(t) = \sin(\omega' t + \Delta\varphi)$ so zu synchronisieren, daß nach einer gewissen Regelzeit sowohl Frequenzdifferenz $\Delta\omega = \omega - \omega'$ als auch Phasendifferenz $\Delta\varphi$ zwischen beiden Signalen verschwindet. Abhängig von der Implementierung und speziell in Abhängigkeit von den verarbeiteten Signalen wird in [Best, 1996] zwischen *analoger, linearer, digitaler* und *programmierter* Phasenregelschleife unterschieden. Analoge PLLs stellen mit analogen Komponenten realisierte PLLs dar, die analoge Signale verarbeiten. Lineare PLLs sind Elemente der digitalen Signalverarbeitung, die analogen PLLs funktional ähnlich sind. Digitale PLLs [Weste und Eshraghian, 1993] werden beispielsweise zur Rekonstruktion von Taktsignalen aus digitalen Datenströmen eingesetzt [Gerdsen, 1996]. Analoge, lineare und digitale PLLs können in Form von Programmen realisiert werden.

Phasendetektor

Wesentlicher Bestandteil der PLL ist der *Phasendetektor*. Dieser liefert ein direkt von der Phasendifferenz $\Delta\varphi$ abhängiges Signal, das zur Steuerung der *Oszillatorkreisfrequenz* ω verwendet

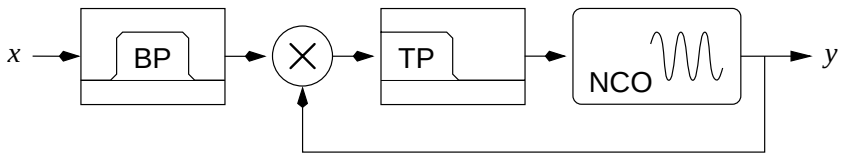


Abbildung 2.11: **PLL-Schema** *BP = Bandpaß (optional) zur Begrenzung des Eingangssignalspektrums, ⊗ = Multiplizierer zur Bildung eines Produkts gemäß (2.37) und TP = Tiefpaß zur Bestimmung der Phase, NCO (*engl. ≈ numerically controlled oscillator*) = numerisch steuerbarer Oszillator, dessen Frequenz oder auch Phase numerisch einstellbar ist.*

wird. Die Phasendifferenz zwischen zwei sinusförmigen Signalen ergibt sich durch Multiplikation der Signale

$$e^{i\omega t} \cdot e^{i\omega t + \Delta\varphi} = e^{2 \cdot i\omega t + \Delta\varphi} \tag{2.35}$$

mit anschließender Tiefpaßfilterung zur Unterdrückung des durch die Multiplikation entstandenen Signalanteils mit der doppelten Kreisfrequenz (2ω). Für zwei reelle Signale $\sin(\omega t)$ und $\sin(\omega' t + \Delta\varphi)$ ergibt sich mit $\sin(x) \cdot \sin(y) = \frac{1}{2} \cdot [\cos(x - y) - \cos(x + y)]$ für $x = \omega t$ und $y = \omega t + \Delta\varphi$ bei $\omega = \omega'$ die Phasendifferenz $\Delta\varphi = \varphi - \varphi'$ als

$$\sin(\omega t) \cdot \sin(\omega' t + \Delta\varphi) \;=\; \frac{1}{2}\left[\cos(\omega t - \omega' t - \Delta\varphi) - \cos(\omega t + \omega' t + \Delta\varphi)\right], \tag{2.36}$$

was sich für $\omega = \omega'$ zu

$$\sin(\omega t) \cdot \sin(\omega t + \Delta\varphi) \;=\; \frac{1}{2}\left(\cos(-\Delta\varphi) - \cos(2 \cdot \omega t + \Delta\varphi)\right] \tag{2.37}$$

vereinfachen läßt. Der Term $-\cos(2 \cdot \omega t + \Delta\varphi)$ des Produktsignals (2.37) wird durch Tiefpaßfilterung unterdrückt, so daß der Term $\frac{1}{2}\cos(\Delta\varphi)$ übrig bleibt ($\cos(-x) = \cos(x)$).

2.3.4 Oszillator

Zur QAM-Modulation und QAM-Demodulation müssen Sinus- und Cosinus-Signale mit sehr genauer Frequenz und Phase erzeugt werden. Dabei muß die Signalerzeugung mit der Abtastfrequenz der Videosignale erfolgen. Die Berechnung über die Taylor-Reihe der Sinus- und Cosinus-Funktion [Bronstein und Semendjajew, 1981] oder über Chebyshev-Polynome [Press *et al.*, 1992], wie in Standard-C-Bibliotheken [Plauger, 1992] implementiert, ist für die Echtzeitverarbeitung von Videosignalen aufgrund der vielen dafür notwendigen Rechenoperationen ungeeignet. Eine Alternative ist die *Direct Digital Synthesis (DDS)*. Die Erzeugung eines sinusförmigen Signals erfolgt dabei durch fortschreitende Akkumulation einer Phasenkonstante $\Delta\varphi$ mit jedem Abtastschritt (vgl. *Quadrature Subcarrier Generation* mit *Ratio Counter* [Jack, 1996]).

Die DDS ermöglicht bei geeigneten numerischen Näherungen eine besonders effiziente Schaltungsimplementierung der Sinus-Funktion, die für schnelle Echtzeitverarbeitung geeignet ist. Dabei lassen sich mit handelsüblichen EPLDs derzeit Abtastfrequenzen im Bereich von bis zu $100\ MHz$ erreichen. Eine Ultra-Sparc II-Workstation unter Solaris (bei einer CPU-Taktfrequenz von $200\ MHz$) oder ein Pentium-PC unter Linux (bei einer CPU-Taktfrequenz von $166\ MHz$) erreichen dem gegenüber "nur" eine Abtastfrequenz von etwa $1\ MHz$, d.h. diese Rechner können, unter Vernachlässigung von Kommunikation mit der Außenwelt, um 10^6 Sinus-Funktionswerte in Fließkommadarstellung pro Sekunde berechnen. Zwar ist die numerische Genauigkeit der Fließkommadarstellung der Berechnungen deutlich höher als bei der DDS, mit Tabellen von vielleicht einigen hundert KByte Größe. Bei bestimmten Echtzeitanwendungen, wie der digitalen Erzeugung von PAL-Farbvideosignalen, kann jedoch auf die sehr hohe numerische

Genauigkeit von Standardprozessoren verzichtet werden und dadurch ein enormer Geschwindigkeitsvorteil einer anwendungsspezifischen Hardware-Implementierung voll ausgeschöpft werden.

Direkte Digitalsynthese

Wie der Name schon andeutet, stellt die DDS eine sehr direkte Methode zur Erzeugung von Signalen dar und ist grundsätzlich nicht auf die Erzeugung von sinusförmigen Signalen beschränkt. Im Kontext der digitalen Farbvideosignalverarbeitung werden für die QAM jedoch nur sinusförmige Signale benötigt. Grundlage der DDS ist eine Tabelle mit Sinus-Funktionswerten einer ganzen Periode. Um Speicherplatz zu sparen, kann eine volle Sinus-Periode grundsätzlich auch aus einer halben Periode oder einer Viertelsinusperiode gewonnen werden. Die Funktionswerttabelle kann in einem ROM gespeichert sein [Jack, 1996]. Vorgreifend sei an dieser Stelle bemerkt, daß die Repräsentation einer solchen Tabelle auch durch ein geeignetes Schaltnetz aus Logikgattern sinnvoll ist, wenn für eine bestimmte Aufgabe eine geringere Datenwortbreite für ein Sinus-Signal ausreichend ist. Ein sinusförmiges Signal $\mathrm{s}(t, \Phi)$ mit einer Frequenz ν und einer Phase Φ kann als Funktion der Zeit t in der Form

$$\mathrm{s}(t, \Phi) = \sin(\underbrace{2\pi \cdot \nu \cdot t}_{\varphi(t)} + \Phi) \tag{2.38}$$

geschrieben werden. Durch Wahl der Phasenverschiebungskonstante Φ im Funktionsargument von (2.38) kann das sinusförmige Signal (2.38) mit beliebiger Phasenlage Φ erzeugt werden. Zur Vereinfachung wird im folgenden jedoch $\Phi := 0$ gesetzt. Das Funktionsargument in (2.38) ist ein Phasenwinkel $\varphi(t)$ zum Zeitpunkt t. Durch zeitdiskrete Abtastung mit einer Abtastfrequenz $f_s = 1/\Delta t_s$ ergibt sich eine Folge von Phasenwinkeln $\varphi(t) = \{\varphi(t_0), \varphi(t_1), \varphi(t_2), \ldots, \varphi(t_i)\}$ zu diskreten Zeitpunkten $t = \{t_0, t_1, t_2, \ldots, t_i\}$. Zum i-ten Abtastzeitpunkt t_i ist $t_i = i \cdot \Delta t_s$. Für den Phasenwinkel $\varphi_i = \varphi(t_i)$ zum i-ten Abtastzeitpunkt gilt also

$$\varphi_i \;=\; i \cdot \Delta\varphi = \sum_{j=0}^{i-1} \Delta\varphi, \tag{2.39}$$

wobei $\Delta\varphi = 2\pi \cdot \nu \cdot \Delta t_s = 2\pi \cdot \frac{\nu}{f_s}$. Die Berechnung des Phasenwinkels $\Delta\varphi_i$ durch Summieren (2.39) der konstanten Phasenwinkeldifferenz $\Delta\varphi$ ist für die effiziente Hardware-Implementierung bestens geeignet. Damit errechnet sich ein sinusförmiges Signal $\sin_i$ zum i-ten Abtastzeitpunkt t_i zu $\sin_i = \sin(\varphi_i)$ bzw. $\sin_i = \sin(\varphi_i + \Phi)$. Eine Sinus-Funktionstabelle `sin[p]` mit N Tabelleneinträgen über *genau eine* Sinus-Periode wird gemäß

$$\texttt{sin[p]} := \sin(2\pi \cdot \mathtt{p}/\mathtt{N}) \qquad \text{mit} \qquad \mathtt{p} = 0, 1, 2, \ldots, N-1 \tag{2.40}$$

errechnet. Der Tabellenindex, die natürliche Zahl `p` repräsentiert dabei den Phasenwinkel φ. Der Quotient $\mathtt{p}/N$ repräsentiert den auf eine Periode (2π) normierten Phasenwinkel φ. Es ist zwar nicht zwingend notwendig, aber zweckmäßig, für die Zahl der Tabelleneinträge `N` eine Zweierpotenz zu wählen, so daß für natürliche Zahlen `n`, `N` gilt $2^{\mathtt{n}} = \mathtt{N}$.

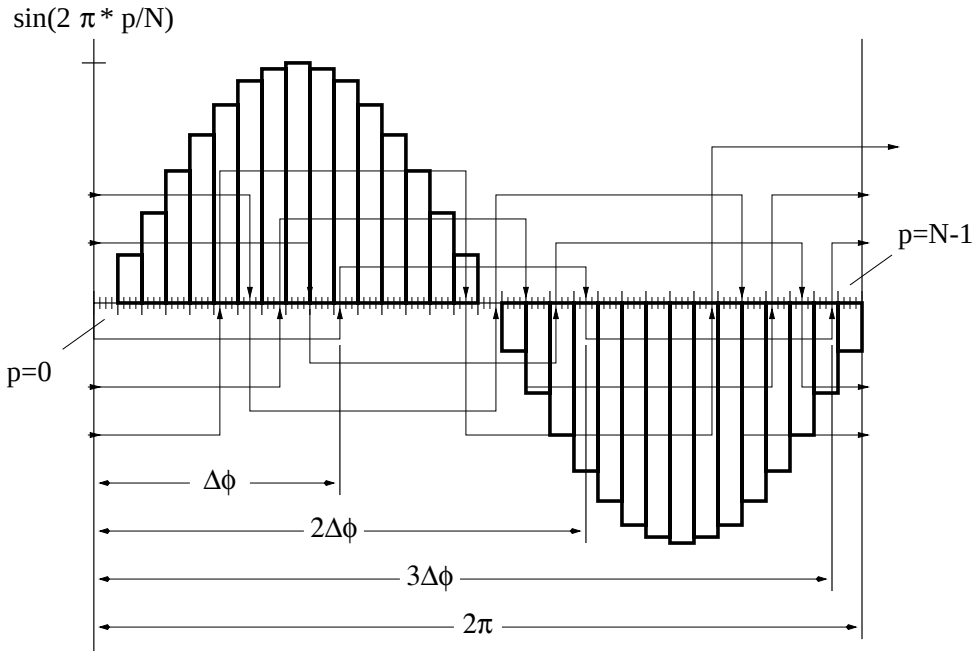


Abbildung 2.12: **DDS-Schema** *Bei der Direkten Digital Synthese (DDS) wird eine Sinus-Tabelle, die* `N` *Werte* genau einer *Sinus-Periode enthält, mit einer äquidistanten Schrittweite* $\Delta\varphi = 2\pi \cdot \nu / f_s$ *zyklisch durchlaufen. Dabei ist* f_s *die Abtastfrequenz und* ν *die Frequenz des erzeugten Sinus-Signals. Die Akkumulation der Phasendifferenz* $\Delta\varphi$ *erfolgt mit feinerer Schrittweite, als die Sinus-Tabelle Einträge hat. Die Darstellungsgenauigkeit von* $\Delta\varphi$ *bestimmt die Genauigkeit des Frequenzverhältnisses* ν / f_s.

Die Darstellungsgenauigkeit der Tabelleneinträge in der Sinus-Tabelle legt den durch Quantisierungsrauschen bestimmten Signal-Rausch-Abstand eines erzeugten Sinus-Signals fest. Ein Sinus-Signal, das mit einer Wortbreite von n Bits (bei linearer, ganzzahliger Darstellung) quantisiert ist, weist einen Signal-Rausch-Abstand von $\approx 1,8$ [dB] $+ n \cdot 6$ [dB] auf [Tietze und Schenk, 1993]. Dazu ist jedoch die Zahl der Tabelleneinträge derart zu wählen, daß der Differenzbetrag aufeinander folgender Tabelleneinträgen $|\mathtt{sin[i]} - \mathtt{sin[i+1]}| \leq 1.0$ ist, da $|\frac{d}{dx}\sin(x)| \leq 1.0$ ist. Diese Bedingung ist erfüllt, wenn die Zahl der Tabelleneinträge $\mathtt{N} > 2^n \cdot \pi$ ist, da das Seitenverhältnis einer vollen Periode ($2\pi \approx \mathtt{N}$) zur Sinus-Amplitude ($\pm 1 \approx 2^n$) gerade $2\pi/2$ beträgt. Der Phasenwinkel φ_i zum i-ten Abtastzeitpunkt wird mit Hilfe des Phasenakkumulators `A` in vorzeichenloser Zahlendarstellung gemäß

$$\mathtt{A}_i := (\mathtt{A}_{i-1} + \mathtt{C}) \bmod 2^{\mathtt{a}} \quad \text{wobei} \quad \mathtt{C} = 2^{\mathtt{a}} \cdot \frac{\nu}{f_s} \qquad \text{mit} \qquad \mathtt{A}, \mathtt{C} \in [0, \ldots, 2^{\mathtt{a}} - 1] \quad (2.41)$$

berechnet. Dabei ist `a` die Wortbreite des Phasenakkumulators. Die erreichbare Frequenzgenauigkeit des Farbträgersignals, d.h. die Genauigkeit, mit der das Frequenzverhältnis ν / f_s darstellbar ist, hängt von der Wortbreite des Phasenakkumulators und der Akkumulationskonstanten `C` ab. Die Konstante `C` repräsentiert die konstante Phasenwinkeldifferenz $\Delta\varphi$. Durch Veränderung der

Konstanten C lassen sich beliebige Signalfrequenzen einstellen, und eine Anpassung auf unterschiedliche Abtastfrequenzen f_s ist möglich. Der Sinus-Wert zum i-ten Abtastzeitpunkt ist

$$\sin_i = \texttt{sin[A}_i\texttt{((a-1) downto (a-n)) + } \Phi \texttt{]}. \tag{2.42}$$

Für die Adressierung der Sinus-Tabelle mit $\texttt{N} = 2^{\texttt{n}}$ Werten werden also nur die höchstwertigsten n Bits (*MSBs* $\approx$ *Most Significant Bits*) des Phasenakkumulators A verwendet. Die in (2.42) verwendete Schreibweise (downto) ist an die Schaltungsbeschreibungssprache VHDL [IEEE, 1994] angelehnt. Bei vorzeichenloser Darstellung von A mit a Bits ist $\texttt{C} = 2^{\texttt{a}} \cdot \Delta\varphi$. Ein Überlauf (mod $2^{\texttt{a}}$) von A bewirkt die korrekte, unendliche und periodische Fortsetzung der endlichen Sinus-Wertetabelle.

Welche Sinus-Auflösung mit welcher Phasenauflösung und Frequenzauflösung für eine vorliegende Aufgabe erforderlich ist, kann bei bestimmten Anwendungen abschließend nur durch geeignete Simulationsmethodik bestimmt werden. Die digitale Verarbeitung analoger Videosignale stellt in diesem Kontext eine besondere Herausforderung dar, die mit analytischen Methoden nicht oder nur schwer zu bewältigen ist. So konnte im Rahmen der vorliegenden Arbeit durch Simulationen zur rein digitalen Erzeugung von PAL-Videosignalen und deren Bewertung mit Hilfe von *VidTrans* durch Darstellung auf einem Farbfernseher gezeigt werden, daß die DDS geeignet ist, um eine Farbbildqualität zu erreichen, die von der analoger Videosignalquellen fast nicht zu unterschieden ist. Die für Backpropagation-Netzwerke benötigte Sigmoidfunktion (2.3) läßt sich für Schaltungsimplementierungen ebenfalls effizient annähern.

2.4 Videosignalverarbeitung

Im folgenden Abschnitt werden die Grundlagen der Videosignalverarbeitung skizziert, die für die vorliegende Arbeit von Bedeutung sind. Der Ausgangspunkt für die gemachten Untersuchungen und Implementierungen zur digitalen Verarbeitung analoger Farbvideosignale ist das Systemkonzept für eine Architektur zur Echtzeitverarbeitung von Videobildern mit neuronalen Netzwerken (s. Abschnitt 3.3, S. 72, NeNEB, Abb. 3.4, S. 73). Die rein digitale Videosignalverarbeitung ist für die Implementierung von Systems-on-a-Chip (SoC) besonders gut geeignet [Keating und Bricaud, 1998], da zur Implementierung von ausschließlich digitalen Komponenten auf die etablierte CMOS-Technologie und leistungsfähige CAD-Werkzeuge zur Realisierung digitaler Schaltungen zurückgegriffen werden kann. So sind digitale Komponenten zur Videosignalverarbeitung sogar für die Implementierung in FPGAs geeignet, wie anhand einer Hardware-Implementierung eines digitalen PAL-Encoders [Larsson, 1999b] gezeigt werden konnte. Darüber hinaus ist der Entwurf rein digitaler Komponenten mit verfügbaren CAD-Werkzeugen weniger problematisch – in Bezug auf Simulation und Simulationszeiten – als der Entwurf analoger Komponenten oder gemischt analog-digitaler Komponenten.

Aufgrund der Allgegenwärtigkeit analoger Farbfernsehgeräte kann die digitale Erzeugung von PAL-Videosignalen für Anwendungen, die Bildausgaben erfordern, wohl als eine Art Standardsignalverarbeitungsaufgabe verstanden werden. Zwar gibt es inzwischen von verschiedenen Halbleiterherstellern digitale Videosignal-Encoder und digitale Videosignal-Decoder. Die Unabhängigkeit von Herstellern digitaler Videosignal-Encoder und Videosignal-Decodern kann

jedoch durch Einsatz validierter Module (IP-Cores) anstelle von handelsüblichen Chips unter Umständen ein Redesign bei Einstellung der Produktion eines handelsüblichen Chips vermeiden helfen [Stogdill, 1999].

Die digitale Videosignalverarbeitung hat darüber hinaus die Vorteile aufzuweisen, die digitale Signalverarbeitung gegenüber analoger Verarbeitung hat. Digitale Signalverarbeitung hat grundsätzlich den Vorteil gegenüber der analogen Signalverarbeitung, daß die Verarbeitung weitgehend unabhängig von elektrischen Parametern der zur Signalverarbeitung verwendeten Komponenten ist. Darüber hinaus ist die digitale Signalverarbeitung auch unempfindlich gegenüber Änderungen elektrischer Parameter, wie etwa durch Alterung. Die Validierung von videosignalverarbeitenden Einheiten ist eine wissenschaftliche Herausforderung in Bezug auf geeignete Entwurfsmethodik. Im Rahmen der vorliegenden Arbeit wurden hierzu wissenschaftliche Beiträge auf Basis von programmierbarer Logik erarbeitet [Larsson, 1996, 1999b].

2.4.1 PAL-Videosignal

Die genaue zeitliche Struktur des PAL-Videosignals ist in Grundlagenliteratur zur Fernsehtechnik beschrieben [Kirsch, 1993; Mäusel, 1995]. Weitere Detailinformationen der analogen Videosignalstandards (NTSC, PAL, SECAM) sind in [Jack, 1996] zu finden. Da im Rahmen der vorliegenden Arbeit nur die Verarbeitung von Farbvideosignalen im PAL-Standard untersucht und implementiert wurde, werden im folgenden nur die für die vorliegende Arbeit relevanten Teile der PAL-Videosignalverarbeitung skizziert.

Das PAL-Videosignal setzt sich aus verschiedenen Signalkomponenten zusammen. Horizontalsynchronimpulse (H-SYNC) und Vertikalsynchronimpulse (V-SYNC) dienen der Synchro-

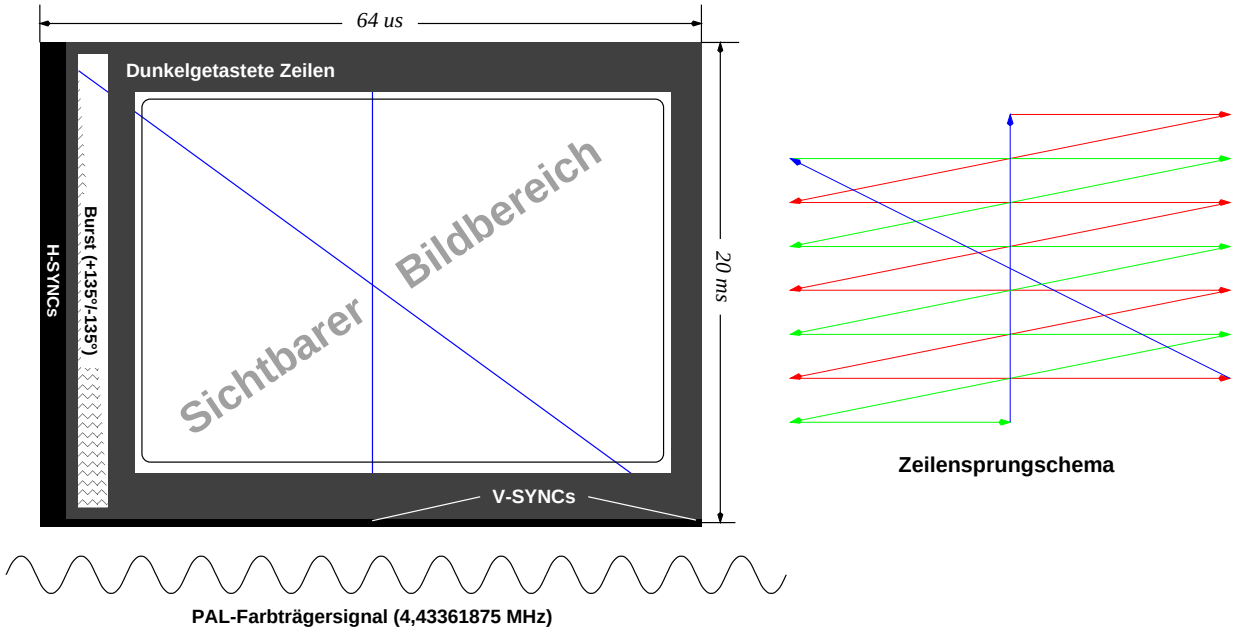


Abbildung 2.13: **BAS - Bild-Austast-Synchron-Signale** - *schematisch [Larsson, 1999b].*

nisation von Videobildquelle (Kamera) und Videobildziel (TV, Videorekorder). H-SYNC und V-SYNC werden zu Composite-Synchronimpulsen (C-SYNC) zusammengesetzt (s. Abbildung 2.14). Vor und nach dem V-SYNC werden zusätzliche H-SYNC-Impulse mit der halben H-SYNC-Impulslänge, aber doppelter H-SYNC-Frequenz eingefügt (Vortrabanten und Nachtrabanten), die der Signalverarbeitung analoger Fernsehgeräte dienen. Beim PAL-Verfahren werden 25 Bilder mit 625 Zeilen pro Sekunde übertragen. Zur Verringerung des Flimmerns, werden jedoch 50, vertikal um eine Zeile versetzte Halbbilder mit 312,5 Zeilen pro Halbbild übertragen.

Durch das Zeilensprungverfahren wird die Verringerung des wahrnehmbaren Bildflimmerns erreicht, ohne die zur Bildübertragung erforderliche Videosignalbandbreite zu erhöhen. Bereiche außerhalb des sichtbaren Bildausschnitts (s. Abbildung 2.13) werden dunkel getastet. Durch das Zeilensprungverfahren ergibt sich ein halbzeiliges Synchonisationsschema, aus dem Zeilen- und Bildsynchronisationssignale abzuleiten sind. Innerhalb der Bildzeilen liegt eine Signalstruktur vor, die primär der Zeilensynchronisation dient, aber zusätzliche, nachträglich eingefügte Signalkomponenten enthält, die für die Farbwiedergabe im PAL-Standard erforderlich sind. Die Erzeugung der Austastsignale, Synchronisationssignale (vgl. Abbildung 2.14) und verschiedener Steuersignale (Farbträgerphase, Bildspeicheradressen, Bildnummer, interne Steuersignale, etc.) läßt sich leicht durch Zählerkaskaden realisieren.

Die Helligkeit von Bildpunkten wird durch die Amplitude der als Luminanz-Signal bezeichneten PAL-Videosignalkomponente repräsentiert. Das C-SYNC-Signal wechselt zwischen zwei Zuständen, nämlich zwischen Minimal-Pegel (SYNC-Pegel) des PAL-Videosignals und

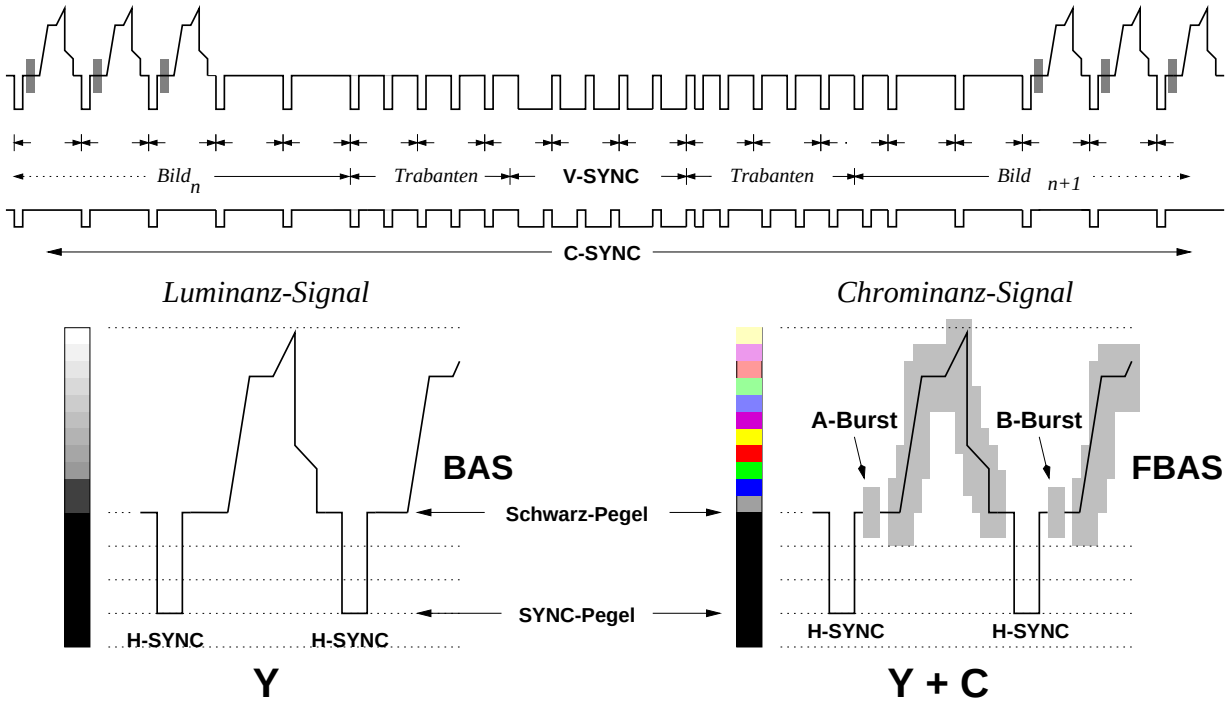


Abbildung 2.14: **FBAS - Farbe-Bild-Austast-Synchron-Signal** - *schematisch [Larsson, 1999b].*

Schwarz-Pegel. Das Luminanz-Signal wird zu dem Schwarzpegel addiert. Das sich ergebende Signalgemisch zur Übertragung von Grauwertbildern wird als BAS-Signal (BAS ≈ **B**ild-**A**ustast-**S**ynchronsignal [Kirsch, 1993]) bezeichnet. Die Erweiterung des BAS-Signals durch zusätzliche Signalkomponenten für die Farb-Übertragung wird als FBAS-Signal (FBAS ≈ Farb-BAS [Kirsch, 1993]) bezeichnet. BAS-Signal und FBAS-Signal sind in Abbildung 2.14 skizziert.

Die Farbübertragung des PAL-Standards erfolgt nicht direkt auf Basis des RGB-Farbraums, sondern auf Basis des YUV-Farbraums. Dazu werden ergänzend zur Bildpunkthelligkeit (*Luminanz*) y zwei Farbdifferenzsignale u und v übertragen. RGB-Farbraum und YUV-Farbraum sind durch lineare Transformationen ineinander überführbar [Jack, 1996].

$$\begin{array}{lcl} y & = & 0,30 \cdot r + 0,59 \cdot g + 0,11 \cdot b \\ u & = & b - y \\ v & = & r - y \end{array} \qquad \begin{array}{lcllll} r & = & y & & & +1,14 \cdot v \\ g & = & y & -0,39 \cdot u & & -0,58 \cdot v \\ b & = & y & +2,04 \cdot u & & \end{array} \tag{2.43}$$

Die beiden Farbdifferenzsignale u und v werden mittels Quadratur-Amplituden-Modulation übertragen [Kirsch, 1993; Mäusel, 1995]. Zur Vermeidung bzw. Reduzierung von Über- und Untersteuerung werden die Farbdifferenzsignale zuvor gemäß $u := u \cdot 0,49$ und $v := v \cdot 0,88$ reduziert. Die Quadratur-Amplituden-Modulation

$$c(t) = u(t) \cdot \cos(2\pi \cdot f_{PAL} \cdot t) \pm v(t) \sin(2\pi \cdot f_{PAL} \cdot t) \tag{2.44}$$

liefert das *Chrominanz*-Signal c. Dabei ist $f_{PAL} = 4,43361875\ MHz$ die Frequenz des PAL-Farbträgersignals. Das Vorzeichen des Farbdifferenzsignals v wird beim PAL-Verfahren von Zeile zu Zeile gewechselt, wodurch das PAL-Verfahren weniger anfällig gegenüber phasenfehlerbedingten Farbfehlern als das verwandte NTSC-Verfahren ist [Kirsch, 1993; Mäusel, 1995]. Zur Rekonstruktion der Farbdifferenzsignale u und v in einem PAL-Decoder ist die genaue Kenntnis der Phase und des Vorzeichens von v des Farbträgersignals im PAL-Encoder erforderlich. Im PAL-Encoder wird dann das Chrominanz-Signal $c(t)$ mit $\cos(\omega \cdot t)$ und mit $\sin(\omega \cdot t)$, wobei $\omega = 2\pi \cdot f_{PAL}$, multipliziert.

$$\begin{array}{lcll} d(t) & = & c(t) \cdot \cos(\omega \cdot t) & = \frac{1}{2}u(t) + \frac{1}{2}u(t) \cdot \cos(2\omega \cdot t) + \frac{1}{2}v(t) \cdot \sin(2\omega \cdot t) \\ e(t) & = & c(t) \cdot \sin(\omega \cdot t) & = \frac{1}{2}v(t) + \frac{1}{2}u(t) \cdot \sin(2\omega \cdot t) - \frac{1}{2}v(t) \cdot \cos(2\omega \cdot t) \end{array} \tag{2.45}$$

Durch Tiefpaßfilterung werden die Terme in (2.45) mit $\cos(2\omega \cdot t)$ und $\sin(2\omega \cdot t)$ im PAL-Decoder unterdrückt, so daß die Terme $\frac{1}{2}u(t - T_G)$ und $\frac{1}{2}v(t - T_G)$ von (2.45) übrig bleiben. Das Luminanz-Signal y wird um die Gruppenlaufzeit T_G der Tiefpaßfilter zu $y(t - T_G)$ verzögert. Neben der Kenntnis der Phase des PAL-Farbträgers ist die Kenntnis des zeilenweise wechselnden Vorzeichens ($\pm$) des Farbdifferenzsignals v in (2.44) notwendig. Dazu wird beim PAL-Verfahren die Phase des Burst-Signals (s. Abbildung 2.15) von Zeile zu Zeile zwischen $+135^o$ (sog. A-Burst) und -135^o (sog. B-Burst) hin und her geschaltet ($180^o \pm 45^o$). Die Phasenwinkel sind relativ zur Phase des Farbträgersignals (0^o). Die Phase des Farbträgers wird für die QAM-Modulation (2.44) jedoch nicht hin und her geschaltet. Der Farbträgersignalgenerator muß daher

ein sinusförmiges Farbträgersignal $f(t)$ mit vier verschiedenen Phasen zur Verfügung stellen. Je nach Abtastzeitintervall (A-Burst, B-Burst, Bildbereich ($+u$, $\pm v$)) wird ein sinusförmiges Farbträgersignal

$$f(t) \quad = \quad \begin{cases} \cos(\omega t - 0^o) & : \text{u-Träger } (u(t) \cdot \cos(\omega t)) \\ \cos(\omega t - (+90^o)) & : \text{v-Träger } (v(t) \cdot \sin(\omega t)) \\ \cos(\omega t - (+135^o)) & : \text{A-Burst-Träger} \\ \cos(\omega t - (-135^o)) & : \text{B-Burst-Träger} \end{cases} \tag{2.46}$$

ausgewählt und nachgeschalteten Signalverarbeitungseinheiten bereitgestellt. Das PAL-Farbträgersignal (e^{iwt}) muß unabhängig vom Zeitraster der Synchronisationssignalen zur Verfügung gestellt werden. Das Farbträgersignal (2.46) wird mit der für die unterschiedlichen Zeitintervallen (A-Burst, B-Burst, sichtbarer Bildbereich) zugeordneten Phase ausgewählt und dem BAS-Signal hinzugefügt, wie in Abbildung 2.15 skizziert.

2.4.2 PAL-Encoder

Die Aufgabe eines PAL-Encoders ist die Konvertierung von Farbbildinformation in ein Signalgemisch, das zur Darstellung dieser Farbbildinformation auf analogen PAL-Farbfernsehern geeignet ist. Ein PAL-Encoder konvertiert einen Bilddatenstrom

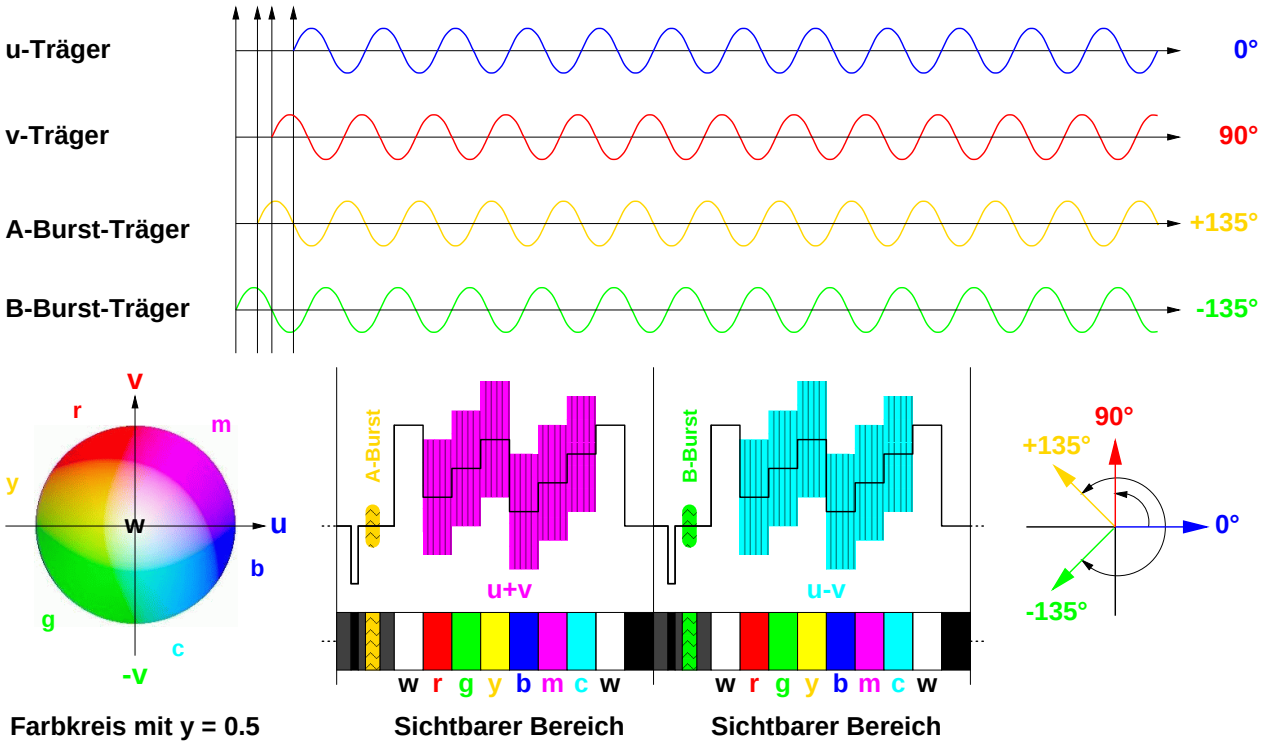


Abbildung 2.15: **PAL-Videosignal - Phasenbeziehungen** - *schematisch [Larsson, 1999b].*

$$(r(t), g(t), b(t)) \longrightarrow (y(t), u(t), v(t)) \longrightarrow \text{fbas}(t). \tag{2.47}$$

Wie eingangs schon erwähnt, wurden im Rahmen der vorliegenden Arbeit Untersuchungen zur digitalen Verarbeitung analoger Farbvideosignale im Kontext eines Systemkonzepts zur Echtzeitverarbeitung von Videobildern mit neuronalen Netzwerken mit dem Ziel durchgeführt, wissenschaftliche Grundlagen für die Implementierung solcher Systeme als rein digitale Systeme – möglichst auf einem einzigen Chip (System-on-a-Chip (SoC)) – zu erarbeiten. So wurden bei der Konzipierung und Software-Implementierung von digitalen PAL-Encodern und digitalen PAL-Decodern für spätere, effiziente Hardware-Implementierung technische Randbedingungen

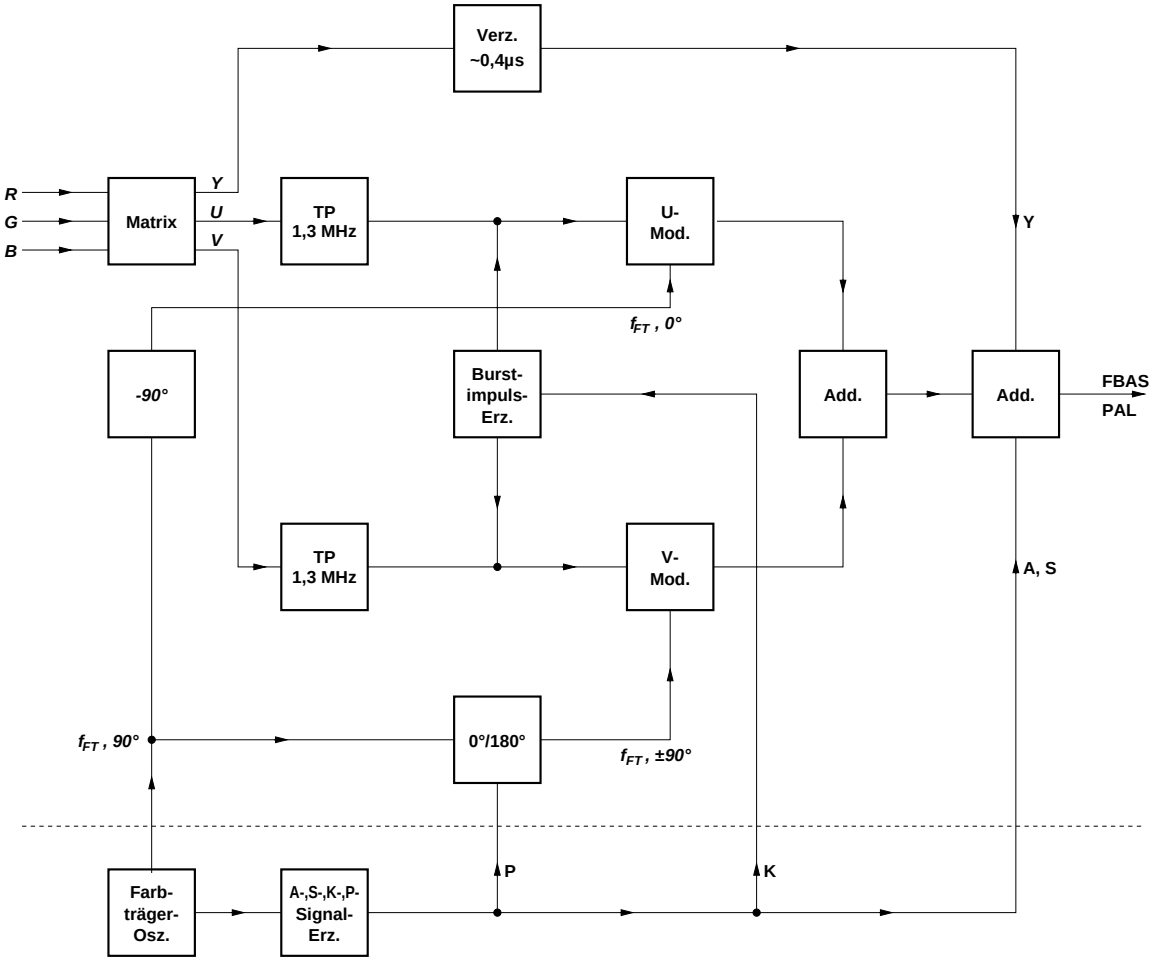


Abbildung 2.16: **PAL-Videosignal-Encoder - Blockschaltbild** *nach [Mäusel, 1995]. Burst-Auftastimpuls (K-Impuls (Burst-Gate)), PAL-Kennimpuls (P-Impuls (Vorzeichen von v)), Austast-Impuls (A-Impuls) und Synchonisations-Impuls (S-Impuls) steuern die verschiedenen signalerzeugenden Komponenten eines PAL-Encoders.*

des Entwurfs digitaler Schaltungen berücksichtigt. Die im Rahmen der betreuten Studienarbeit [Jürgens, 1996] und der betreuten Diplomarbeit [Jürgens, 1999] implementierten PAL-Encoder und PAL-Decoder weichen daher in wesentlichen Details von in der Literatur [Kirsch, 1993; Mäusel, 1995] beschriebenen ab. Darüber hinaus bestehen weitere Unterschiede bezüglich einiger Implementierungsdetails zwischen den im Rahmen der oben genannten Arbeiten und begleitend zu diesen Arbeiten implementiertem PAL-Encoder und PAL-Dekoder. Daher werden an dieser Stelle die wesentlichen Merkmale von PAL-Encoder und PAL-Dekoder, wie in der Literatur beschrieben, im folgenden zusammenfassend beschrieben.

Um Übersteuerungen des PAL-Videosignals zu vermeiden, werden die reduzierten Farbdifferenzsignale $u \cdot 0,49$ und $v \cdot 0,88$ anstelle der Farbdifferenzsignale u und v übertragen. Im Empfänger (TV) werden die Farbdifferenzsignale durch geeignete Verstärkung aus den empfangenen, reduzierten Farbdifferenzsignalen rekonstruiert. Die beiden reduzierten Farbdifferenzsignale werden vor der Modulation auf den Farbträger durch Tiefpaßfilter (TP) auf $1,3MHz$ bandbegrenzt. Das Vorzeichen des Signals v wird von Zeile zu Zeile umgepolt. Das Chrominanz-Signal wird durch Addition der auf den Farbträger aufmodulierten (reduzierten) Farbdifferenzsignale erzeugt. Das Luminanz-Signal y wird um die Gruppenlaufzeit der Tiefpaßfilter verzögert, um einen Versatz zwischen Graubild ($y(t)$) und Farbbild ($u(t)$, $v(t)$) zu vermeiden. Dem resultierenden Luminanz-Chrominanz-Signal werden das Composite-Synchronisationssignal (C-SYNC) und die, von Zeile zu Zeile in der Phase ($\pm 135^o \Leftrightarrow 180 \pm 45^o$) alternierenden, Burst-Signale zugemischt. Die vier verschiedenen Phasenlagen des Farbträgersignals (vgl. Gleichung (2.46)) werden durch Phasenschieber [Tietze und Schenk, 1993] aus dem zentral erzeugten Farbträgersignal gewonnen. In Abbildung 2.16 ist ein Blockschaltbild eines (analogen) PAL-Encoders aus [Mäusel, 1995] dargestellt.

Die Implementierung eines digitalen PAL-Encoders könnte grundsätzlich durch Nachbildung der analogen, signalverarbeitenden Komponenten eines PAL-Encoders realisiert werden. Die analoge Architektur ist jedoch zur direkten Umsetzung in eine funktional gleichartige digitale Architektur weniger gut geeignet. So wurde in [Larsson, 1999b] auf Tiefpässe zur Bandbreitenbegrenzung der reduzierten Farbdifferenzsignale verzichtet, da für die Implementierung (schneller) digitaler Filter viele Komponenten (Gatter) erforderlich sind. Weiter wurde der QAM-Modulator durch eine MAC-Einheit realisiert und nicht in Form zweier separater Modulatoren für die beiden Farbdifferenzsignale, weil dafür zwei getrennte Multiplizierer erforderlich gewesen wären – für die MAC-Einheit ist jedoch nur ein einziger Multiplizierer erforderlich. Statt Phasenschiebern wird die zu einem bestimmten Verarbeitungstakt benötigte Phase des implementierten DDS-Oszillators schnell, d.h. von einem (30 MHz) Takt zum nächsten Takt, umgetastet.

Bei der Implementierung des digitalen PAL-Encoders [Larsson, 1999b] wurde also eine digitale Architektur realisiert, die sich in wesentlichen Komponenten deutlich von einer mit analogen Komponenten realisierten Architektur unterscheidet, um eine effiziente Hardware-Implementierung zu erreichen. Dadurch ergibt sich ein etwas anderes Verhalten. Wie sich Implementierungsdetails auf der Systemebene auf die Funktionsweise und auf die erreichte Bildqualität auswirken, konnte durch Software-Implementierung vor der Hardware-Implementierung des digitalen PAL-Encoders [Larsson, 1999b] mit *VidTrans* [Larsson, 1996] evaluiert werden.

2.4.3 PAL-Decoder

Die Aufgabe eines PAL-Decoders ist die Extraktion von Farbbildinformation aus einem Signalgemisch, das von einer analogen Farbbildquelle als PAL-Videosignal geliefert wird. Ein PAL-Decoder konvertiert ein Signal

$$\text{fbas}(t) \longrightarrow (y(t), u(t), v(t)) \longrightarrow (r(t), g(t), b(t)). \tag{2.48}$$

Zur Demodulation der QAM-modulierten Signale u und v ist die Kenntnis der Frequenz *und* der Phase des zur QAM-Modulation verwendeten Farbträgers erforderlich. Darüber hinaus ist beim PAL-Verfahren das Vorzeichen des Signals v erforderlich. Nach der Abtrennung des Composite-Synchronsignals vom Bildsignal $(y + v)$ stehen diese Signale zur weiteren Verarbeitung zur Verfügung. Aus dem Composite-Synchronsignal werden das Horizontal-Synchronisationssignal (H-SYNC) und das Vertikal-Synchronisationssignal (V-SYNC) gewonnen. Durch diese Signale wird die Bildtopologie von Bildquelle und Bildziel synchronisiert. Ein Blockschaltbild eines analogen PAL-Decoders ist in Abbildung 2.17 dargestellt.

Der Burst wird in einem festgelegten Zeitfenster relativ zur fallenden Flanke des Horizontal-Synchronisationssignals übertragen. Die Burst-Signale aus diesem Zeitfenster werden auf den Phasendiskriminator gegeben. Der Phasendiskriminator steuert einen Farbträgeroszillator. Pha-

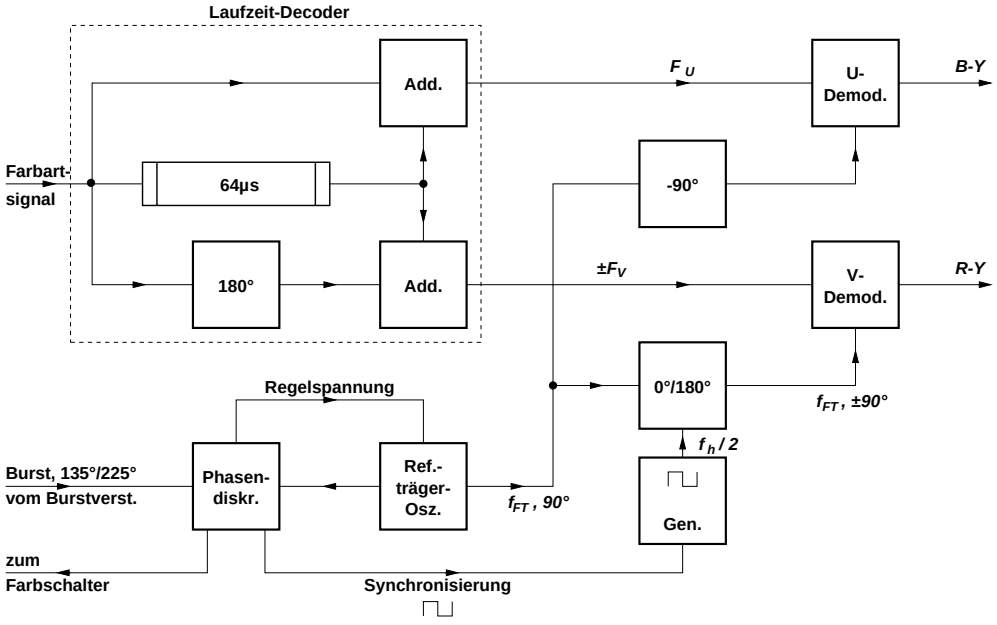


Abbildung 2.17: **PAL-Videosignal-Decoder - Blockschaltbild** *nach [Mäusel, 1995]. Wesentliches Merkmal des PAL-Encoders ist der Laufzeit-Decoder zur Kompensation von Phasenfehlern des Chrominanz-Signals, die auf dem Übertragungsweg auftreten können. Die Einheit zur Erkennung der Burst-Phase (Phasendiskriminator) steuert auch den sogenannten Farbschalter, der die Farbdarstellung beim Farbfernseher abschaltet, wenn die Burst-Signale nicht in erforderlicher Weise erkannt werden, um Farbrauschen bei Schwarz-Weiß-Videosignalen zu vermeiden.*

sendiskriminator und steuerbarer Farbträgeroszillator bilden einen Phasenregelkeis (PLL). Alternativ kann der Burst auch verwendet werden, um einen Quarz anzustoßen [Limann, 1978]. Die QAM-Demodulation erfolgt, wie im Abschnitt 2.3.2 beschrieben. Aus der Phase des Burst-Signals, relativ zum rekonstruierten Farbträgersignal, wird das Vorzeichen des Signals v bestimmt.

Wesentlicher Bestandteil eines PAL-Decoders ist der *Laufzeit-Decoder* [Mäusel, 1995]. Durch den Laufzeit-Decoder wird eine geringere Empfindlichkeit des PAL-Encoders gegenüber Phasenfehlern des Chrominanz-Signals, wie sie auf dem Signalübertragungsweg auftreten können, erreicht [Kirsch, 1993; Morgenstern, 1994; Mäusel, 1995]. Der Laufzeit-Decoder erfordert die Verzögerung um eine sehr genaue Zeitdifferenz (*genau* 283,5 oder 284 Perioden der Farbträgerfrequenz, entsprechend $63,943$ μs [Morgenstern, 1994] oder $64,056$ μs [Mäusel, 1995]) durch Speicherung einer vollständigen Bildzeile. Ohne den Laufzeit-Decoder entspricht ein PAL-Encoder in der Funktionsweise eher einem NTSC-Decoder. Der Verzicht auf den PAL-Laufzeit-Decoder führt zu einer höheren Empfindlichkeit gegenüber Phasenfehlern des Chrominanz-Signals wie bei NTSC-Encoder. Aus Sicht des Entwurfs digitaler Schaltungen ist der Verzicht auf den Laufzeit-Decoder jedoch von Vorteil. Zum einen kann dadurch auf ein (On-Chip-)RAM verzichtet werden, zum anderen ergibt sich ein Freiheitsgrad bezüglich der Wahl der Taktfrequenz eines digitalen Entwurfs. Eine Signalverzögerung um ein nichtganzzahliges Vielfaches einer Abtastfrequenz bedeutet einen zusätzlichen Schaltungsaufwand, wie etwa die Erzeugung eines Signals mit einer geeigneten Abtastfrequenz oder Interpolationen von Zwischenwerten bei Abtastung mit nichtganzzahligem Abtastfrequenzverhältnis.

In Hinblick auf effiziente Hardware-Implementierung wurden bei der Software-Implementierung digitaler PAL-Decoder Signalverarbeitungskomponenten simuliert, die sich wesentlich von mit analogen Komponenten realisierten PAL-Decodern unterscheiden. Die Software-Implementierungen der PAL-Decoder wurden auf Basis von analogen PAL-Videosignalen realer Signalquellen (Video-Kamera, terrestrisch empfangene Videosignale, VHS-Video-Rekorder) validiert, die mit Hilfe von *VidTrans* [Larsson, 1996] aufgenommen wurden. Bei Farbbildern, die aus mit Software-PAL-Encodern erzeugten PAL-Videosignalen extrahiert wurden, sind kaum Unterschiede zwischen den Quellfarbbildern und den extrahierten Farbbildern wahrnehmbar. So ergab sich die Rechtfertigung der gemachten Vereinfachungen durch Beurteilung der Simulationsergebnisse auf der Systemebene.

2.5 Digitaler Schaltungsentwurf

Die Umsetzung vieler rechenintensiver Algorithmen zur Bearbeitung von Aufgaben in Echtzeit wird häufig erst durch die Implementierung mit geeigneter Hardware ermöglicht. Durch technologische Fortschritte können ständig neue Anwendungsbereiche erschlossen werden, die in der nahen Vergangenheit nicht umsetzbar gewesen wären.

Die Hardware-Implementierung eines bestimmten Algorithmus ist meist nur dann effizient möglich, wenn der zu implementierende Algorithmus numerische Näherungen zuläßt. Ist Gleitkommaarithmetik (mit hoher Genauigkeit) für einen Algorithmus zwingend notwendig, so ist eine Software-Implementierung auf einer Standardplattform (mit einem kommerziellen

Prozessor) vorzuziehen. Ebenso sind Algorithmen, die in hohem Maße die Verwendung dynamischer Datenstrukturen erfordern, in der Regel für Hardware-Implementierungen ungeeignet. Eine Schaltungsimplementierung erfordert immer, einen geeigneten Kompromiß zwischen algorithmischen, numerischen und durch die Hardware bestimmten Randbedingungen zu finden. Das gilt besonders für neuronale Algorithmen [Moerland und Fiesler, 1996].

Die exponentielle Steigerung der Verarbeitungsgeschwindigkeit von Prozessoren lag in den siebziger Jahren zwischen 25% bis 30% pro Jahr und liegt jetzt – in den neunziger Jahren – bei etwa 35% pro Jahr [Hennessy und Patterson, 1996]. Das bedeutet zum einem, daß Algorithmen, die heute in Hardware implementiert werden, in der Zukunft in Software realisiert werden können und dabei die gleiche Verarbeitungsgeschwindigkeit erreichen werden. Beispiele dafür, aus Sicht der Vergangenheit in die Gegenwart, sind etwa digitale Audiosignalverarbeitung (Software-Synthesizer) oder das Abspielen von Videosequenzen (z.B. MS Video for Windows). Anders ausgedrückt bedeutet das jedoch, daß durch eine Hardware-Implementierung unter Umständen ein Technologievorsprung von einigen Jahren gegenüber einer funktional gleichwertigen Software-Implementierung erreicht werden kann. Aber auch wenn eine bestimmte Aufgabe durch eine Software-Implementierung lösbar ist, so ist eine Hardware-Implementierung dann trotzdem sinnvoll, wenn damit die CPU eines Rechners von rechenzeitintensiven Aufgaben entlastet werden kann. Beispiele dafür sind DMA-Controller ($\approx$ *Direct Memory Access*), die es schon in den achtziger Jahren für die damals populäre Z80-Prozessor-Familie als ergänzende Bausteine gab [Zilog, 1981]. Ein Chip zur Messung von elektrischer Effektivleistung, RMS (*Root Mean Square*) von Spannung und Strom [Tietze und Schenk, 1993] und anderer Größen, der am Arbeitsbereich "Grundlagen der Technischen Informatik" der Universität Hamburg entwickelt wurde [Grupe und Rauscher, 1991], ist ein Beispiel für die Hardware-Implementierung von Algorithmen, die zwar mittlerweile ebenso gut als Software zur Verarbeitung in Echtzeit realisierbar sind, jedoch ermöglicht erst die Realisierung als Mikrochip beispielsweise die Integration in handliche, elektrische Geräte zur Leistungsüberwachung.

Bei einer CPU-Performanzsteigerung von 35% pro Jahr ergibt sich nach a Jahren eine resultierende Performanzsteigerung von $1,35^a$, was also eine Verzehnfachung nach $\ln(10)/\ln(1,35) \approx 8$ Jahren, eine Verhundertfachung nach $\ln(100)/\ln(1,35) \approx 15$ Jahren, eine Vertausendfachung nach $\ln(1000)/\ln(1,35) \approx 23$ Jahren bedeutet. Eine um einen Faktor zehn bis hundert größere Verarbeitungsgeschwindigkeit einer Hardware-Implementierung gegenüber einer Software-Implementierung ist bei bestimmten Algorithmen durchaus möglich – in Jahren ausgedrückt – ein Technologievorsprung von einem Jahrzehnt.

Integrierte Schaltungen

Eine Hardware-Implementierung kann etwa als ASIC ($\approx$ *Application Specific Integrated Circuit*) oder mit Hilfe von programmierbaren Bausteinen (EPLDs $\approx$ *Electrically Programmable Logic Device* oder FPGAs $\approx$ *Field Programmable Gate Array*) erfolgen. Häufig wird die Bezeichnung FPGA auch stellvertretend für programmierbare Bausteine verwendet. Dagegen wird die Abkürzung (PLD $\approx$ *Programmable Logic Device*) als Oberbegriff für programmierbare Logikschaltungen vergleichsweise selten verwendet. Darüber hinaus gibt es herstellerspezifische Produktbezeichnungen für programmierbare Logikbausteine.

Die Wahl einer Taxonomie integrierter Schaltungen spiegelt jeweils eine bestimmte Perspektive wider und kann nach unterschiedlichen Parametern erfolgen. Im Rahmen der vorliegenden Arbeit steht ASIC stellvertretend für Bausteine, deren Verschaltungsstruktur vom Hersteller des betreffenden Bausteins durch Fertigungsschritte vorgenommen wird und nicht durch elektronische Programmierung über von außen zugängliche Anschlüsse erfolgen kann. Im Gegensatz dazu werden als programmierbare Bausteine solche integrierten Schaltungen bezeichnet, deren interne, anwendungsspezifische Verschaltungsstruktur nicht vom Hersteller der betreffenden Bausteine vorgenommen werden muß, sondern über die von außen zugänglichen Anschlüsse elektronisch programmiert werden kann.

Die endgültige Wahl der Zieltechnologie (ASIC, EPLD, FPGA, ...) ist bei Spezifikation in einer geeigneten Schaltungsbeschreibungssprache wie VHDL (≈ *V*HSIC *HDL* ≈ *Very High Speed Integrated Circuit Hardware Description Language*) [IEEE, 1994] dabei zunächst nicht notwendigerweise festgelegt. Daher kommt der verwendeten Zieltechnologie bei der Spezifikation einer Schaltung zunächst nur eine untergeordnete Rolle zu, wenn man von der *Testbarkeit* einer integrierten Schaltung zunächst einmal absieht. Unterschiede zeigen sich im wesentlichen bei der Integrationsdichte, Verarbeitungsgeschwindigkeit, Bausteinkosten und der Zeit, die benötigt wird, um eine anwendungsspezifische Verschaltung vorzunehmen. Vorteile einer Hardware-Implementierung gegenüber einer Software-Implementierung auf einer Standardplattform sind:

- höhere erreichbare Verarbeitungsgeschwindigkeit
- geringere Größe und geringeres Gewicht
- Geheimhaltung und Kopierschutz
- geringere Leistungsaufnahme
- geringere Stückkosten bei Massenfertigung

So kann die Implementierung eines Algorithmus' als schnelle Spezialschaltung zum einen die Option einer Echtzeitverarbeitung erschließen, zum anderen aber durch geringe Größe und geringen Stromverbrauch den Einsatz in mobilen Systemen unter Umständen erstmalig ermöglichen. So stellen integrierte Schaltungen die Schlüsseltechnologie für Digital Audio und Digital Video dar [Luther, 1997]. Die Implementierung von Algorithmen in Form von Schaltungen ist jedoch im allgemeinen zeitaufwendiger als eine entsprechende Software-Lösung. Bei industriellen Entwürfen mit sehr großen Stückzahlen spielen die durch ASIC-Fertigung erreichbaren geringeren Kosten eine entscheidende Rolle.

Schaltungsspezifikation

In der Vergangenheit erfolgte die Spezifikation einer (digitalen) Schaltung überwiegend durch graphische Eingabe in Form von Schaltplänen (*Schematic Entry*). Die Verwendung von Schaltungsbeschreibungssprachen wie etwa VHDL oder Verilog erlaubt, verglichen mit der graphischen Eingabe von Schaltplänen, die Spezifikation von Schaltungen textuell – als ASCII-Datei – auf einer höheren Abstraktionsebene, in einem menschenlesbaren Dateiformat. Dadurch *kann* –

strukturiertes Entwurfsvorgehen vorausgesetzt – eine bessere Lesbarkeit von entwickelten Modulen sowie deren Wiederverwendbarkeit (*re-usability*) erreicht werden [Keating und Bricaud, 1998]. Die Wiederverwendbarkeit wird besonders durch die Standardisierung durch IEEE (*Institute of Electrical and Electronics Engineers*) unterstützt, weil dadurch prinzipiell Unabhängigkeit von herstellerspezifischen Entwicklungswerkzeugen erreicht werden kann. Dem steht jedoch entgegen, daß nicht alle Entwurfswerkzeuge die festgelegten Standards in vollem Umfang unterstützen. Das gilt für die Syntax, aber insbesondere für Restriktionen bezüglich der zur Logiksynthese verwendeten Entwurfswerkzeuge.

Einer Schaltungsentwicklung geht in der Regel eine Software-Implementierung voraus, um zunächst einmal eine ausführbare Spezifikation des zu implementierenden Algorithmus zur Verfügung zu haben, die eine funktionale Validierung des implementierten Algorithmus ermöglicht und später auch die Validierung der Hardware-Implementierung erlaubt. Ferner dient eine ausführbare Spezifikation dazu, das Verhalten des betreffenden Algorithmus zu untersuchen, da Software- und Hardware-Implementierungen stark unterschiedliche Randbedingungen aufweisen, die es vor der Schaltungsimplementierung zu evaluieren gilt. Zum Beispiel stehen für eine Software-Implementierung auf einer Standardplattform benötigte mathematische Grundfunktionen im allgemeinen zur Verfügung, wogegen solche Grundfunktionen für eine Schaltungsimplementierung bei Bedarf erst erstellt werden müssen. Darüber hinaus spielt die Simulationsgeschwindigkeit bei der Evaluation von Algorithmen und bei der Festlegung von Parametern, etwa um notwendige Datenwortbreiten für die Verarbeitung von Zwischenergebnissen zu evaluieren, eine entscheidende Rolle. So kann die Simulationsgeschwindigkeit einer ausführbaren Spezifikation - beispielsweise in C programmiert - durchaus hundertfach größer sein als eine funktional äquivalente (synthetisierbare) Verhaltensbeschreibung – beispielsweise in VHDL erstellt –, wie in Abschnitt 4.3.2 im Zusammenhang mit der exemplarischen Simulation eines FIR-Filters gezeigt wird. Eine Schaltungsimplementierung erlaubt unter Umständen eine stark problemadaptierte Verwendung von Pipeline-Registern und damit eine Steigerung der Verarbeitungsgeschwindigkeit durch Parallelisierung gegenüber einer Software-Implementierung auf Standardprozessoren [Bayoumi, 1994]. Die diskrete Faltung (2.23) ist ein Beispiel für einen Algorithmus, welcher besonders gut zur Hardware-Implementierung als parallele Architektur geeignet ist (s. Abbildung 2.8, S. 29).

Abstraktion, Hierarchie, Regularität, Modularität und Lokalität

Der Schaltungsentwurf impliziert die Verwendung verschiedener Abstraktionsebenen [Weste und Eshraghian, 1993]. Hierarchisches Entwurfsvorgehen ist zur Handhabung komplexer Schaltungsentwürfe erforderlich. Die Einführung von Hierarchie dient primär der funktionalen Partitionierung von Schaltungsentwürfen und ermöglicht letztendlich erst die Bearbeitung einer komplexen Entwurfsaufgabe durch ein Team. Die Einführung von Hierarchie allein reduziert nicht notwendigerweise die Komplexität eines Entwurfs, sondern erst im Zusammenspiel mit regulären Strukturen und der Einführung von Modularität, wobei durch definierte Schnittstellen von Modulen Lokalität erreicht werden kann, so daß Module unabhängig von anderen Modulen entworfen werden können [Weste und Eshraghian, 1993]. Schaltungsbeschreibungen unterschiedlicher Abstraktionsebenen stellen Repräsentationen einer Schaltung mit unterschiedlichem

Detaillierungsgrad dar [Gajski, 1988]. Ebenen niedriger Abstraktion weisen einen höheren Detaillierungsgrad als höhere Abstraktionsebenen auf [Weste und Eshraghian, 1993]. Mit dem in Richtung zu niedrigeren Abstraktionsebenen einhergehenden höheren Detaillierungsgrad geht auch ein unter Umständen um Größenordnungen wachsender Simulationszeitbedarf einher.

Entwurfsebenen

Der Entwurfsprozeß kann in unterschiedliche Entwurfsebenen eingeteilt werden, die sich an den für die verschiedenen Ebenen typischen funktionalen Elementen mit ihren spezifischen Eigenschaften orientieren. In [Rauscher, 1996] ist folgende Einteilung dargestellt:

- Architekturebene
- Algorithmenebene
- Registertransferebene
- (Ebene der funktionalen Blöcke)
- Logikebene
- Schalterebene
- Transistorebene
- Physikalische Ebene

Dieses Schema kommt als graphische Metapher des interaktiven Charakters des Entwurfsvorgehens im *Jo-Jo-Modell* zum Ausdruck. Das bekannte *Y-Diagramm* nach [Gajski, 1988] ist eine

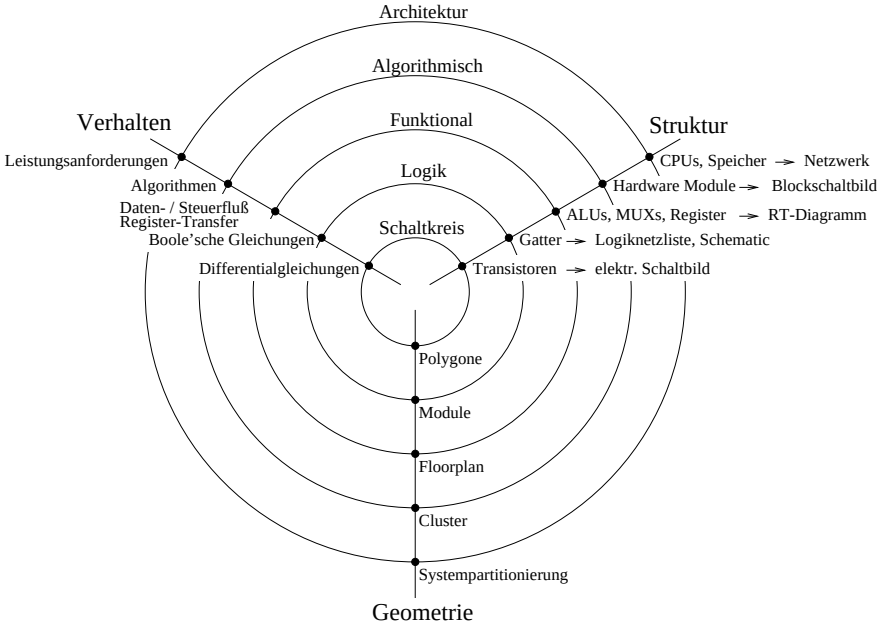


Abbildung 2.18: **Y-Diagramm** *nach [Gajski, 1988]*

zweidimensionale, radiale Darstellungsform mit Schichten (s. Abb 2.18). Über den Entwurfsebenen bzw. außerhalb des Y-Diagramms steht die *Aufgabenspezifikation* als eine Art Metaebene. Eine Aufgabenspezifikation kann in ganz unterschiedlicher Weise vorliegen, meist jedoch als Mischung von Spezifikationen verschiedener Teilaufgaben in Kombination mit Verständnis und Intuition der Entwerfer.

Logiksynthese

Die Transformation einer in einer Schaltungsbeschreibungssprache [Bhasker, 1996, 1997] formulierten Schaltungsbeschreibung einer höheren Abstraktionsebene – etwa eine funktionale Verhaltensbeschreibung – in eine Schaltungsbeschreibung niedrigerer Abstraktion – etwa eine Gatternetzliste – ist Aufgabe der *Schaltungssynthese*, und leistungsfähige Synthesewerkzeuge ermöglichen hochgradig automatische Logiksynthese [Kurup und Abbasi, 1995]. Verfügbare Synthesewerkzeuge unterscheiden sich jedoch sehr stark bezüglich der von der Logiksynthese unterstützten Sprachkonstrukte. Der Begriff Logiksynthese ist dabei eigentlich etwas irreführend: Die Spezifikation einer Schaltung, also der (funktionale) Entwurf einer Schaltung, ist nach wie vor Aufgabe des Entwerfers. Die Kunst der Schaltungsbeschreibung besteht darin, diese so zu formulieren, daß automatische Werkzeuge die Schaltungssynthese effektiv umzusetzen vermögen. Schwierig ist dabei auch der Umstand, daß mehrere semantisch gleiche Beschreibungen einer bestimmten Schaltung zu sehr unterschiedlichen Ergebnissen bezüglich des Realisierungsaufwands oder der Verarbeitungsgeschwindigkeit führen können [Bhasker, 1996; Pellerin und Taylor, 1997]. Darüber hinaus unterstützen auch nicht alle verfügbaren Synthesewerkzeuge die gleichen Sprachkonstrukte. So wird beispielsweise die Synthese von mehrdimensionalen Signalen (z.B. `type array2d is array (natural range 0 to 31) of std_logic_vector (7 downto 0);`) derzeit nicht von allen VHDL-Synthesewerkzeugen unterstützt [Pellerin und Taylor, 1997].

Die verschiedenen Abstraktionsebenen sind durch charakteristische funktionale Elemente bestimmt. Die Logiksynthese, d.h. die Umsetzung von einer in einer Schaltungsbeschreibungssprache wie etwa VHDL formulierten Verhaltensbeschreibung mit einer bestimmten Semantik auf eine Struktur, die das gewünschte Verhalten zeigt, ist aufgrund leistungsfähiger CAD-Entwurfswerkzeuge mittlerweile hochgradig automatisiert, vergleichbar mit der Compilation eines hochsprachlich formulierten Algorithmus in ausführbaren Maschinencode. VHDL erlaubt jedoch neben der textuellen Beschreibung von Zustandsgraphen endlicher Automaten auch direkt die Formulierung nebenläufiger, interagierender Prozesse. Daher verlagert sich auch der Schwerpunkt des digitalen Schaltungsentwurfs in zunehmenden Maße von strukturbestimmten Ebenen (Registertransferebene, Gatterebene, ...) auf die algorithmische und verhaltensbestimmte Abstraktionsebene. Damit geht auch ein deutlich erhöhter, vom einzelnen Entwerfer zu handhabender, Umfang digitaler Schaltungsentwürfe einher [Keating und Bricaud, 1998]. Diese wachsende Komplexität der Schaltungsentwürfe und deren Integration in Systeme erfordert systemnahe Methoden beim Schaltungsentwurf und bei der Schaltungssimulation zur Validierung.

2.5.1 Schaltungsimplementierung von Algorithmen

Die anwendungsspezifische Schaltungsimplementierung von Algorithmen erfordert im allgemeinen numerische Vereinfachungen, wenn eine Steigerung der Verarbeitungsgeschwindigkeit mit vertretbarem Schaltungsaufwand gegenüber einer Software-Implementierung auf einer Standardplattform erreicht werden soll. Mit numerischen Vereinfachungen sind Störungen verbunden, deren Auswirkung auf das Verhalten eines übergeordneten Systems es zu bewerten gilt, um notwendige Entwurfsentscheidungen treffen zu können.

Zahlendarstellung

Gleitkommadarstellung ist für bestimmte numerische Berechnungen, wie sie bei neuronalen Netzwerken und Signalverarbeitungsalgorithmen benötigt werden, bezüglich der Verarbeitungsgeschwindigkeit auf PCs oder Workstations mittlerweile ebenso gut oder sogar besser geeignet als Festkommadarstellung, da die Prozessoren dieser Plattformen über leistungsfähige Gleitkommarechenwerke verfügen, so daß Gleitkommaoperationen (floating point) schneller als Ganzzahloperationen (integer) ausgeführt werden können. Darüber hinaus können Gleitkommaoperationen meist parallel zu ganzzahligen Berechnungen ausgeführt werden – Beispiele dafür sind der Intel i486 Mikroprozessor [Intel Corporation, 1990], Intel Pentium [Intel Corporation, 1995] oder der Digital Alpha 21164PC Mikroprozessor [Digital Semiconductor, 1997]. Daher können Programmimplementierungen neuronaler Algorithmen oder von Signalverarbeitungsalgorithmen bei Gleitkommadarstellung eine höhere Verarbeitungsgeschwindigkeit erreichen, als bei Implementierung der gleichen Algorithmen mit Festkommadarstellung.

Gleitkommaarithmetik bietet den Vorteil höherer Rechengenauigkeit und besserer Zahlendarstellungsdynamik allerdings zum Preis höheren Implementierungsaufwands [Hennessy und Patterson, 1996]. Der größere Implementierungsaufwand für integrierte Gleitkommarechenwerke spielt jedoch bei Standardprozessoren, die in sehr großen Stückzahlen gefertigt werden, nur eine untergeordnete Rolle. Darüber hinaus hat der Prozessor in einem Computer einen vergleichsweise geringen Anteil an den Gesamtsystemkosten, verglichen mit dem Kostenanteil eines Prozessors in einer Steuerung oder in einem mobilen System.

Bei Prozessoren früherer Generationen (z.B. Intel 80386, Motorola MC68020) war kein Gleitkommarechenwerk integriert, sondern es war allenfalls als Coprozessor verfügbar (z.B. Intel 80387, Motorola MC68881). Neben Signalprozessoren mit Ganzzahlrechenwerk gibt es Signalprozessoren mit Gleitkommarechenwerk oder architekturverwandte Signalprozessoren mit Festkommarechenwerk oder Gleitkommarechenwerk (z.B. Motorola DSP56002 (24-Bit Festkommaprozessor) / DSP96002 (32-Bit Gleitkommaprozessor)). Dagegen ist der Videosignalprozessor TMS320CX80 der Firma Texas Instruments ein Ein-Chip-Multiprozessor, bestehend aus vier Signalprozessoren mit Ganzzahlrechenwerk und (nur) einem mit Gleitkommarechenwerk ausgestatteten Signalprozessor [Texas Instruments Inc., 1997].

Abschließend läßt sich festhalten, daß Gleitkommaarithmetik gegenüber Festkommaarithmetik aus Sicht der Implementierung von numerischen Algorithmen auf Standardplattformen den Vorteil der höheren Darstellungsgenauigkeit und der höheren Rechengenauigkeit hat und daher vorzuziehen ist, abgesehen von der höheren, erreichbaren Verarbeitungsgeschwindigkeit. Bei

anwendungsspezifischen Schaltungsimplementierungen sind die Randbedingungen entgegengesetzt. Festkommaarithmetik ist in der Regel erforderlich, um eine effektive Implementierung erreichen zu können. Andernfalls muß auf Standardprozessoren zurückgegriffen werden.

Schaltungstechnische Randbedingungen

Zur effektiven Implementierung von numerischen Algorithmen mit Hilfe von anwendungsspezifischen Schaltungen ist zuerst die Frage zu klären, ob betreffende Algorithmen grundsätzlich zur Implementierung auf Basis von Festkommaarithmetik oder anderer numerischer Näherungen geeignet sind. Gegebenenfalls sind dann benötigte Datenwortbreiten unter Berücksichtigung von sich ergebenden Quantisierungseffekten zu bestimmen. Dabei ist zu beachten, daß der Implementierungsaufwand (Chipflächenbedarf) für bestimmte arithmetische Operationen nichtlinear von der Datenwortbreite abhängt [Omondi, 1994]. Mögliche Zahlenbereichsüberschreitungen sind zu berücksichtigen und gegebenenfalls zu behandeln – etwa durch einfache Begrenzung auf festgelegte Grenzwerte. Dazu sind jedoch unter Umständen Erweiterungen notwendig, wie Schutzstellen mit zusätzlichen Datenbits.

Quantisierungseffekte aufgrund endlicher Datenwortbreiten haben einen Einfluß auf das resultierende Verhalten eines übergeordneten Systems. Ein anschauliches und analytisch einfach zu handhabendes Beispiel liefern FIR-Filter: Ein zu filterndes Signal ist in Zeit und Amplitude quantisiert. Die zeitliche Quantisierung eines Signals ist durch das Abtasttheorem (s. Abschnitt 2.3, S. 26) charakterisiert. Die Quantisierung der Amplitude limitiert den maximalen Signalrauschabstand (s. Abschnitt 2.3, S. 26). Durch Quantisierung der Filterkoeffizienten – etwa durch Festkommadarstellung – ergibt sich verglichen mit reellen Filterkoeffizienten ein veränderter Frequenzgang. Der Frequenzgang eines FIR-Filters läßt sich durch Fourier-Transformation der Filterkoeffizienten errechnen. Mit einem Einheitsimpuls – ein Impuls mit Maximalamplitude und Dauer einer Abtastung – können Filterkoeffizienten quasi aus einem Filter ausgelesen werden, auch wenn diese nicht direkt von außen zugänglich sind. Dabei sind dann Quantisierungseffekte sowie mögliche Zahlenbereichsbegrenzungen berücksichtigt. Ein Beispiel ist etwa die Beschränkung auf positive Signalwerte: Auch wenn mit negativen Filterkoeffizienten gerechnet wird, so werden nur positive Ergebnisse ausgegeben. Dadurch werden mit einem Einheitsimpuls nur positive Filterkoeffizienten ausgegeben, die einen etwas veränderten Frequenzgang repräsentieren, der durch Fourier-Transformation der Impulsantwort genau bestimmt werden kann. Abhandlungen in Bezug auf Quantisierungseffekte bei der Implementierung von Filtern ist etwa in [von Grünigen, 1993; Hess, 1993] zu finden.

Beim konkreten Schaltungsentwurf sind möglichst Schaltungsbeschreibungen zu realisieren, die von einem verwendeten Synthesewerkzeug effizient auf *funktionale Blöcke* – etwa bestimmte Standardzellen im VLSI-Entwurf [Kolla *et al.*, 1989], Logikzellen beim FPGA-Entwurf – des verwendeten Entwurfswerkzeugs abgebildet werden können. Wie *effizient* eine bestimmte Variante wirklich ist, läßt sich oft nur durch Implementierung eines Prototypen bestimmen. Beispielsweise scheint die Quantisierung von Faktoren auf Zweierpotenzen eine sehr effiziente Schaltungsimplementierung der Multiplikation zu ermöglichen, da die Multiplikation dabei einfach durch eine Schiebeoperation realisiert werden kann. Stellt ein Entwurfssystem jedoch flächenoptimierte Multiplizierer als funktionale Blöcke (etwa als Standardzellen oder Makrozel-

len) zur Verfügung, so läßt sich die direkte Multiplikation fast ebenso effizient realisieren, aber mit höherer Rechengenauigkeit. Diese Möglichkeit wird auch im Zusammenhang mit digitalen Filtern in [Hess, 1993] diskutiert. Ein im Kontext der Bewertung von Simulationsergebnissen erstellter, synthetisierbarer FIR-Filter wurde mit Hilfe einer im Rahmen der Arbeit entwickelten Schaltung zum Austausch von analogen Videosignalen mit Simulationen [Larsson, 1996] bezüglich direkter Multiplikation und Schiebeoperation dahingehend untersucht und ist in Abschnitt 4.3.2 beschrieben.

2.5.2 Simulation und Validierung

Beim (digitalen) Schaltungsentwurf können aus ganz unterschiedlichen Gründen Fehler auftreten. Korrekturen fehlerhafter Schaltungsentwürfe sind in der Regel sehr viel teurer als etwa Korrekturen fehlerhafter Programme. Das gilt insbesondere für den Entwurf von nichtprogrammierbaren, integrierten Schaltungen (ASICs). Die Korrektur eines Entwurfsfehlers erfordert eine vollständige Neuherstellung des betreffenden Chips. Bei programmierbaren Logikbausteinen (EPLDs, FPGAs) ist das anders: Eine Korrektur erfolgt durch Neuprogrammierung.

Die erreichbare Packungsdichte ist jedoch beim Entwurf mit Standardzellen [Weste und Eshraghian, 1993] sehr viel höher als bei programmierbaren Bausteinen. Diese Bausteine eignen sich jedoch sehr gut zur Realisierung von Prototypen oder für Kleinstserien bis zu Einzelstücken.

Simulationen dienen der funktionalen Validierung von implementierten Algorithmen. Bei der Implementierung von Algorithmen in Form von Software gibt es, verglichen mit der Implementierung in Form von Hardware, eigentlich nur zwei Erscheinungsformen eines Programms, nämlich den Quellcode (die Spezifikation) und den Binärcode (das ausführbare Programm). Technische Erscheinungsformen eines Programms, wie Objektcode, sind aus Sicht des Software-Entwurfs nur technische Zwischenschritte.

Die Validierung eines Programms zur Überprüfung von zuvor in einer Programmiersprache spezifiziertem Verhalten erfolgt durch Ausführen. Verglichen mit dem Schaltungsentwurf stellt dies eine funktionale Validierung dar. Bei der Software-Implementierung von Echtzeitsystemen ist zudem noch die Überprüfung der Einhaltung von gestellten Echtzeitanforderungen notwendig. Die Validierung eines Schaltungsentwurfs umfaßt dagegen mehrere Abschnitte. Am Anfang steht ebenfalls die funktionale Validerung. Die *formale Verifikation* [Kropf, 1998] steht im Gegensatz zu simulativen Methoden [Rammig, 1989]. Die formale Verifikation verwendet mathematische Techniken, um die funktionale Äquivalenz zwischen verschiedenen Spezifikationen nachzuweisen – also etwa die funktionale Äquivalenz zwischen einer abstrakten Verhaltensbeschreibung und einer daraus synthetisierten Gatternetzliste [Keating und Bricaud, 1998]. Inhaltliche Fehler einer Spezifikation lassen sich mit einer formalen Verifikation nicht aufspüren [Kropf, 1998].

Ein funktional validierter Schaltungsentwurf erfordert unter bestimmten technischen Randbedingungen die Validierung unter Berücksichtigung von zunächst als typisch angenommenen Signallaufzeiten mit Hilfe der *Timing-Simulation* [Weste und Eshraghian, 1993]. Darüber hinaus ist vor der Fertigung eines Schaltungsentwurfs als VLSI-Chip eine weitere *Post-Layout-Timing-Simulation* notwendig, bei der – im Gegensatz zu als typisch angenommenen Signallaufzeiten und geschätzten Verbindungslaufzeiten – aus einem vorliegenden Layout durch Analyse extra-

hierte Signalverzögerungszeiten verwendet werden. Der Grund dafür, daß Schaltungssimulationen auf den verschiedenen Abstraktionsebenen – funktionale Simulation, Timing-Simulation, Post-Layout-Simulation – durchgeführt werden, liegt in den zur Simulation benötigten Rechenzeiten, da diese von Abstraktionsebene zu Abstraktionsebene um Größenordnungen (einige Zehnerpotenzen) ansteigen.

Zur Logiksimulation und Logiksynthese eingesetzte kommerzielle Programmpakete können wie andere Programme auch fehlerhaft sein. Außerdem können Synthesewerkzeuge bei der Logiksynthese impliziten Annahmen unterliegen, die unter Umständen nicht vom Autor eines VHDL-Modells beabsichtigt sind [Bhasker, 1996]. So verfügen manche Entwurfsprogramme über Funktionen zur automatischen Verifikation oder Validierung von Schaltungsbeschreibungen unterschiedlicher Abstraktionsebenen. Eine formale Verifikation kann zwei unterschiedliche Spezifikationen einer Schaltung auf funktionale Äquivalenz hin überprüfen. Zwar kann durch formale Verifikation die funktionale Äquivalenz unterschiedlicher Spezifikationen einer Schaltung auf unterschiedlichen Abstraktionsniveaus einer bestimmten Schaltung nachgewiesen werden, Spezifikationsfehler, die bei der Erstellung einer funktionalen Spezifikation einer Schaltung gemacht wurden, lassen sich dadurch jedoch auch nicht ausschließen [Kropf, 1998].

Die Simulation einer Schaltung sollte so nah wie möglich am Zielsystem orientiert sein und in der Weise erfolgen, wie die Schaltung später in einer realen Umgebung eingesetzt wird [Weste und Eshraghian, 1993]. Dabei ist es unter Umständen erforderlich sich weitgehend auf die Simulation von Verhaltensmodellen höherer Abstraktionsebenen zu beschränken und nur abschließend Schaltungsbeschreibungen oder Schaltungsbeschreibungen von Teilkomponenten unterschiedlicher Abstraktionsebenen gegeneinander auf funktionale Äquivalenz zu überprüfen.

Simulationsmodelle

Es ist aufgrund der benötigten Rechenzeit nicht praktikabel, große digitale Schaltungen zur funktionalen Validierung auf der elektrotechnischen Ebene zu simulieren, sondern digitale Komponenten werden statt dessen als mehrwertige Logik (MVL $\approx$ *Multiple Valued Logic*) modelliert, zusammen mit vereinfachten Modellen für die Verzögerung und Übertragung logischer Signale [Marwedel, 1993]. Sowohl MVL als auch die verwendeten Modelle für Verzögerungszeiten und Signalübertragung resultieren aus elektrotechnischen Vorgängen. Für die digitale Schaltungssimulation ist diese Vereinfachung jedoch im allgemeinen hinreichend. Simulationsmodelle sollten einerseits so detailliert wie möglich sein, um möglichst realitätsnahe Simulationen durchführen zu können. Um Simulationen mit praktikablen Rechenzeiten durchführen zu können, sollten andererseits die verwendeten Simulationsmodelle nur so weit wie nötig detailliert sein.

Testumgebung

Stehen (synthetisierbare) Schaltungsbeschreibungen erst einmal zur Verfügung, so sind diese zunächst funktional zu validieren. Dazu werden die Schaltungsbeschreibungen in eine *Testumgebung* integriert (s. Abbildung 2.19) und mit Signalen an den Schaltungseingängen stimuliert, den sogenannten *Stimuli*. Früher wurden dazu zunächst Stimuli – meist in einem festen Zeitraster – erzeugt und bei der Logiksimulation an die Schaltungseingänge gelegt.

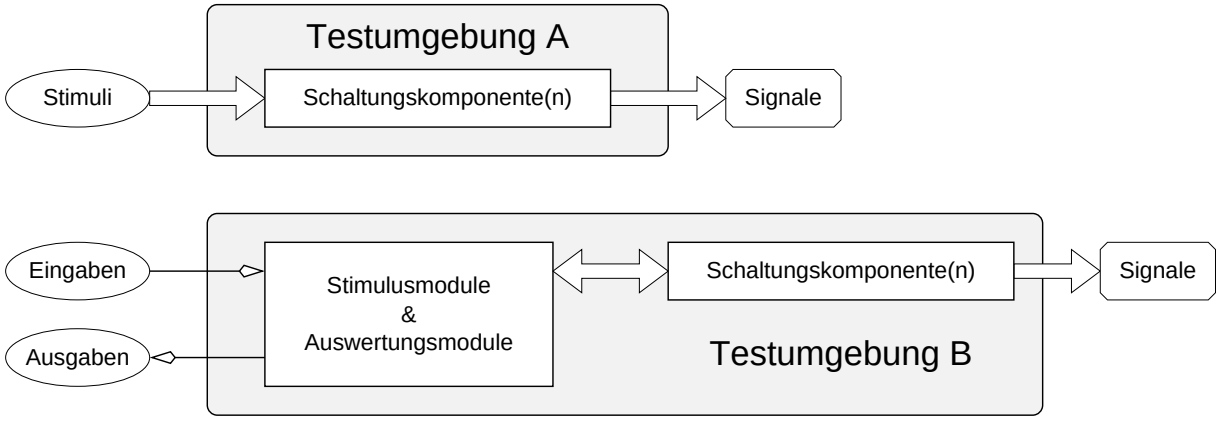


Abbildung 2.19: **Testumgebungen**

Die Verwendung statischer Stimuli zur Simulation ist nach wie vor über definierte Dateischnittstellen mit VHDL oder Verilog möglich [Smith, 1998]. Beide Sprachen bieten jedoch über das Schema starrer Stimuli hinausgehende Möglichkeiten: Die prozedurale Erzeugung von Stimuli während der Simulation mit Hilfe von geeigneten Verhaltensmodellen, in Form autonomer Überwachungsmodule, mit denen die funktional zu validierenden Schaltungsbeschreibungen interagieren.

Ein einfaches Beispiel ist etwa ein DRAM-Controller, der mit einem (nicht synthetisierbaren) Verhaltensmodell eines DRAMs interagiert. Ein DRAM-Modell wird durch Signale eines funktional zu validierenden DRAM-Controllers angesteuert und reagiert auf die Stimuli wie ein DRAM. Zusätzlich besteht die Möglichkeit, daß das DRAM-Modell Warnungen bei fehlerhaften Signalfolgen ausgibt, wie im folgenden Abschnitt skizziert wird. Derartiges Vorgehen wurde auch bei der Entwicklung der Videosignalsimulationsschnittstelle *VidTrans* (vgl. Abschnitt 4.3, S. 115) erfolgreich eingesetzt. Durch interagierende Simulationsmodelle von Systemkomponenten lassen sich Logiksimulationen erheblich realitätsnäher durchführen als mit starren Folgen von Signalvektoren.

Impulsdiagramme

Schaltungsentwürfe können durch geeignete Simulationen validiert werden. Logiksimulatoren ermöglichen zwar meist sehr dataillierte Simulationen, die aber entsprechend umfangreiche Simulationsergebnisse liefern können, die es auszuwerten und zu bewerten gilt. Zur Darstellung von Simulationsergebnissen dienen Impulsdiagramme, wie in Abbildung 2.20 dargestellt. Die Darstellung der Signalverläufe in verschiedenen Zeitskalen ist zwar gut zur Bewertung von Detailergebnissen geeignet, jedoch nur bedingt, um komplexe Wechselwirkungen zwischen Systemkomponenten zu beurteilen.

Impulsdiagramme sind ein wertvolles Hilfsmittel zum Aufspüren von Fehlern in überschaubaren Teilschaltungen und deren Validierung. Dazu sind von einer Simulation gelieferte Signalverläufe mit erwarteten Signalverläufen zu vergleichen. Die Gefahr, daß dabei Fehlverhalten ei-

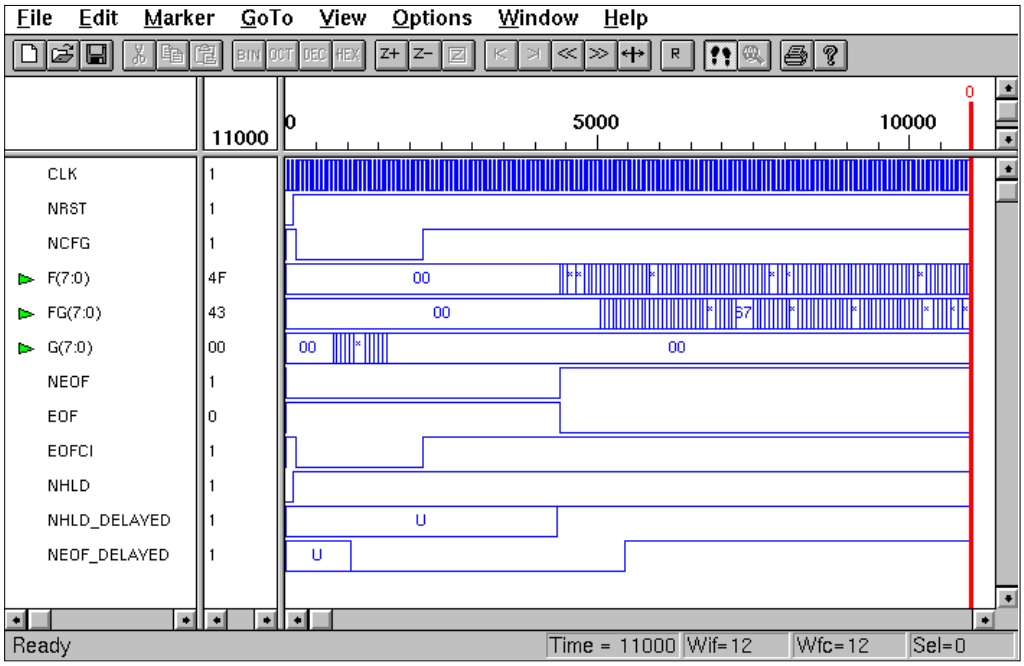


Abbildung 2.20: **Impulsdiagramm** *als Ergebnis einer Logiksimulation (FIR-Filter).*

ner entworfenen Schaltung nicht entdeckt wird, ist jedoch sehr groß, wenn für die Bewertung von Simulationsergebnissen die Berücksichtigung dargestellter Signale über viele Zeitskalen hinweg erforderlich ist.

Signalverläufe, die sich über Zeitskalen vieler Größenordnungen erstrecken (*Multiple Time Scale*) lassen sich anhand von Impulsdiagrammen nicht validieren. So ist ist es ebenso wenig möglich, mit einem Oszillographen eine Aussage darüber zu machen, ob ein dargestelltes Farbvideosignal auch tatsächlich auf einem Farbfernseher ein farbiges Bild liefert – die zur Bewertung erforderliche Zeitskala erstreckt sich dabei von der Größenordnung um $100\ ns$ bis in die Größenordnung von $100\ ms$. Um ein erzeugtes Farbvideosignal als ein gültiges zu identifizieren, ist die Darstellung mit einem Farbfernseher eine Möglichkeit, oder die Auswertung des Signals mit Hilfe eines Programms, welches einen Farbfernseher nachbildet. Ähnlich sind die Verhältnisse bei der Schaltungsimplementierung von neuronalen Algorithmen – die Bewertung von Simulationsergebnissen erfordert Methoden zur Bewertung des Verhaltens eines implementierten Netzwerks.

Simulation des Zeitverhaltens

Ein funktional validierter Schaltungsentwurf muß Randbedingungen bezüglich seines Zeitverhaltens genügen. Zur Überprüfung dienen Schaltungssimulationen mit aus einem synthetisierten Modell extrahierten Gatterlaufzeiten und Signallaufzeiten. Die Extraktion von Laufzeiten wird

als *Back-Annotation* bezeichnet [Weste und Eshraghian, 1993]. Die Entwurfswerkzeuge verwenden dazu bestimmte Bauteilbibliotheken, die typische Gatterlaufzeiten funktionaler Blöcke (Gatter, Flipflops, etc.) enthalten. Dabei wird in der Regel eine Laufzeit für einen bestimmten logischen Block angenommen. Extraktion von Signallaufzeiten beruht auf typisch angenommenen Signallaufzeiten pro Leitungslängeneinheit. Das Zeitverhalten einer implementierten Schaltung wird aufgrund von zufälligen Laufzeitstreuungen von der Simulation abweichen. Hängt das funktionale Verhalten einer Schaltung von tatsächlichen Laufzeiten oder Laufzeitdifferenzen ab, so besteht sogar die Gefahr, daß sogar das funktionale Verhalten der implementierten Schaltung von der funktionalen Simulation abweicht – die betreffende Schaltung funktioniert dann nicht wie erwartet. Dieses Problem läßt sich durch *synchrone Schaltungsentwürfe* vermeiden, da dabei die Gültigkeit von Signalen in einem festen Zeitraster durch das *Taktsignal* extern bestimmt wird und nicht mehr von internen, zufällig gestreuten Laufzeiten abhängt. Die maximale, interne Laufzeit eines Signals – bestimmt durch den längsten Pfad – begrenzt dann nur noch die maximal mögliche Verarbeitungsgeschwindigkeit einer Schaltung, jedoch nicht deren funktionales Verhalten.

Autonome Überwachungsmodule

Schaltungsbeschreibungssprachen wie VHDL oder Verilog erlauben die Realisierung von Modulen, die die eigenständige Überwachung von Teilschaltungen ermöglichen und nur bei Fehlern entsprechende Ausgaben liefern. Dadurch kann die Menge der auszuwertenden und zu bewertenden Simulationsergebnisse erheblich reduziert werden. Ein einfaches Beispiel dazu ist das im folgenden in VHDL formulierte Verhaltensmodell, welches während einer Simulation die Einhaltung von *Setup Time* und *Hold Time* selbständig überwacht und bei Verletzung eine entsprechende Warnung ausgibt, wie das folgende Beispiel zeigt. Dabei seien *Setup Time* und *Hold Time* als Konstanten `T_DS` und `T_DH` definiert, um eine bessere Übersichtlichkeit gewährleisten zu können.

```
constant T_DS  : time := 1 ns;  -- Data In Setup Time
constant T_DH  : time := 5 ns;  -- Data In Hold Time
```

Ein Verhaltensmodell zur Überwachung ist als ein `process` innerhalb einer `architecture` in VHDL formuliert. Der Prozeß ist auf Änderungen der zwei Signale `clk` und `d` sensitiv. Das Signal `clk` hat dabei die Semantik eines Taktsignals und das Signal `d` die eines Datensignals, wie sie etwa bei einem D-Flipflop vorkommen.

```
chksh_p: process (clk, d)
         begin
           if (clk'event and clk = '1') then
               assert d'stable (T_DS)
                 report "DFFSH : setup time violation" severity warning;
           end if;
           if (d'event and clk = '1') then
               assert clk'stable (T_DH)
                 report "DFFSH : hold time violation" severity warning;
           end if;
         end process;
```

Die überwachten Ereignisse sind in Abbildung 2.21 skizziert. Der erste `if-then`-Block überprüft bei steigender Flanke des Taktsignals `clk`, ob das Signal `d` sich während der Zeit `T_DS` *nicht* geändert hat, also stabil war, sonst wird: `"DFFSH : setup time violation"` als Warnung ausgegeben. Der zweite `if-then`-Block überprüft dann, wenn das Taktsignal `clk` logisch `'1'` ist, bei einem Pegelwechsel des Signals `d`, ob das Taktsignal `clk` sich während der Zeit `T_DH` *nicht* geändert hat, also vor dem Pegelwechsel des Signals `d` stabil war, sonst erfolgt die Ausgabe der Warnung: `"DFFSH : hold time violation"`.

Verilog stellt zur Überprüfung von Setup-Time- und Hold-Time-Bedingungen vordefinierte Funktionen (`$setup`, `$hold`, `$setuphold`) zur Verfügung [Bhasker, 1997]. Von der Möglichkeit, eine automatische Überwachung von Zeitbedingungen innerhalb von Simulationsmodellen zu verwenden, macht beispielsweise das EPLD/FPGA-Entwurfssystem *MAX+PLUS II* [Altera Corporation, 1998d] Gebrauch. Synthetisierte VHDL (oder Verilog) Modelle von gewählten EPLDs oder FPGAs können vom Entwurfssystem in eine Datei geschrieben werden und dann zur Schaltungssimulation (*Gate Level / Timing Simulation*) auf der Logikzellenebene verwendet werden, um einen Entwurf bezüglich der Einhaltung von Zeitbedingungen zu überprüfen. Die dazu verwendeten Logikzellenmodelle enthalten auch typische Signallaufzeiten.

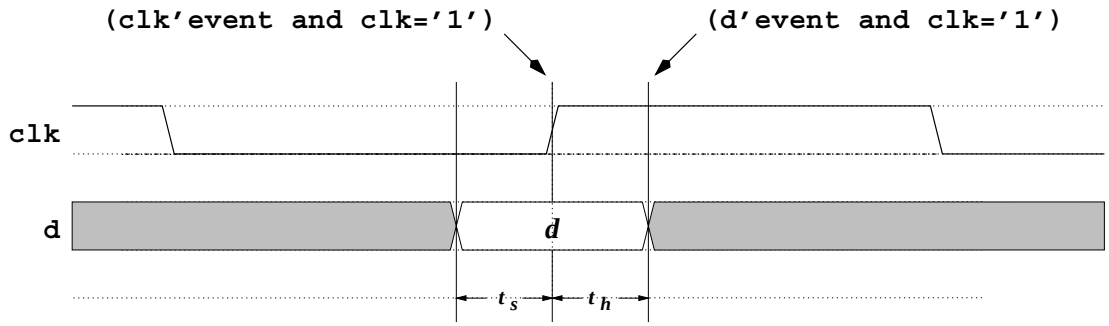


Abbildung 2.21: **Überprüfung von Zeitbedingungen – Setup-/Hold-Time.** *VHDL stellt Attribute ('event, 'stable, ...) zur Verfügung, welche die Formulierung von Zeitbedingungen zur automatischen Überwachung von Signalen während einer Simulation ermöglichen.*

Systemsimulation

Da integrierte Schaltungen immer mehr Komponenten enthalten, so daß ganze Systeme auf einem Chip integriert werden können, werden kommerzielle Werkzeuge zur Unterstützung von Systemsimulationen angeboten. Ein Beispiel ist die Produktfamilie namens *System Realizer* der Firma *Quickturn Design Systems* zur Validierung komplexer, integrierter Schaltungen – derzeit mit bis zu 10^7 Gattern – in der Systemumgebung. Um Schaltungssimulationen auch über große Zeitskalen hinweg durchführen zu können, werden zum einen Simulationsbeschleuniger, basierend auf FPGAs oder speziellen Parallelrechnern, angeboten, und zum anderen werden Systemkomponenten (PC, Telekommunikation, ...) angeboten, die soweit verlangsamt sind, daß sie direkt mit den Simulationsbeschleunigern kommunizieren können.

Die Systemsimulation ist ein Hilfsmittel zum Aufdecken verdeckter Abhängigkeiten zwischen neu entworfenen Systemkomponenten mit anderen Systemkomponenten. Auf die Notwendigkeit von möglichst systemnahen Simulationen wird schon in [Weste und Eshraghian, 1993] im Zusammenhang mit der Realisierung von digitalen Filtern zur Filterung von Videosignalen im Zusammenhang mit Entwurfsmethodik hingewiesen. Die im Rahmen der vorliegenden Arbeit entwickelten Verfahren gehen noch einen Schritt weiter: *VidTrans* [Larsson, 1996] (vgl. Abschnitt 4.3, S. 115) ist eine als Hardware realisierte Bewertungsmethode für videosignalverarbeitende Schaltungen, die sogar analoge Signalpfade mit einbezieht, und Lerntrajektorien [Larsson, 1997] (vgl. Abschnitt 4.2, S. 82) sind eine als Software realisierte Bewertungsmethode für Implementierungen des *Lernens* mit dem Backpropagation-Algorithmus, die aber auch als Visualisierungsmethode für verwandte Optimierungsaufgaben dienen kann.

Prototyping mit PLDs

Programmierbare Logikbausteine eignen sich für die Realisierung von Prototypen und deren Test unter Systembedingungen in einer realen Zielsystemumgebung und ermöglichen so, die Wechselwirkung zwischen entwickelten Systemkomponenten mit vorhandenen Systemkomponenten zu untersuchen, um einen Entwurf funktional zu validieren, wie auch im Rahmen der vorliegenden Arbeit anhand eines Fallbeispiels gezeigt wurde [Larsson, 1999b]. Darüber hinaus eignen sich PLDs für den Einsatz in der Lehre, um die Wechselwirkung zwischen entwickelten digitalen Komponenten und einer Zielsystemumgebung praktisch zu vermitteln [Larsson *et al.*, 1994].

2.5.3 Testbarkeit

Neben dem funktionalen Test zum Aufspüren von Entwurfsfehlern, die als Folge fehlerhafter beziehungsweise unvollständiger Schaltungsspezifikationen aber auch durch fehlerhafte Entwurfswerkzeuge auftreten können, ist der Test auf Herstellungsfehler notwendig. Obwohl die zahlreichen und sehr aufwendigen Herstellungsschritte bei der Chipfertigung hochgradig fehlerfrei verlaufen, sind Herstellungsfehler nicht zu vermeiden. Darüber hinaus liegt bei der Herstellung integrierter Schaltungen, aufgrund verschiedener technischer Randbedingungen und betriebswirtschaftlicher Voraussetzungen, ein Herstellungskostenoptimum bei einer Chipausbeute von etwa 37% [Lagemann, 1987]. Um nicht fertigungsfehlerhafte Chips auszuliefern, muß jede inte-

grierte Schaltung nach der Fertigung beim Halbleiterhersteller auf Fertigungsfehler getestet werden. Im Gegensatz zu funktionalen Tests, welche zur Überpüfung einer integrierten Schaltung als Einheit dienen, dienen die Tests nach der Herstellung der Überprüfung der einzelnen Logikgatter einer integrierten Schaltung auf Fertigungsfehler, um defekte integrierte Schaltungen frühzeitig ausmustern zu können.

Im Gegensatz zu diskret aufgebauten Schaltungen ist bei integrierten Schaltungen nur ein sehr geringer Bruchteil interner Signale von außen zugänglich, da sowohl die aktiven Elemente als auch deren Verdrahtung in mehreren, übereinander liegenden Schichten realisiert werden. Im Labor sind allenfalls die obersten Schichten zugänglich. Der Test auf Fertigungsfehler muß daher über die extern zugänglichen Signale – über die Anschlüsse – erfolgen. Dieser Umstand ist schon beim Entwurf integrierter Schaltungen (ASICs) zu berücksichtigen oder wenigstens vor der endgültigen Fertigstellung der betreffenden integrierten Schaltung. Darüber hinaus muß der Test auf Fertigungsfehler in möglichst kurzer Zeit erfolgen, um nicht ein Engpaß bei der Chipherstellung zu sein. Das Aufspüren von Fertigungsfehlern erfolgt mit sogenannten *Testmustern*. Dabei ist jedoch nur die Aussage, daß ein Fertigungsfehler vorliegt entscheidend, um den betreffenden Chip auszumustern. Wo ein Fertigungsfehler auf dem Chip lokalisiert ist, ist für die Ausmusterung irrelevant.

Bei programmierbaren Logikbausteinen – wie auch bei integrierten Standardschaltungen – ist der Test auf Fertigungsfehler Aufgabe der Hersteller. Um die Testbarkeit einer Schaltung zu gewährleisten, ist unter Umständen zusätzliche Logik erforderlich, die etwa nachträglich hinzugefügt werden kann, oder durch Austausch funktionaler Blöcke, die durch zusätzliche Elemente (*Scan Path*) zwecks Testbarkeit erweitert sind.

Fehlermodell

Für integrierte CMOS-Schaltungen wird ein vereinfachtes Modell für die Auswirkung von Fertigungsfehlern auf das Schaltungsverhalten zugrunde gelegt [Weste und Eshraghian, 1993], die sogenannten *Stuck-At-Fehler*. Dabei wird davon ausgegangen, daß ein Fertigungsfehler einen Kurzschluß zwischen Leitungen hervorruft, der einen Gattereingang auf einen festen Signalpegel klemmt: entweder auf logisch 0, bezeichnet als *stuck-at-0*, oder auf logisch 1, was als *stuck-at-1* bezeichnet wird. Es gibt darüber hinaus auch weitergehende Fehlermodelle, die Fertigungsfehler als Kurzschlüsse und als offene Leitungen modellieren [Weste und Eshraghian, 1993]. Von darüber hinaus gehenden Störungen elektrischer Parameter aktiver Elemente, wie beispielsweise veränderten Innenwiderständen elektronischer Elemente, wird in diesem Zusammenhang abstrahiert [Williams, 1986]. So sind durchaus Fertigungsfehler denkbar, die sich beispielsweise als zu geringe Treiberleistung von Ausgängen – außerhalb der Spezifikation – bemerkbar machen, sich jedoch erst in einer Systemumgebung als Fehler bemerkbar machen.

Schaltnetze und Schaltwerke

Um ein Schaltnetz mit n Eingängen bezüglich von Stuck-At-Fehlern vollständig testen zu können, sind prinzipiell 2^n Bitvektoren – Testmuster oder Testvektoren – notwendig, wobei die Reaktion des zu testenden Schaltnetzes mit der erwarteten Reaktion zu vergleichen ist. Um einen

Automaten mit n Eingängen und m Zustandsbits vollständig zu testen, erscheinen zunächst sogar mindestens 2^{n+m} Muster für einen vollständigen Test notwendig [Weste und Eshraghian, 1993].

Fehlersimulation und Fehlerüberdeckung

Allerdings sind in der Regel jedoch nicht alle 2^n beziehungsweise 2^{n+m} Testmuster tatsächlich erforderlich, um eine Schaltung in Bezug auf Stuck-At-Fehler zu testen. In diesem Zusammenhang ist der Begriff der *Fehlerüberdeckung* zu nennen. Unter Fehlerüberdeckung wird der Anteil der Stuck-At-Fehler verstanden, der durch einen Teil von Testmustern abgedeckt wird. Dieser Anteil wird mit Hilfe von sogenannten *Fehlersimulationen* bestimmt. Testbarkeit, Testmustergenerierung, Fehlerüberdeckung, Entwurf zur Testbarkeit sind nicht Schwerpunkt der vorliegenden Arbeit und werden daher im folgenden auch nicht weiter vertieft, sondern es sei an dieser Stelle auf die Literatur verwiesen [Williams, 1986; Weste und Eshraghian, 1993].

2.6 Offene Fragen

Die Erhöhung der Integrationsdichte und Vergrößerung der Chipfläche ermöglicht, die Zahl der Komponenten, die auf einem Chip untergebracht werden können, zu steigern. So nimmt diese Zahl schon über viele Jahre *exponentiell* zu und gehorcht damit schon über nunmehr drei Jahrzehnte der als *Moore'sches Gesetz* bekannten These [Moore, 1998; Bondyopadhyay, 1998], die exemplarisch für eine Prozessor-Familie mit Zahlen in [Intel Corporation, 1997] aufgeführt ist. Mit der Steigerung der Integrationsdichte, durch Verkleinerung der geometrischen Strukturen integrierter Schaltungen, können darüber hinaus Signallaufzeiten verringert werden [Weste und Eshraghian, 1993]. Schnellere integrierte Schaltungen erschließen immer neue Anwendungsbereiche. Dabei werden zunehmend Anwendungsbereiche erreichbar, die einen sehr hohen Datendurchsatz erfordern. Um den für Anwendungen aus dem Bereich der digitalen Signalverarbeitung notwendigen Datendurchsatz erreichen zu können, wird als erforderliche Performanz in [Ramakrishna und Bayoumi, 1994] ein Bereich von $10^2 \ldots 10^6\ MOPS$ (≈ *Millions of Operations per Second*) als typisch charakterisiert.

Zeitskalendynamikumfang

Bei bestimmten Anwendungen erstreckt sich die Verarbeitung über Zeitskalen von mehreren Größenordnungen (*engl. Multiple-Time-Scale*) in einer Weise, daß die Beobachtung und Bewertung von Ergebnissen mit hoher Zeitauflösung über lange Zeitintervalle erfolgen muß. Digitale Videosignalverarbeitung und neuronale Netzwerke sind Beispiele für solche Anwendungen. Es genügt zur Validierung nicht, nur einige wenige, zeitlich lokale Verarbeitungsoperationen zu beobachten, statt dessen ist es notwendig, Systemverhalten über einen längeren Zeitraum zu beobachten. Ob die Implementierung eines neuronalen Algorithmus wie erwartet funktioniert, läßt sich durch Beobachtung des Verhaltens des Algorithmus validieren. Zur Validierung digital videosignalverarbeitender Systeme, wie im Rahmen der vorliegenden Arbeit evaluiert, ist die Beobachtung einer Bildfolge, also eines Videosignals von der Dauer einiger Bilder bei geeigneter Abtastfrequenz, erforderlich.

Robustheit

Der klassische Schaltungsentwurf basiert meist auf einer expliziten, aber unvollständigen Spezifikation. Sobald eine simulierbare Spezifikation – etwa in einer Schaltungsbeschreibungssprache – vorliegt, wird der vorliegende Entwurf durch Simulation überprüft und gegebenenfalls korrigiert oder auch erweitert. Zur Simulation dient ein Satz von Beispielen. Neben der expliziten Spezifikation, verbal, etwa in Form eines Pflichtenhefts, existiert eine implizite Spezifikation, in Form von Hintergrundwissen und darauf basierenden Annahmen. Entscheidend für eine praktische Nutzbarkeit eines Systems ist dessen Stabilität in Bezug auf geringfügige Abweichungen von expliziten Spezifikationen – seine *Robustheit*. Ein System soll auch bei Eingaben, die von Spezifikationen in einem bestimmten Maß abweichen, sinnvolle Ausgaben liefern. So sollen signalverarbeitende Systeme sich bis zu einem gewissen Grad gegenüber verrauschten Signalen tolerant verhalten. Andererseits sind Ausgaben möglichst so zu gestalten, daß die Weiterverarbeitung durch andere Systeme von diesen möglichst wenig Toleranz voraussetzt, wobei jedoch definiertes Verhalten außerhalb der Spezifikationen der Zielsysteme zu berücksichtigen ist. Beispiele dafür sind Farbfernsehgeräte und Farbvideorekorder. Ein Farbfernseher stellt auch noch Videosignale mit einem Farbsignalanteil farbig dar, auch wenn der Farbsignalanteil sehr gering ist und damit deutlich außerhalb der Spezifikation liegt. Andererseits kann ein Fernseher, im Gegensatz zu einem Videorekorder, sich noch auf Videosignale einsynchronisieren, deren Synchronsignale außerhalb der Spezifikation liegen. Der Entwurf integrierter Schaltungen, die in einem nicht formal spezifizierbaren Kontext eingesetzt werden sollen, erfordert neue, den Entwurf unterstützende Evaluationsmethoden. Die Notwendigkeit neuer Methoden ist eine Folge der ständig zunehmenden Leistungsfähigkeit integrierter Schaltungen und die damit verbundene Ausdehnung der Anwendungsgebiete in Bereiche komplexerer Anwendungen. Dabei besteht eine gewisse Ähnlichkeit zwischen Software und Hardware: Sehr umfangreiche Software-Systeme erfordern andere Entwurfsmethoden, wie etwa die objektorientierte Programmierung, als kleine Programme, die unter Umständen sogar in Assembler geschrieben werden können.

Impulsdiagramme

Klassische Methoden wie die Darstellung von Impulsdiagrammen unterstützen den Entwerfer zwar beim Aufspüren von Fehlern, die sich innerhalb von relativ kurzen Simulationszeitintervallen zeigen, ermöglichen jedoch nicht oder nur unzureichend eine funktionale Validierung von Systemen, deren Verarbeitung sich über Zeitskalen erstreckt, die einen Umfang von mehreren Zehnerpotenzen aufweisen. Neuronale Netzwerke und Architekturen zur digitalen Verarbeitung analoger Videosignale sind Beispiele für Systeme, deren Verarbeitung sich über Zeitskalen mehrerer Größenordnungen erstreckt.

Simulationsbewertung

Standardisierte Schaltungsbeschreibungssprachen stellen zwar Sprachkonstrukte zur prozeduralen Erzeugung von Stimuli sowie zur automatisierten Auswertung und Überwachung von Detailergebnissen dar, die funktionale Validierung von komplexen Schaltungsentwürfen erfordert jedoch neue, darüber hinausgehende Methoden. Der Ansatz der vorliegenden Arbeit versucht, im

Kontext des digitalen Schaltungsentwurfs, durch die Bereitstellung neuer Methoden einen Beitrag zur Bewertung von Simulationsergebnissen zur Validierung durch *Beobachtung von Systemverhalten* zu leisten.

Aber selbst kleinere, digitale Schaltungen lassen sich durch geeignete Beobachtung des Verhaltens verschiedener, interagierender Systemkomponenten besser validieren, als das mit statischen Stimuli und der Bewertung von Impulsdiagrammen, als ausschließlichem Instrument zur Bewertung von Schaltungsentwürfen, möglich ist. So erschließen interaktive Schaltungssimulationen [Larsson *et al.*, 1997c] schon im Grundstudium neue Möglichkeiten, auf diese Problematik frühzeitig aufmerksam zu machen.

Wechselwirkung mit der Systemumgebung

Die Validierung von (neu) entworfenen Komponenten im System wird durch die standardisierten Schaltungsbeschreibungssprachen unterstützt. Kommuniziert eine entworfene Komponente mit analogen Systemen, so ergeben sich dadurch zusätzliche Randbedingungen für die funktionale Validierung, da analoge Systeme auch außerhalb ihrer Spezifikation definiertes Verhalten zeigen und die entworfene Schaltung sonst nicht in jeder realen Anwendungsumgebung eingesetzt werden kann. Ein Beispiel dafür sind etwa analoge Fernseher, die auch Videosignale noch farbig darstellen, die laut Spezifikation eigentlich nicht farbig dargestellt werden sollten. Die Signaleigenschaften verschiedener Videosignalquellen können erhebliche Unterschiede bezüglich bestimmter Eigenschaften aufweisen, was zu berücksichtigen ist, wenn solche Signale digital verarbeitet werden sollen. Digitalisierte Signale werden durch einen analogen Signalpfad – beispielsweise Vorverstärker, ADC oder DAC – mit charakteristischen Störungen – wie Rauschen oder Verzerrungen – versehen. Digitale Modulationsverfahren, wie sie bei digitaler Videosignalübertragung oder bei der Mobilkommunikation Anwendung finden, sind in diesem Kontext weitere Beispiele.

Reale, analoge Signale können Eigenschaften außerhalb von Spezifikationen aufweisen, deren Berücksichtigung von entscheidender Bedeutung für den praktischen Einsatz ist. Bei der Spezifikation einer Schaltung werden vom Entwerfer häufig implizite Annahmen gemacht. Verdeckte Abhängigkeiten lassen sich nur mit Hilfe geeigneter Systemsimulationen aufdecken. Diese Problematik besteht im Zusammenhang mit digitaler Signalverarbeitung eigentlich schon lange. Durch die ständig wachsende Integrationsdichte integrierter Schaltungen werden aber zunehmend neue Anwendungsbereiche erschlossen, die immer komplexere Schaltungsentwürfe erfordern. So dringt die digitale Signalverarbeitung in Frequenzbereiche vor, wie beispielsweise die digitale Videosignalverarbeitung, die noch vor wenigen einigen Jahren nur mit analogen Schaltungen kostengünstig möglich war.

Signalschnittstellen

Der Austausch von Signalen zwischen simulierten Komponenten und realen Komponenten erlaubt die Validierung von Schaltungsentwürfen in besonders systemnaher Weise. Für den Austausch von Farbvideosignalen kämen grundsätzlich auch kommerziell erhältliche Transienten--Rekorder in Frage, sofern sie über hinreichend viel Speicher verfügen und in der Lage sind, abgetastete Videosignale auf den Abtastwert genau zyklisch abzuspielen, da diese Eigenschaft

für die Bewertung von Simulationsergebnissen im Zusammenhang mit der Entwicklung videosignalverarbeitender digitaler Schaltungen von großer Bedeutung ist.

Ein besonderer Nachteil kommerzieller Transienten-Rekorder besteht jedoch darin, daß analoge Signale die analogen Komponenten und Signalpfade des Transienten-Rekorders durchlaufen, die jedoch im allgemeinen andere sind als bei einem Gerät, das sich gerade in der Entwicklung befindet. Die Verfügbarkeit hinreichend performanter PLDs ermöglicht es, einen problemadaptierten Transienten-Rekorder zu realisieren, bei dem die analogen Signale die gleichen Komponenten und Signalpfade wie bei einem sich in der Entwicklung befindlichen Gerät durchlaufen können, wodurch besonders systemnahe Simulationen ermöglicht werden.

Signal- und Spektrendarstellung

Graphische Darstellungsverfahren, die zur Bewertung von Simulationsergebnissen verwendet werden können, wie die Darstellung von Signalen als Funktion der Zeit in Form von Zeitdiagrammen oder die Darstellung von Signalspektren mit der Fourier-Transformation, liefern zwar exakte, quantitative Berwertungskriterien, erlauben jedoch bezüglich des Zusammenspiels zwischen komplexen Systemkomponenten keine oder nur unzureichende, abschließende Bewertung dargestellter Ergebnisse. Ein großes Problem stellen in diesem Zusammenhang die bei der Simulation größerer Schaltungen anfallenden Datenmengen dar, wenn sich die Simulationen darüber hinaus noch über mehrere Zeitskalen erstrecken müssen, die das Ergebnis von sehr umfangreicheren Operationen sein können, die dann nicht alle im einzelnen nachvollzogen werden können.

Die Auswirkungen von numerischen Näherungen etwa bei Filterkoeffizienten lassen sich zwar durch Angabe von Signalrauschverhältnis oder graphische Darstellung von resultierenden Frequenzgängen abschätzen. Wie sich solche Näherungen jedoch letztendlich auswirken, wenn digitale Filter Komponenten eines übergeordneten Systems sind, läßt sich allenfalls nur abschätzen. Der direkte Austausch von Signalen zwischen Computersimulation und realen (analogen) Systemen erlaubt eine direkte Bewertung von vorgenommenen numerischen Vereinfachungen, was anhand eines rein digitalen PAL-Encoders und eines rein digitalen PAL-Decoders zusammen mit der bidirektionalen Videosignalschnittstelle *VidTrans* im Abschnitt 4.3.1 ab Seite 118 aufgezeigt wird.

Backpropagation-Algorithmus

Die Bewertung der Folgen von numerischen Vereinfachungen bei der Implementierung des Backpropagation-Algorithmus erfolgt oft nur durch Trainieren von bekannten Standardproblemen und anschließende Bewertung der Klassifizierungsergebnisse. Soll die Gesamtfehlerfunktion des betreffenden Backpropagation-Netzwerks beobachtet werden, so erfolgt dies im allgemeinen mit dreidimensionalen Darstellungen, wobei weitere Dimensionen gegebenenfalls festgehalten werden. Die niedrigdimensionale Visualisierung des hochdimensionalen Backpropagation-Lernens kann hier unter Umständen durch direktere Beobachtbarkeit einen Beitrag zur Folgenabschätzung von numerischen Näherungen leisten. Direkte Beobachtung des hochdimensionalen Backpropagation-Lernens gab es bisher nicht. Die im Rahmen der Arbeit eingeführten *Lerntra-*

jektorien (s. Abschnitt 4.2, S. 82) erlauben nun eine direktere Beobachtung als etwa das Beobachten des Gesamtfehlers oder nur die abschließende Beurteilung eines Klassifizierungsergebnisses.

Ausgehend von einem Rahmensystem zur Klassifizierung von Videobildern in Echtzeit mit Hilfe eines auf dem Backpropagation-Algorithmus basierenden neuronalen Netzwerks wurden im Rahmen der vorliegenden Arbeit neue Methoden für die Bewertung von Simulationsergebnissen im Kontext des digitalen Schaltungsentwurfs entwickelt. Darüber hinaus sind ausführbare Spezifikationen für Module zur digitalen Verarbeitung analoger Videosignale entstanden.

Kapitel 3

Gewählter Ansatz

Die *systemnahe Validierung originärer Spezifikationen* komplexer, digitaler Systeme wurde im Rahmen der vorliegenden Arbeit als Ansatz gewählt. Ziel war hierbei die Überprüfung, ob ein spezifiziertes digitales System *robust* ist. Robustheit bedeutet hier Unempfindlichkeit von Algorithmen und Systemen und deren Implementierungen gegenüber Abweichungen vom spezifizierten Verhalten. Beispielsweise soll ein System zur digitalen Signalverarbeitung bis zu einem gewissen Grad verrauschte Signale korrekt verarbeiten können, auch wenn das betreffende System nicht explizit für die Verarbeitung verrauschter Signale spezifiziert wurde. Im Rahmen der vorliegenden Arbeit werden solche Systeme implementiert, die mit Komponenten in *realen* Anwendungsumgebungen interagieren. Die im Rahmen der vorliegenden Arbeit analysierten realen Systemkomponenten zeichnen sich dadurch aus, daß sie sich allenfalls unvollständig modellieren lassen und sich daher der formalen Spezifikation entziehen. Außerdem koexistieren beim Ent-

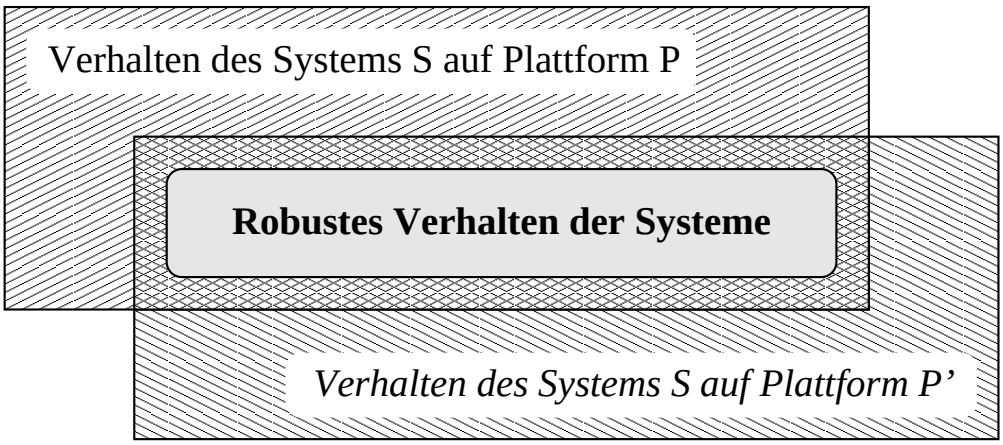


Abbildung 3.1: **System vs. System** *beim Entwurf digitaler Schaltungen. Ist ein System S für unterschiedliche Plattformen P und P' spezifiziert, so kann das Verhalten der Implementierungen des Systems auf den unterschiedlichen Plattformen voneinander abweichen. Zeigen aber beide Implementierungen robustes Verhalten, so können beide Implementierungen, obwohl sie nur ähnliches Verhalten zeigen, als gleichwertig für die Lösung einer bestimmten Aufgabe angesehen werden.*

wurf digitaler Schaltungen meist mehrere, *nicht-äquivalente Spezifikationen* eines Systems, die aber *ähnliches Verhalten* zeigen. Der praktische Einsatz erfordert dann die vergleichende Validierung solcher nicht-äquivalenter Spezifikationen bezüglich der *Robustheit* ihres Verhaltens in der vorgesehenen, realen Anwendungsumgebung.

Die skizzierte Problematik hat beim Entwurf digitaler Schaltungen in den letzten Jahren an Bedeutung gewonnen und wird in Zukunft noch an Bedeutung gewinnen, da die Umsetzung originärer Spezifikationen bei vielen Anwendungen sehr stark automatisiert erfolgen kann. Ferner werden durch den ständigen Fortschritt in der Mikroelektronik immer mehr Komponenten für die Realisierung bestimmter Anwendungen zur Verfügung gestellt, so daß auch anwendungsspezifische Schaltungsentwürfe Komplexitäten erreichen, die noch vor wenigen Jahren allenfalls bei Schaltungsentwürfen für universelle Anwendungen realisiert wurden.

So sind Mikroprozessoren [Tannenbaum und Goodman, 1999] Chips, die universell für verschiedene Anwendungen geeignet sind. Dagegen sind beispielsweise MPEG-Encoder-Chips [Philips Semiconductors, 1998; Sony Semiconductor Company of America, 1999] anwendungsspezifische Chips zur Kompression von Videosequenzen nach dem MPEG-Standard [Jack, 1996; Reimers, 1997], die jedoch im Gegensatz zu derzeit handelsüblichen, universellen Mikroprozessoren in der Lage sind, Videosequenzen in *Echtzeit* zu komprimieren.

Auf einem Intel Pentium Prozessor sind $3,1 \cdot 10^6$ Transistoren integriert, und auf einem Intel Pentium Pro Prozessor sind $5,5 \cdot 10^6$ Transistoren integriert, und auf einem Intel Pentium II Prozessor sind $7,5 \cdot 10^6$ Transistoren integriert. Auf dem MPEG-2 Video Encoder Chip [Sony Semiconductor Company of America, 1999] der Firma Sony sind $4,5 \cdot 10^6$ Transistoren integriert. In [Keating und Bricaud, 1998] wird für das Jahr 2001 für integrierte Schaltungen – Systems-on-a-Chip (SoC) – eine Komplexität von $12 \cdot 10^6$ Logikgattern einer Taktfrequenz von 600 MHz und eine hypothetische Entwurfszeit von 500 Jahren für einen Entwerfer prognostiziert – bei einer fortgeschriebenen Entwurfsgeschwindigkeit von derzeit 100 Gattern pro Tag pro Entwerfer. Auch programmierbare Logikbausteine (FPGAs), die sich aufgrund ihrer Anwenderprogrammierbarkeit besonders für anwendungsspezifische Entwürfe bis zu Einzelstückzahlen eignen, dringen zunehmend in den Bereich von einer Million Gatter-Äquivalenten vor [Altera Corporation, 1998a; Xilinx, Inc., 1999].

Die technischen Möglichkeiten der Mikroelektronik stellen daher steigende Anforderungen an die Entwurfswerkzeuge, aber auch an die Entwurfsmethodik. Die Möglichkeit, ganze Systeme auf einem einzigen Chip unterbringen zu können, erschließt Anwendungen, bei denen die entworfenen Chips direkt mit realen Systemkomponenten interagieren. Für solche Chips müssen robuste Spezifikationen erstellt werden, die durch Simulation vor der Fertigung der Chips im System validiert werden müssen.

3.1 Kontext

Die vorliegende Arbeit ist im Kontext des Entwurfs digitaler Schaltungen entstanden. Inspiriert durch ALVINN (≈ *Autonomous Land Vehicle in a Neural Network*) [Pomerleau, 1989, 1993] wurde in Anlehnung an ALVINN ein Konzept für ein digitales System zur Echtzeitverarbeitung von Videobildern entworfen [Larsson *et al.*, 1996, 1997a,b]. Die Hardware-Realisierung

eines neuronalen Netzwerks zur Echtzeit-Bildklassifikation – NeNEB-Chip – wurde im Rahmen der Diplomarbeit [Krol, 1996] durchgeführt. Das konzipierte System schließt neben einem neuronalen Netzwerk zur Klassifizierung von Bildern auch die digitale Vorverarbeitung analoger Videosignale mit ein. Erste Untersuchungen und Hardware-Implementierungen zur digitalen Verarbeitung von Schwarz-Weiß-Videobildern wurden im Rahmen einer Studienarbeit [Hahn, 1995] durchgeführt. Unterstützt durch die bidirektionale Schnittstelle *VidTrans* [Larsson, 1996] zum Datenaustausch von Farbvideosignalen zwischen analogen Videogeräten und Computersimulationen sind im Rahmen einer Studienarbeit [Jürgens, 1996] und einer Diplomarbeit [Jürgens, 1999] Software-Implementierungen von PAL-Encodern und PAL-Decodern entstanden. Auf Ba-

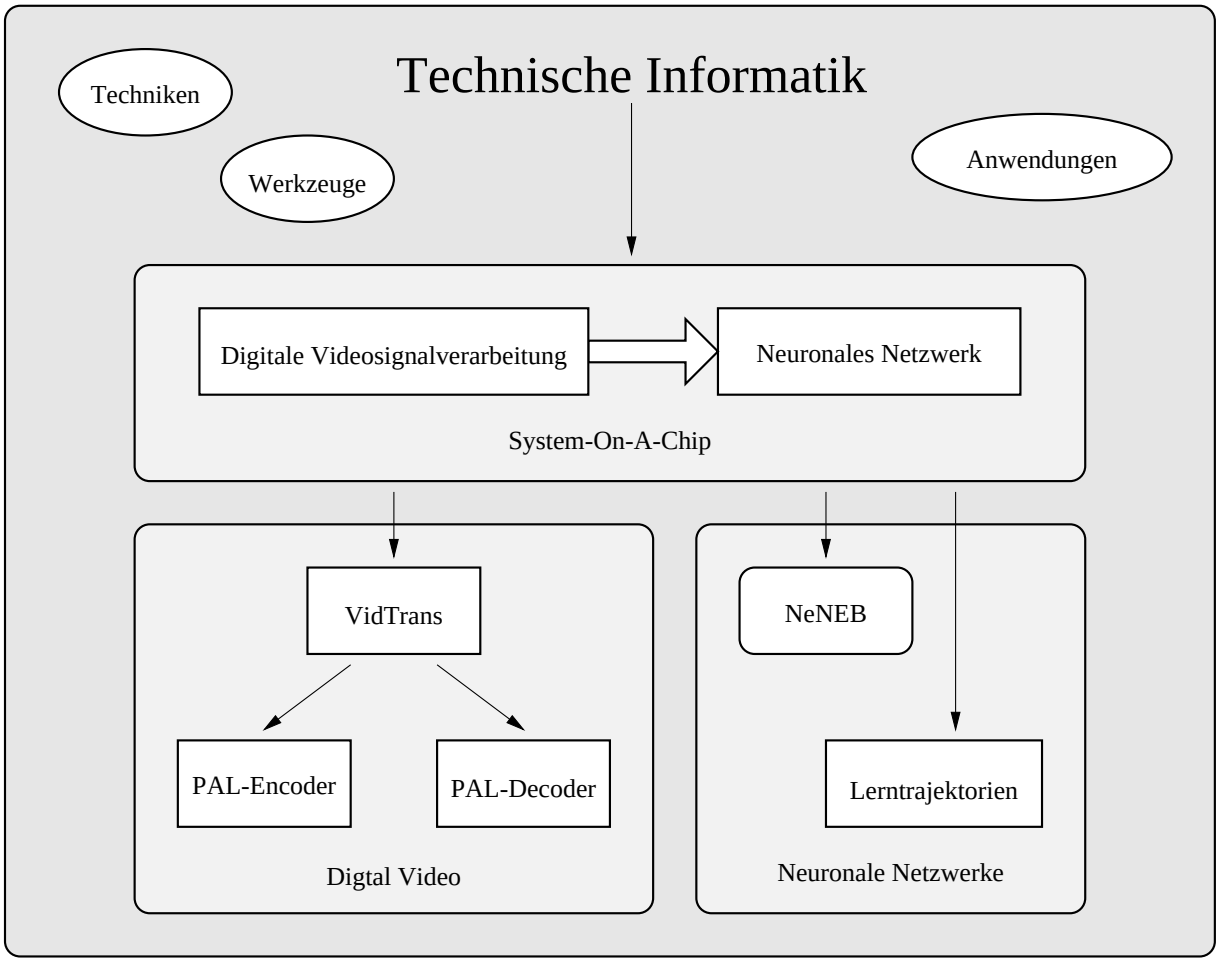


Abbildung 3.2: **Rahmen.** *Ausgehend von einem Rahmensystem zur Echtzeitverarbeitung von Videobildern auf Basis eines neuronalen Netzwerks wurden verschiedene Systemkomponenten implementiert, die oberflächlich betrachtet zunächst den Anschein erwecken, wenig Gemeinsamkeiten aufzuweisen. Im Kontext des Entwurfs digitaler Schaltungen wird jedoch deutlich, daß für den Entwurf digitaler Systeme übergeordnete Methoden erforderlich sind, die beim Entwurf von bestimmten Systemkomponenten Gemeinsamkeiten aufweisen.*

sis der im Rahmen der betreuten Arbeiten gewonnenen Erkenntnisse wurde zum ersten Mal ein rein digitaler PAL-Encoder [Larsson, 1999b] in einem einzigen FPGA als Hardware implementiert. Dadurch konnte der gewählte Ansatz der vorliegenden Arbeit bestätigt werden. Die Zusammenhänge zwischen den verschiedenen Teilaufgaben sind in Abbildung 3.2 skizziert.

Bei der Implementierung der unterschiedlichen Systemkomponenten wurde festgestellt, daß geeignete Methoden zur Validierung dieser Komponenten auf der Systemebene fehlen. Im Rahmen der vorliegenden Arbeit wurden daher neue Methoden zur Validierung des konzipierten Systems zur Echtzeitverarbeitung von Videobildern mit neuronalen Netzwerken erarbeitet. Diese Methoden mögen auf den ersten Blick vielleicht den Anschein erwecken, speziell auf die Validierung dieses Systems und seiner Komponenten zugeschnitten zu sein. Diese Beschränkung besteht jedoch nicht. Vielmehr sind diese Methoden in allgemeinerer Weise geeignet, Verhalten von Algorithmen und Systemen zu beobachten und zu untersuchen, um dadurch *robuste Spezifikationen* erstellen zu können.

3.2 Hintergrund

Die Technische Informatik stellt Methoden, Werkzeuge und Techniken für die Realisierung komplexer mikroelektronischer Systeme zur Verfügung. Der ständige Fortschritt der Mikroelektronik eröffnet dabei immer neue Anwendungen durch Erhöhung der Integrationsdichte mikroelektronischer Bauelemente und durch Reduzierung von Signallaufzeiten.

Höhere Integrationsdichte und geringere Signallaufzeiten sind die Hauptgründe für die zunehmende Steigerung der Verarbeitungsleistung mikroelektronischer Systeme. Höhere Integrationsdichte erhöht die Anzahl der Komponenten, die für die Realisierung mikroelektronischer Systeme zur Verfügung stehen. Für den Entwurf mikroelektronischer Systeme bedeutet das aber, daß eine steigende Zahl interagierender Komponenten zu handhaben ist. Die Steigerung der Verarbeitungsgeschwindigkeit – durch Verringerung der Signallaufzeiten – ermöglicht zunehmend die Realisierung rechenintensiver Algorithmen. Für die Validierung mikroelektronischer Systeme bedeutet das aber, daß die Simulationszeit für die Validierung steigt und daß die Menge zu handhabender Simulationsdaten ebenfalls steigt.

Die Abbildung einer abstrakten Beschreibung eines digitalen Systems auf einen Satz verfügbarer Komponenten ist mit Hilfe geeigneter Entwurfswerkzeuge zwar weitgehend automatisch möglich. Aber die Erstellung und Validierung einer originären Spezifikation, deren Weiterverarbeitung nach der Validierung weitgehend automatisch erfolgt, erfordert zeitintensiven Personeneinsatz. Daher sind auch geeignete Verfahren zur Unterstützung für die Erstellung einer robusten originären Spezifikation erforderlich. Im Kapitel 4, S. 77) werden solche Unterstützungsverfahren eingeführt.

Wie in [Keating und Bricaud, 1998] aufgeführt, ist die Realisierung eines Chip-Prototypen essentiell für die Verifikation der Robustheit des Entwurfs und der Korrektheit der originären Spezifikation, da zwar 90% neuer Chips beim ersten Versuch funktionieren, aber nur 50% der Chips korrekt im System funktionieren – diese These wird auch als *90-50-Regel* bezeichnet. Die Realisierung eines Prototypen setzt aber eine – zumindest ansatzweise – funktionsfähige Verhaltensbeschreibung voraus.

Erfordern technische Randbedingungen plattformspezifische Spezifikationsänderungen, wie etwa der Wechsel von Fließkommazahlendarstellung zu Festkommazahlendarstellung beim Übergang von einer Software-Implementierung zu einer Hardware-Implementierung, so können die entwickelten Verfahren zur Validierung *funktionaler Ähnlichkeit* der unterschiedlichen Spezifikationen eingesetzt werden. Funktionale Ähnlichkeit bedeutet dabei, daß zwei Implementierungen zwar nicht identisches Verhalten zeigen, aber daß beide Implementierungen *robust* spezifiziert und für eine vorgesehene Aufgabe gleichwertig sind. Die Implementierungen müssen robust und zueinander kompatibel sein.

Die entwickelten Verfahren unterstützen die Erstellung robuster Spezifikation und die Evaluation von Parametersätzen *vor* einer endgültigen Implementierung schon in der Anfangsphase

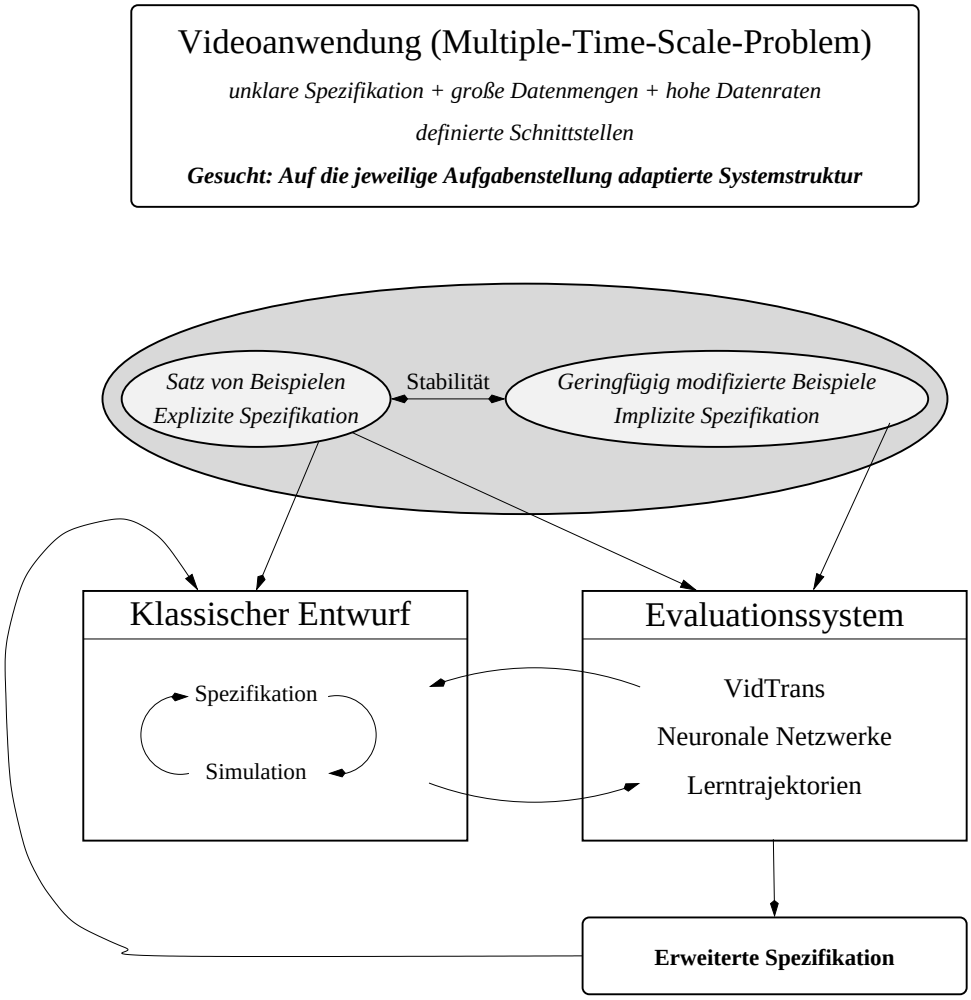


Abbildung 3.3: **Lösungsansatz.** *Anwendungen, bei denen sich die Informationsverarbeitung über mehrere Zeitskalen hinweg erstreckt, sind schwierig zu realisieren und erfordern besondere Methoden bei der Bewertung von Simulationsergebnissen.*

einer Implementierung durch Verhaltensbeobachtung. Die im Rahmen der Arbeit eingeführten *Lerntrajektorien* [Larsson, 1997, 1999a] können zur Beobachtung von lokalen Suchalgorithmen, wie dem Backpropagation-Algorithmus, eingesetzt werden. Die auf der Verwendung von einer auf programmierbaren Logikbausteinen basierenden Schnittstelle *VidTrans* [Larsson, 1996], zum Austausch von Videosignalen zwischen Anwendungsumgebung und Computersimulation, bildet eine Brücke, die besonders systemnahe Simulationen und die Bewertung von Simulationsergebnissen unter Echtzeitbedingungen ermöglicht und ist nicht auf die digitale Verarbeitung analoger Farbvideosignale beschränkt, sondern ist für ähnliche Systeme der digitalen Signalverarbeitung geeignet, die unter vergleichbaren Echtzeitbedingungen arbeiten.

3.3 Rahmensystem

Wesentlich bei dem im Rahmen dieser Arbeit konzipierten Rahmensystem ist, daß alle Systemkomponenten als rein digitale Module realisierbar sein sollten, um zu jeglichen *digitalen* Zielkomponenten (ASICs, EPLDs, FPGAs, ...) kompatibel zu sein, da der Entwurf digitaler Komponenten, im Gegensatz zum Entwurf analoger Komponenten, hochgradig technologieunabhängig erfolgen kann und digitale Komponenten darüber hinaus weitgehend unabhängig von elektrotechnischen Parametern sind. Daher ist die Verarbeitung analoger Videosignale Bestandteil des Systemkonzepts. Das Konzept eines rein digitalen Systems – abgesehen von ADC (*Analog Digital Converter*) und DAC (*Digital Analog Converter*) – eröffnet besondere Möglichkeiten. Der rein digitale Schaltungsentwurf bietet darüber hinaus den Vorteil, daß die Umsetzung von abstrakten Verhaltensbeschreibungen mit geeigneten Werkzeugen zur Logiksynthese weitgehend automatisiert erfolgen kann – im Gegensatz zu analogen Schaltungen. Dabei tritt jedoch der Systemaspekt in den Vordergrund, wogegen die technische Umsetzung in digitale Schaltungen in den Hintergrund tritt.

System-on-a-Chip-Kompatibilität

Der hohe Automatisierungsgrad der Logiksynthese erlaubt nun aber die Realisierung umfangreicher, digitaler Systeme, die zudem direkt in realen Umgebungen eingesetzt werden und nicht in einem Kontext mit starr definierten Schnittstellen. Vielmehr erfordert der Einsatz in realen Umgebungen besondere Toleranz gegenüber von Spezifikationen abweichenden Eingangsgrößen. Daher erfordert der Entwurf derartiger Systeme besondere Methoden. Derartige Methoden wurden im Rahmen der vorliegenden Arbeit entwickelt. Bei Verwendung von leistungsfähigen Entwurfswerkzeugen verlagert sich der Entwurf digitaler Schaltungen weg von der Handhabung technischer Details hin zum Systementwurf.

Die Realisierung eines System-on-a-Chip [Keating und Bricaud, 1998] erschließt den Einsatz als mobiles System. Die Verfügbarkeit hoher Verarbeitungsgeschwindigkeit zusammen mit geringen Abmessungen ist für viele technische Anwendungen erforderlich und typisch für den Einsatz integrierter Schaltungen.

Neuronale Netze und digitale Signalverarbeitung

Die Entwicklungen zu neuronalen Netzwerken und digitaler Signalverarbeitung im Zusammenhang mit der rein digitalen Verarbeitung analoger Videosignale weisen oberflächlich betrachtet zunächst keine offensichtlichen Gemeinsamkeiten auf. Vielmehr treten die Gemeinsamkeiten bei der Schaltungsimplementierung und Simulation derartiger Systeme zu Tage. Die Gemeinsamkeiten bestehen bei den zu implementierenden numerischen Operationen und in dem Umstand, daß die Bewertung von Simulationsergebnissen die Beobachtung von Zeitskalen über mehrere Größenordnungen hinweg erfordert und daher die Auswertung von Simulationsergebnissen erheblich erschwert.

Die Teilkomponenten – wie etwa Automaten, Register, Rechenwerke – eines übergeordneten Systems lassen sich sehr detailliert simulieren, und die Simulationsergebnisse können mehr oder weniger direkt, etwa anhand von Impulsdiagrammen, bewertet und validiert werden. Liefert das Zusammenspiel verschiedener Teilkomponenten jedoch ein Signal mit einer komplexen Zeitstruktur, wie beispielsweise ein Videosignal, so sind zur Bewertung der Simulationsergebnisse weitergehende Methoden erforderlich. Die Software-Implementierung eines digitalen PAL-Farbvideosignal-Encoders umfaßt etwa eintausend Zeilen in der Programmiersprache C. Die funktional ähnliche Hardware-Implementierung eines digitalen PAL-Farbvideosignal-Encoders [Larsson, 1999b] umfaßt etwa zweitausend Zeilen VHDL. Obwohl beide Implementierungen einen überschaubaren Umfang haben, war für beide die Validierung der Implementierungen durch Beobachtung ihres Verhaltens in realer Anwendungsumgebung unumgänglich.

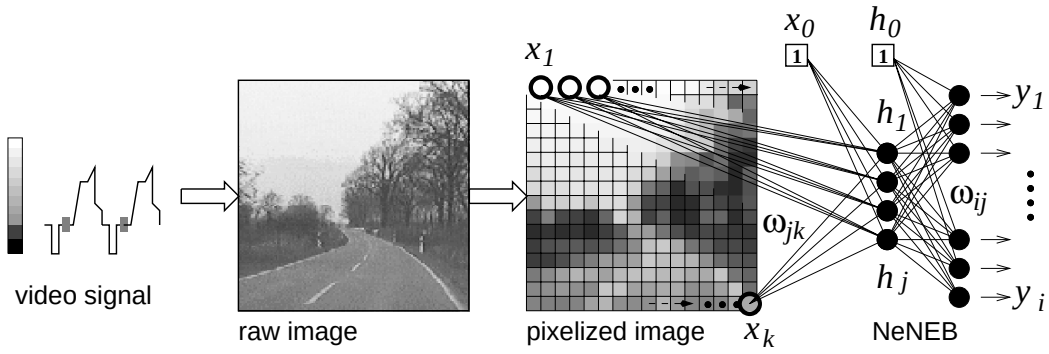


Abbildung 3.4: **Klassifizierungsschema von Bildern mit NeNEB** *[Larsson* et al.*, 1996]. Die Klassifizierung $\vec{y}$ eines Bildes mit k Bildpunkten x_k erfolgt durch Verknüpfung eines Bildpunktvektors $\vec{x}$ mit einem Datenstrom synaptischer Gewichte. Dazu akkumulieren acht verdeckte Neuronen h_j parallel Produkte $\omega_{jk} \cdot x_k$ Pixel für Pixel x_k, und zweiunddreißig Ausgangsneuronen y_i akkumulieren die Aktivitäten der verdeckten Neuronen h_j.*

Arithmetische Systemkomponenten

Die im Rahmen der Arbeit untersuchten Systemkomponenten weisen primär Gemeinsamkeiten hinsichtlich zu implementierender Rechenoperationen auf. Die Dendriten der Neuronen eines Backpropagation-Netzwerks entsprechen MAC-Einheiten eines FIR-Filters. Die nicht lineare Aktivierungsfunktion, ein Quotient mit Exponentialfunktion, der Neuronen kann ebenso wie die für die Quadraturamplitudenmodulation bei der Farbvideosignalverarbeitung benötigten trigonometrische Funktionen durch geeignete Näherungen effizient als *Look Up Table (LUT)* in Hardware realisiert werden. Dabei sind sowohl Exponentialfunktion als auch die trigonometrischen Funktionen mathematische Funktionen, die zwar auf Standardprozessoren relativ schnell und mit sehr hoher numerischer Genauigkeit berechnet werden, was aber für bestimmte Echtzeitanwendungen um Größenordnungen zu langsam ist.

Multiple-Time-Scale

Die wesentliche Gemeinsamkeit der untersuchten Systemkomponenten besteht jedoch darin, daß sich die Informationsverarbeitung über Zeitskalen mehrerer Größenordnungen (*Multiple Time Scale*) erstreckt, also nicht wie bei vielen anderen Anwendungen zeitlich lokal ist. Bei Videoanwendungen unter Echtzeitbedingungen sind hohe Datenraten zu verarbeiten. Hohe Datenraten über große Zeitskalen führen zu vergleichsweise großen Datenmengen, die es gegebenenfalls zu bewerten gilt. Darüber hinaus ist der Entwurf derartiger Systeme durch unklare Spezifikationen zusätzlich erschwert. Der klassische Schaltungsentwurf stellt einen wichtigen Teil bei dem Entwurf derartiger Systeme dar. Im Rahmen der vorliegenden Arbeit wird eine Erweiterung in Form von Evaluationsmethoden vorgestellt, die den Entwurf vereinfachen und so die Realisierung *robuster* Systeme unterstützen (Abb. 3.3). Das Attribut *robust* steht in diesem Zusammenhang für die Toleranz gegenüber von expliziten Spezifikationen abweichenden Eingangsdaten, aber auch für die Ausgabe von Daten, die von Zielgeräten nur geringe Toleranz erwarten.

Bewertung von Simulationsergebnissen

Die Ergebnisbewertung von digital erzeugten Videosignalen durch Darstellung auf Farbfernsehgeräten und Anschauen der Bilder ist ein sehr empirisches Vorgehen, das nicht sicherstellt, daß das erzeugte Videosignal auf jedem anderen Fernseher in gleicher oder zumindest ähnlicher Weise dargestellt werden kann. Diese Vorgehensweise ist zwar sehr empirisch, die Aufgabenstellung erfordert jedoch aus Mangel an besseren, nicht-empirischen Methoden zunächst diesen empirischen Ansatz. Dadurch wird jedoch grundsätzlich die Entwicklung eines generischen Modells eines Farbfernsehgeräts, also eines Moduls, das erzeugte Videosignale entgegennimmt und darstellt, ermöglicht, was im Rahmen der vorliegenden Arbeit mit *VidTrans* und digitalem PAL-Encoder und PAL-Decoder als eigenständige Module gezeigt werden konnte. Sowohl die Software-Implementierung funktionsfähiger PAL-Encoder und PAL-Decoder, wie auch die Hardware-Implementierung des PAL-Encoders *PalCo* [Larsson, 1999b] bestätigen den im Rahmen der vorliegenden Arbeit gewählten Ansatz.

3.4 Lösungsansatz

Der neue Ansatz der vorliegenden Arbeit liegt in der systemnahen Validierung von Systemen durch Beobachtung von Systemverhalten mit dem Ziel, robuste Systeme aus stabilen Komponenten zusammensetzen und Implementierungen auf unterschiedlichen Plattformen miteinander zu vergleichen. Der nicht-formale Ansatz wurde wegen fehlender formaler Spezifikation realer Systemumgebungen gewählt.

Werden in Zukunft verstärkt solche Systeme mit Techniken der Mikroelektronik realisiert, die direkt mit realen Systemkomponenten wechselwirken, so werden Methoden, wie sie im Rahmen der vorliegenden Arbeit eingeführt wurden, für den Entwurf digitaler integrierter Schaltungen notwendig werden. Die entwickelten Methoden sind für die Validierung solcher Systeme erforderlich, die direkt mit realen Systemkomponenten interagieren, die nicht oder nur unzureichend formal spezifizierbar sind.

3.4.1 NeNEB

Das durch *ALVINN* [Pomerleau, 1989, 1993] inspirierte Architekturkonzept von *NeNEB* [Larsson *et al.*, 1996, 1997a,b] und der digitale Chip-Entwurf [Krol, 1996] gaben Anlaß, neue Methoden zur Validierung ähnlicher digitaler Systeme zu entwickeln, aber auch die rein digitale Verarbeitung analoger Videosignale zu untersuchen [Jürgens, 1996, 1999] und exemplarisch eine entsprechende Hardware-Implementierung zu realisieren [Larsson, 1999b]. Die Untersuchungen zur digitalen Verarbeitung analoger PAL-Farbvideosignale wurden zunächst weitgehend unabhängig durchgeführt.

3.4.2 Lerntrajektorien

Die Beobachtung des Lernens eines Backpropagation-Netzwerks durch *Lerntrajektorien* [Larsson, 1997, 1999a] wurde primär zum Vergleich von Implementierungen auf unterschiedlichen Plattformen – Software vs. Hardware – eingeführt. Die Hardware-Implementierung von NeNEB [Krol, 1996] diente primär der Aufwandsabschätzung für ein solches System und stellt eine Vorstufe für ein Backpropagation-Netzwerk mit On-Chip-Lernen dar. Mit den bei der Implementierung von NeNEB gewonnenen Erkenntnissen und den Lerntrajektorien sind wissenschaftliche Grundlagen für die Implementierung und für weitergehende Untersuchungen geschaffen worden.

3.4.3 VidTrans

Bei den Untersuchungen und Software-Implementierungen zur digitalen Verarbeitung analoger Videosignale wurde die Notwendigkeit geeigneter Hilfsmittel [Larsson, 1996] deutlich. Diese zeigte sich auch bei exemplarischer Hardware-Implementierung des digitalen PAL-Encoders [Larsson, 1999b], für den der Implementierungsaufwand vergleichsweise gering ist. Die Implementierung wäre jedoch ohne den Austausch von analogen Videosignalen zwischen Computersimulationen und realen Systemkomponenten nicht möglich gewesen.

Kapitel 4

Entwickelte Verfahren

4.1 Bewertungsmethoden

Im diesem Kapitel werden die im Rahmen der vorliegenden Arbeit neu eingeführten Methoden zur Bewertung von Simulationsergebnissen im Kontext des Entwurfs digitaler Schaltungen vorgestellt. Die eingeführten Bewertungsmethoden dienen der Beobachtung von Systemverhalten auf einer hohen Abstraktionsebene, zur Validierung von Systemspezifikationen und zur Untersuchung der Robustheit von Spezifikationen und Implementierungen von Systemen im Vorfeld endgültiger Realisierung sowie der Validierung von Implementierungen auf unterschiedlichen Zielplattformen, wie etwa Implementierungen eines Algorithmus als Software und als Hardware – mit unterschiedlichen numerischen Vereinfachungen. Die Idee ist dabei, zunächst das Verhalten eines Systems vor der Implementierung zu untersuchen und Analysen unterschiedlicher Parametrierungen durchzuführen, um einen Ansatzpunkt für die Spezifikation eines robusten Systems zu erhalten. Im Gegensatz dazu dienen formale Verifikationsmethoden dem Nachweis funktiona-

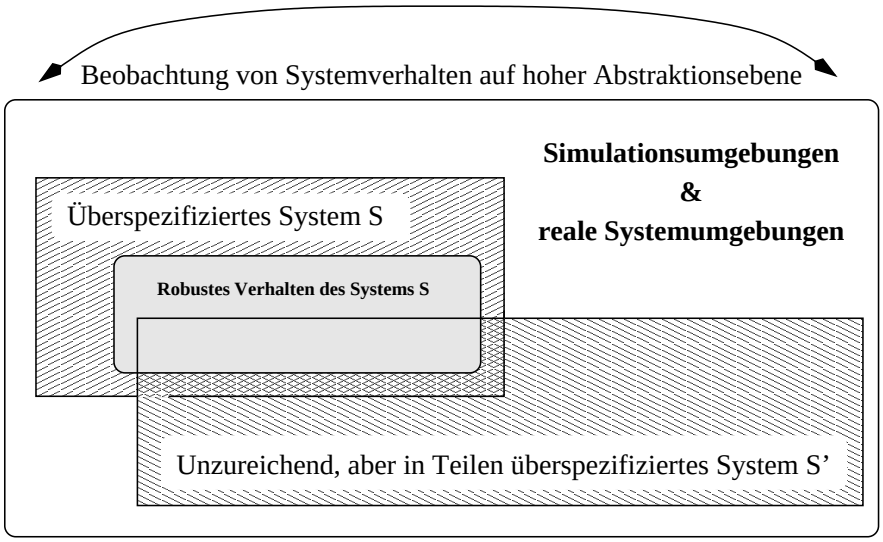


Abbildung 4.1: **System vs. Spezifikation**

ler Äquivalenz unterschiedlicher Repräsentationen eines Systems, wie etwa der Vergleich einer in einer Schaltungsbeschreibungssprache formulierten Verhaltensbeschreibung mit einer daraus synthetisierten Strukturbeschreibung in Form einer Gatternetzliste [Keating und Bricaud, 1998]. Dabei wird jedoch nicht überprüft, ob die Systemimplementierungen in realer Anwendungsumgebung robustes Verhalten zeigen [Kropf, 1998], denn dazu wären formal spezifizierte Verhaltensmodelle der realen Systemumgebung und aller Systemkomponenten erforderlich – ein Modell stellt jedoch immer eine Vereinfachung des modellierten, realen Systems dar.

4.1.1 Systementwurf

Die Implementierung von komplexen, informationsverarbeitenden Systemen erfordert spezifische Validierungsmethoden für die unterschiedlichen Abstraktionsebenen mit unterschiedlichem Detaillierungsgrad. Die direkte Beobachtung von – für ein in der Entwicklung befindliches System – charakteristischen Parametern leistet so einen möglichen Beitrag zur Validierung des vorliegenden Entwurfs. Auf der obersten Abstraktionsebene stellt die Beobachtung des Verhalten eines Gesamtsystems ein wichtiges Hilfsmittel für eine abschließende Validierung eines im allgemeinen unvollständig spezifizierten Systems dar. Insbesondere erlaubt die Verhaltensbeobachtung auf der obersten Abstraktionsebene Untersuchungen zur Robustheit eines implementierten Systems in Bezug auf Abweichungen der Implementierung gegenüber der ursprünglichen Spezifikation, aber auch in Bezug auf die Robustheit gegenüber Abweichungen von der Spezifikation der zu verarbeitenden Daten. Darüber hinaus erlaubt die Beobachtung von Verhalten auf der Systemebene die Bewertung von Entwurfsentscheidungen, wie etwa die Festlegung von internen Datenwortbreiten arithmetischer Rechenkomponenten. Außerdem erlaubt die Beobachtung von Systemverhalten, solche Entwurfsfehler aufzuspüren, deren Auswirkung erst durch Akkumulation über einen längeren Zeitraum auf der Systemebene erkennbar werden, wie beispielsweise Zahlenbereichsüberschreitungen von Zwischenergebnissen.

Je nach Systemanforderungen können solche Zahlenbereichsüberschreitungen vernachlässigbar sein oder aber Systemverhalten in unzulässiger Weise stören – das hängt vom betreffenden System ab und kann gegebenenfalls durch Beobachtung des Systemverhaltens bewertbar werden.

Bei der Realisierung großer, informationsverarbeitender Systeme wird eine komplexe Struktur aus untereinander wechselwirkenden Komponenten geringerer Komplexität zusammengesetzt. Wie komplex das Verhalten eines konstruierten Systems ist, hängt von dessen Größe und von den Kopplungen zwischen den zur Realisierung verwendeten Komponenten ab. Die Wech-

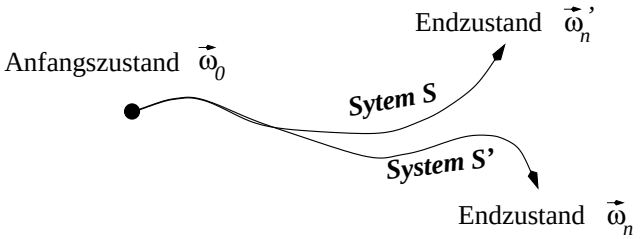


Abbildung 4.2: **Verhalten** *zweier Implementierungen S und S' eines Systems [Larsson, 1999a].*

selwirkungen können von der Art sein, daß aufgrund verdeckter Abhängigkeiten zwischen zahlreichen Systemkomponenten Simulationen erst dann aussagekräftige Ergebnisse liefern, wenn diese einen sehr großen Umfang erreichen. Der Umfang kann jedoch ein derartiges Volumen erreichen, daß eine manuelle Auswertung von Detailergebnissen unmöglich wird. Neuronale Netzwerke und digitale Videosignalverarbeitung stellen diesbezüglich im Zusammenhang mit dem Entwurf digitaler Systeme eine besondere Herausforderung dar.

Die im Rahmen der vorliegenden Arbeit im Kontext des Backpropagation Netzwerks eingeführte Methode der *Lerntrajektorien* visualisiert das Verhalten eines für das Verhalten des neuronalen Netzwerks charakteristischen Parametersatzes während des Lernvorgangs und dient damit der Validierung. Die ebenfalls im Rahmen der vorliegenden Arbeit entwickelte bidirektionale Schnittstelle *VidTrans*, die den Austausch von Videosignalen zwischen analogen Farbvideosignalen und Computersimulationen ermöglicht, erlaubt damit die Visualisierung des Verhaltens videosignalverarbeitender Geräte und deren Wechselwirkung mit einer realen Anwendungsumgebung.

4.1.2 Untersuchte Systeme

Informationsverarbeitende Systeme, die viele Parameter modifizieren, sind schwer zu validieren. Insbesondere ist die Validierung von Spezifikationen solcher Systeme und deren Robustheit in Bezug auf reale Signale einer Systemumgebung schwierig. Implementierungen eines informationsverarbeitenden Systems auf unterschiedlichen Plattformen – wie beispielsweise eine Software-Implementierung und eine Hardware-Implementierung eines numerischen Verfahrens – führen zu unterschiedlichem Verhalten der Implementierungen. Daher sind Methoden notwendig, die es erlauben zu entscheiden, ob unterschiedliche Implementierungen eines informationsverarbeitenden Systems gestellten Anforderungen entsprechen.

Die Einführung eines Zustandsvektors, der sich aus für ein betrachtetes System charakteristischen Parametern zusammensetzt, ermöglicht die Beschreibung von Systemverhalten als eine Folge von solchen Zustandsvektoren. Implementierungsunterschiede werden durch den Vergleich von Zustandsvektorfolgen als Abweichungen beobachtbar (Abb. 4.2).

Die Validierung von Systemen, deren Informationsverarbeitung über eine Zeitskala ausgedehnt ist, die sich über mehrere Größenordnungen erstreckt, ist dann besonders schwierig, wenn mehrere Zeitskalen zur Validierung von Bedeutung sind. Die digitale Verarbeitung von Videosignalen und Systeme zum Training von Backpropagation-Netzwerken sind Beispiele für *Multiple Time Scale Systeme*. In beiden Fällen stellt die Beobachtung des Systems ein Hilfsmittel zur Validierung des entworfenen Systems und dessen Spezifikation dar.

Um zu entscheiden, ob eine spezielle Implementierung eines Backpropagation Netzwerks in erhoffter Weise lernt, ist der Lernvorgang aufzuzeichnen und zu visualisieren. Eine etablierte Methode ist die graphische Darstellung des totalen Fehlers E (s. Gleichung (2.6), Abschnitt 2.2.1 auf S. 16) während des Lernens. Die in [Larsson, 1997] im Kontext der Hardware-Implementierung neuronaler Netzwerke neu eingeführte Methode der *Lerntrajektorien* visualisiert den Lernprozeß, dem hochdimensionalen Charakter des Trainings von Backpropagation-Netzwerken entsprechend, durch vergleichsweise direkte Beobachtung des Lernvorgangs auf einer höheren Abstraktionsebene, durch Beobachtung von Verhalten eines Gesamtsystems [Larsson, 1999a].

4.1.3 Backpropagation-Netzwerke

Ein neuronales Netzwerk, wie das Backpropagation-Netzwerk, ist ein informationsverarbeitendes System mit vielen Parametern, die beim Lernen gleichsam simultan von Lernschritt zu Lernschritt verändert werden. Dabei ist jeder Lernschritt durch einen eigenen Parametersatz – die synaptischen Gewichte des lernenden neuronalen Netzwerks – charakterisiert. Wird ein Parametersatz als Punkt in einem hochdimensionalen Zustandsraum interpretiert, so kann der Lernvorgang als Bewegung eines Punktes in dem hochdimensionalen Zustandsraum interpretiert werden. Diese Metapher liegt den Lerntrajektorien zugrunde. Neu ist dabei die direkte Darstellung der Bewegung eines Punktes in einem hochdimensionalen Zustandsraum in zwei Dimensionen in einer Weise, die Rückschlüsse auf die Beschaffenheit des hochdimensionalen Zustandsraums erlaubt.

Trainieren eines Backpropagation-Netzwerks mit dem gleichnamigen Lernalgorithmus stellt eigentlich das Lösen eines nicht-linearen Gleichungssystems mit einem numerischen Verfahren – dem Gradientenverfahren – dar. Der Backpropagation-Algorithmus ist ein lokales Suchverfahren, ein Minimierungsverfahren [Press *et al.*, 1992]. Die Veränderung der synaptischen Gewichte während des Lernens erfolgt in Richtung des negativen Gradienten des totalen Fehlers E (s. Gleichung (2.6), Abschnitt 2.2.1 auf S. 16). Es ist üblich, den totalen Fehler E während des Lernvorgangs aufzuzeichnen und zur Visualisierung des Lernens graphisch als Funktion des Lernschritts darzustellen. Der Lernprozeß kann sich verlangsamen, wenn der Gradient des Fehlers E in flachen Regionen (Plateaus) klein wird. Lokale Minima stören den Lernprozeß und können das Lernen sogar beenden, oder es kommt zu Oszillationen der synaptischen Gewichte um das betreffende lokale Minimum herum, bevor ein Optimum erreicht wird. Bei Backpropagation-Netzwerken kann es mehrere gleichwertige globale Minima, mit $E \approx 0$, geben [Rojas, 1993]. Solche Funktionen lassen sich einfach basierend auf Gleichung (4.20) konstruieren (S. 93).

Lerntrajektorien visualisieren den Veränderungsprozeß der synaptischen Gewichte eines Backpropagation-Netzwerks während der Lernphase durch Abbildung der Zustandsvektorsequenz im hochdimensionalen Raum auf eine zweidimensionale Darstellung. Lerntrajektorien können helfen, eine Vorstellung über die Gestalt hochdimensionaler Funktionen während der Extremumsuche zu gewinnen. Lerntrajektorien können Wege durch Zustandsräume mit beliebig vielen Dimensionen visualisieren. Durch Lerntrajektorien kann das Lernverhalten von Backpropagation-Netzwerken mit nur einigen synaptischen Gewichten aber auch mit hunderttausenden synaptischen Gewichten untersucht werden, wie mit entsprechenden Beispielen gezeigt wird (s. Abschnitt 4.2.3). Im Gegensatz dazu halten andere Methoden bei der Visualisierung hochdimensionaler Vorgänge meist fast alle bis auf ein oder zwei Parameter fest, und es werden nur die nicht festgehaltenen Parameter variiert.

Zudem ist die Visualisierungsmethode einfach zu implementieren. Lerntrajektorien visualisieren das Verhalten komplexer Systeme in einer direkten Art und Weise auf der Systemebene. Implementierungsdetails eines neuronalen Netzwerks spielen für die Anwendbarkeit der Lerntrajektorien keine Rolle, es ist nur erforderlich, auf den gesamten Satz der synaptischen Gewichte des betreffenden Netzwerks nach jedem Lernschritt zu zugreifen, um den Lernvorgang mit Hilfe von Lerntrajektorien visualisieren zu können. Lerntrajektorien, die zunächst eingeführt wurden, um das Lernen von Backpropagation-Netzwerken zu visualisieren, können auch zur Vi-

sualisierung des Verhaltens anderer Algorithmen eingesetzt werden, die in ähnlicher Weise einen Satz von Parametern modifizieren wie verwandte Optimierungsalgorithmen [Press *et al.*, 1992]. Darüber hinaus können Lerntrajektorien als Hilfsmittel für den Vergleich von Systemverhalten unterschiedlicher Implementierungen eines bestimmten Systems dienen, um funktionale Übereinstimmungen von Implementierungen auf unterschiedlichen Plattformen festzustellen.

4.1.4 Digitale Videosignalverarbeitung

Um zu entscheiden, ob eine spezielle Implementierung eines videosignalverarbeitenden Systems sich wie erwartet verhält, ist es unter Umständen notwendig, Simulationsergebnisse vom Umfang vieler hunderttausend Videosignalabtastwerte zur Bewertung heranzuziehen, die erst in ihrer Gesamtheit ein Bild oder eine Bildsequenz formen. Die bidirektionale Schnittstelle *VidTrans* zum Austausch von Videosignalen zwischen Simulation und realen Videogeräten [Larsson, 1996] hat sich bei der Entwicklung ausführbarer Spezifikationen eines digitalen PAL-Videosignalgenerators (PAL-Encoder) und eines PAL-Videosignal-Frame-Grabbers (PAL-Decoder) als unentbehrliches Hilfsmittel erwiesen. Ausführbare Spezifikationen eines digitalen PAL-Encoders und eines digitalen PAL-Decoders wurden zeitgleich zu einer vom Autor betreuten Studienarbeit [Jürgens, 1996] und vom Autor betreuten Diplomarbeit [Jürgens, 1999] entwickelt. So lagen unabhängig voneinander entwickelte, aber funktional ähnliche Module vor, die unterschiedliche Robustheit gegenüber bestimmten Videosignaleigenschaften zeigten. Diese Verhaltensunterschiede, die auf unterschiedliche Implementierungsdetails zurückzuführen sind, konnten nur mit Hilfe der bidirektionalen Videosignalschnittstelle detektiert und analysiert werden (s. Abschnitt 4.3, S. 115).

Erst durch die bidirektionale Schnittstelle *VidTrans* zwischen Simulation und analogen Videogeräten war die Entwicklung und Validierung der ausführbaren Spezifikationen des PAL-Encoders und des PAL-Decoders möglich. Diese Modelle konnten nur durch Austausch von Simulationsergebnissen und realen, analogen Videosignalen, also durch Bewertung der Simulationen auf der Systemebene, dahingehend verfeinert werden, daß ihr Verhalten dem realer Videogeräte sehr nahe kommt. Mit diesen Modellen ist die Grundlage für die Entwicklung generischer Videogeräte geschaffen, die im Kontext des Entwurfs digitaler videosignalverarbeitender Geräte für Validierungen erster Prototypen dienen könnten. Die gesammelten wissenschaftlichen Erkenntnisse und Erfahrungen wurden an einem praktischen Fallbeispiel überprüft. Als Fallbeispiel diente die Hardware-Implementierung eines rein digitalen PAL-Encoders *PalCo* [Larsson, 1999b]. Diese Hardware-Implementierung ist aus verschiedenen Gründen interessant. Obwohl es sich um einen relativ kleinen Entwurf in der Größenordnung von nur 10.000 FPGA-Gatter-Äquivalenten (90% Auslastung, FLEX 10K10, Typ EPF10K10LC84-3 [Altera Corporation, 1998b]) handelt, stellt diese Entwurfsaufgabe dennoch eine wissenschaftliche Herausforderung bezüglich der Entwurfs- und Validierungsmethodik dar. Trotz des vergleichsweise geringen Realisierungsaufwands ist dieser Entwurf geeignet, wichtige Aspekte (*Intellectual Property, Reusability, Multiple Time Scale*), die bei der Realisierung von SoC-Entwürfen [Keating und Bricaud, 1998] von Bedeutung sind, systematisch zu untersuchen. Nach Wissen des Autors stellt die im Rahmen der vorliegenden Arbeit realisierte Hardware-Implementierung eines vollständigen, rein digitalen PAL-Encoders in einem einzigen FPGA, der *alle* Komponenten beinhaltet, die für diese Signalverarbeitungsaufgabe erforderlich sind, die erste derartige Implementierung dar.

Wiederverwendbarkeit

Einige der zur Realisierung eines digitalen PAL-Encoders zu implementierenden Komponenten sind in leicht abgewandelter Form auch für die Hardware-Implementierung von Backpropagation-Netzwerken geeignet. Dies sind insbesondere die Sinus-LUT (Look Up Table) in Form eines FPGA-gerechten Schaltnetzes, das von einem DDS-Generator (Direkte Digital Synthese) zur Erzeugung des PAL-Farbträgersignals instantiiert wird, und der QAM-Modulator, der in Form einer MAC-Einheit (Multiply-and-Accumulate) realisiert wurde (s. Abschnitt 4.3.1, S. 120). Die Sinus-LUT-Komponente läßt sich als sigmoide Aktivierungsfunktion (2.3) eines Backpropagation-Netzwerks verwenden, wenn die Sinus-Wertetabelle durch eine Sigmoidfunktionstabelle ersetzt wird. Dagegen wurde in [Larsson *et al.*, 1997a] die Sigmoidfunktion durch fünf Geradenstücke approximiert, was aber gegenüber eines LUT-Schaltnetzes einen erhöhten Entwurfsaufwand bedeutete. Eine Sigmoidfunktion-LUT ist von der Logiksynthese, vielleicht entgegen erster Erwartung, gut minimierbar. Den trigonometrischen Funktionen $\sin(), \cos()$ und der Sigmoidfunktion $g()$ ist gemeinsam, daß sie bei hoher numerischer Genauigkeit, wie sie bei Software-Implementierungen zur Verfügung stehen, für eine Hardware-Implementierung ungeeignet sind. Der QAM-Modulator kann ohne Änderung als Neuron wieder verwendet werden, wenn nur die Ansteuerung in anderer Weise erfolgt. Der QAM-Modulator, d.h. die MAC-Einheit, wird bei jedem zweiten Takt mit $c_n := u \cdot \cos(2\pi \cdot f_{Farbträger} \cdot t_n)$ initialisiert, und im folgenden Takt wird $c_{n+1} := c_n \pm v \cdot \sin(2\pi \cdot f_{Farbträger} \cdot t_n)$ berechnet. Wird die MAC-Einheit mit dem Bias $c := \theta_0 (= \omega_{0j})$ initialisiert und dann in j folgenden Takten $c := c + x_j \cdot \omega_{ij}$ berechnet, so enthält die MAC-Einheit nach j Takten die gewichtete Summe $y_i = \sum_{j=0}^{k} \omega_{jk} \cdot x_j$. Dies entspricht dann genau den zur Implementierung der Vorwärtsphase des Backpropagation-Netzwerks notwendigen arithmetischen Operationen gemäß (2.2) in Abschnitt 2.2.1, S. 14. Es gibt also bei der Hardware-Implementierung von Algorithmen Gemeinsamkeiten zwischen zunächst sehr unterschiedlich erscheinenden Systemen, wie bei der vorliegenden Arbeit zwischen der digitalen Verarbeitung analoger Videosignale und dem Backpropagation-Netzwerk. Darüber hinaus bestehen aus numerischer Sicht inhaltliche Verwandschaften zwischen der digitalen Signalverarbeitung und neuronalen Netzwerken, die wohl in den Jahren von 1969 bis 1986 ein Grund für die Verlagerung von Forschungsaktivitäten weg von neuronalen Netzwerken hin zu adaptiver Signalverarbeitung waren (s. Abschnitt 2.1.1, S. 9).

4.2 Lerntrajektorien

Neuronale Netzwerke dienen unter Umständen nur als vorläufiger Prototyp, um zunächst nur die Machbarkeit eines Vorhabens zu zeigen und gegebenenfalls geeignete Parameter zu ermitteln. Die Realisierung eines Chips, etwa in Standardzellen-Technik, ist dazu unter Umständen aufgrund der hohen Herstellungskosten ungeeignet. Die deutliche Steigerung der Integrationsdichte von programmierbaren Logikbausteinen (FPGAs) in die Größenordnung von derzeit einigen hunderttausend Gatter-Äquivalenten, lassen jedoch zunehmend die Implementierung komplexerer arithmetischer Einheiten, und damit auch neuronaler Netzwerke, auf solchen Bausteinen zu.

Die Motivation für die Einführung der Lerntrajektorien zur Visualisierung des Lernens mit

dem Backpropagation-Algorithmus kommt aus dem Kontext des Entwurfs digitaler Informationsverarbeitungssysteme. Dabei dienen Lerntrajektorien der Kontrolle der Arithmetik, die im Fall einer Hardware-Implementierung im allgemeinen nur über eine geringe numerische Genauigkeit verfügt, und der Kontrolle des Lernens bei einer bestimmten Aufgabe mit im allgemeinen empirisch zu bestimmenden, den Lernvorgang steuernden Parametern, wie Zahl der Neuronen, Lernrate, Impulsparameter, Temperaturparameter (s. Abschnitt 2.2, S. 13). Darüber hinaus können Lerntrajektorien als Hilfsmittel dienen, um die Frage zu klären, ob eine für eine bestimmte Aufgabe gewählte Datenrepräsentation überhaupt zur Lösung mit einem implementierten Netzwerk geeignet ist. Die Kodierung der Lernmuster kann einen entscheidenden Einfluß auf die Gestalt des Suchraums haben und über die grundsätzliche Lernbarkeit eines Problems entscheiden, was beim Encoder-Problem besonders deutlich in Erscheinung tritt [Bakker *et al.*, 1993].

Selbstorganisierende Karten [Kohonen, 1972] können zur nachbarschaftserhaltenden Visualisierung hochdimensionaler Verteilungen in zwei Dimensionen verwendet werden. Sie können auch zur Überwachung (Monitoring) von Betriebszuständen von Systemen eingesetzt werden [Kohonen, 1996]. Ein Verfahren zur Visualisierung von Backpropagation-Lernen, das über die graphische Darstellung des Gesamtfehlers (2.6) als Funktion des Lernschritts hinausgeht, wird in [Rojas, 1994] vorgestellt. Die Visualisierung des Lernvorgangs kann unterschiedliche Zielsetzungen verfolgen. Sie kann der Beobachtung des Backpropagation-Algorithmus für Grundlagenuntersuchungen dienen, kann zur Steuerung des Backpropagation-Algorithmus beim Lernen verwendet werden oder, wie im Rahmen der vorliegenden Arbeit, zur Beobachtung der Auswirkungen von numerischen Vereinfachungen auf das Lernverhalten, wie sie für eine Hardware-Implementierung notwendig sind. Unter der Annahme, daß numerische Vereinfachungen die Fehlerfunktion stören und durch Quantisierungen zerklüften, geben Lerntrajektorien, als Wege auf der Fehlerfunktion, Einblicke in die Gestalt der hochdimensionalen Fehlerfunktion und deren ”Zerklüftungsgrad” aufgrund der numerischen Vereinfachungen.

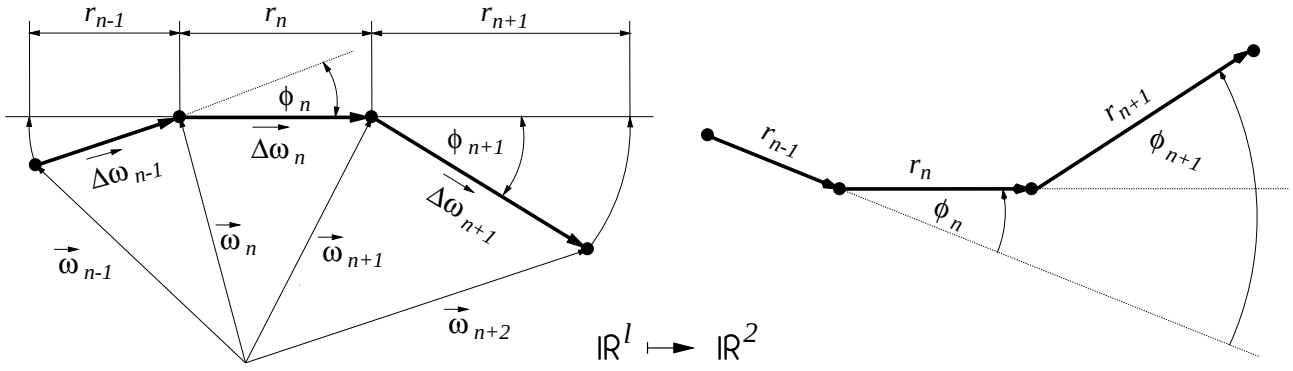


Abbildung 4.3: **Konstruktion der Lerntrajektorien.** *[Larsson, 1999a]*

4.2.1 Mathematische Beschreibung

Nach n Lernschritten befindet sich ein Backpropagation-Netzwerk in einem Zustand $\vec{\omega}_n$. Dieser Lernzustand $\vec{\omega}_n$ ist durch die Gesamtheit der synaptischen Gewichte $\omega_{ij}, \omega'_{jk}$ und Schwellen $\omega_{i0}, \omega'_{j0}$ (*Bias*) eines Netzwerks definiert. Der augenblickliche Lernzustand $\vec{\omega}_n$ eines Backpropagation-Netzwerks mit I Eingängen, J verdeckten Neuronen und K Ausgangsneuronen [K-J-I] ist durch einen Vektor mit

$$L = (I \cdot J + J) + (J \cdot K + K) \tag{4.1}$$

Komponenten in der Form

$$\vec{\omega}_n = (\underbrace{\omega_{10}, \ldots, \omega_{i0}}_{\approx\theta_I}, \underbrace{\omega_{11}, \ldots, \omega_{IJ}}_{\Omega_{IJ}}, \underbrace{\omega'_{10}, \ldots, \omega'_{J0}}_{\approx\theta_J}, \underbrace{\omega'_{11}, \ldots, \omega'_{JK}}_{\Omega_{JK}})_n = (w_1, \ldots, w_L)_n \tag{4.2}$$

beschrieben. Der euklidische Abstand r_n zwischen zwei aufeinander folgenden Lernzuständen $\vec{\omega}_n$ und $\vec{\omega}_{n+1}$ beträgt

$$r_n = |\Delta\vec{\omega}_n| \quad \text{mit} \quad \Delta\vec{\omega}_n = \vec{\omega}_{n+1} - \vec{\omega}_n. \tag{4.3}$$

Der Winkel $\Delta\phi_n$ zwischen zwei Differenzvektoren $\Delta\vec{\omega}_{n-1}$ und $\Delta\vec{\omega}_n$ beträgt

$$\Delta\phi_n = \measuredangle(\Delta\vec{\omega}_{n+1}, \Delta\vec{\omega}_n) = \arccos\left(\frac{\Delta\vec{\omega}_{n+1} \cdot \Delta\vec{\omega}_n}{|\Delta\vec{\omega}_{n+1}| \cdot |\Delta\vec{\omega}_n|}\right). \tag{4.4}$$

Durch Akkumulation der Winkel $\Delta\phi_n$ ergibt sich

$$\phi_n = \sum_{i=1}^{n} \Delta\phi_n. \tag{4.5}$$

Die beiden Größen r_n und ϕ_n können zu zweidimensionalen Vektoren $\vec{\rho}_n = (r_n, \phi_n)$ zusammengefaßt werden. Von Polarkoordinaten in kartesische Koordinaten transformiert, ergibt sich

$$\begin{array}{rcl} \vec{x}_n & = & (x_1, x_2)_n \\ & = & (r_n \cdot \cos(\phi_n), r_n \cdot \sin(\phi_n)). \end{array} \tag{4.6}$$

Eine Folge von n Lernschritten liefert folgende Stützpunkte

$$f_n = \{\vec{\omega}_0, \vec{\omega}_1, \ldots, \vec{\omega}_{n-1}, \vec{\omega}_n\} \tag{4.7}$$

einer Trajektorie $f(t) \in I\!R^L$ mit $t \in I\!R$ im Raum der L synaptischen Gewichte eines [K-J-L] Netzwerks, mit $\lim_{r_n \to 0} f_n = f(t)$. Die l-dimensionalen Stützpunkte f_n lassen sich als zweidimensionale Punktfolge

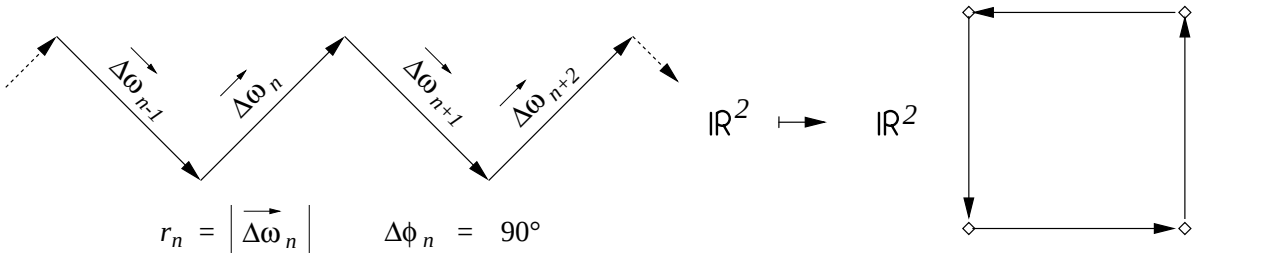
$$p_n = \{\vec{p}_0, \vec{p}_1, \ldots, \vec{p}_{n-1}, \vec{p}_n\} \qquad \text{mit} \qquad \left\{ \begin{array}{rcl} \vec{p}_0 & = & (0,0)_0 \\ \vec{p}_n & = & \vec{p}_{n-1} + \vec{x}_n \quad \text{für} \quad n > 0 \end{array} \right. \tag{4.8}$$

graphisch veranschaulichen, wenn aufeinander folgende Punkte $\{\vec{p}_n, \vec{p}_{n+1}\}$ verbunden werden. Die Form des sich dadurch ergebenden Linienzugs in $I\!R^2$ veranschaulicht die Form einer l-dimensionalen Trajektorie $f(t) \in I\!R^l$. Für $l = 2$ ist $I\!R^2$, so daß $\vec{x}_n = \vec{\omega}_n$ gilt und $p_n = f_n$. Schon für das XOR-Problem [Rumelhart und McClelland, 1986] ist $l = 9$. Geschlossene Teiltrajektorien werden als geschlossene Linienzüge in $I\!R^2$ dargestellt. Der Umkehrschluß ist jedoch nicht zulässig (s. Abbildung 4.4).

Unter Umständen ergibt sich eine bessere Darstellung der Lerntrajektorien, wenn nicht die Folge von *allen* Zustandsvektoren – von Lernschritt zu Lernschritt – visualisiert wird, sondern nur eine Teilmenge einer Stützpunktfolge p_n, wie z.B. statt $\{p_1, p_2, p_3, p_4, p_5, \ldots\}$ etwa $\{p_2, p_4, p_6, p_8, p_{10}, \ldots\}$, oder etwa von Lernzyklus zu Lernzyklus. Darüber hinaus könnte auch die Änderung der synaptischen Gewichte der verdeckten Neuronenschicht und der Ausgangsschicht den Lernvorgang durch zwei separate Lerntrajektorien visualisieren. Ebenso könnten die synaptischen Gewichte des Impulsterms (2.12) als Zustandsfolge mit zur Visualisierung des Lernvorgangs durch zusätzliche Lerntrajektorien herangezogen werden.

4.2.2 Semantik der Lerntrajektorienkrümmung

Der Krümmungsverlauf hochdimensionaler Trajektorien wird durch Überführung in die zweidimensionale Darstellung visualisiert und gibt so den Verlauf einer höherdimensionalen Fehlerfunktion wieder, indem der zurückgelegte Weg durch den Raum der Koeffizienten, die durch den Gradientenabstieg schrittweise verändert werden, dargestellt wird. Der Weg hängt vom Startpunkt ab und ergibt sich aus der Form der Fehlerfunktion, die den Gradienten bestimmt (s. Abbildung 4.9, S. 94).



sägezahnförmige, nicht-geschlossene Trajektorie ***zyklisch geschlossene Trajektorie***

Abbildung 4.4: **Sägezahnförmige, nicht-zyklische Trajektorie.**

Bezug zum Backpropagation-Algorithmus

Eine Lerntrajektorie wird durch den Backpropagation-Algorithmus zusammen mit den Lernmustern bestimmt. *Äußere Störungen* (Parameteränderungen) oder *innere Störungen* (z.B. Rundungsfehler) verändern den Verlauf der Trajektorie. Die Lerntrajektorie kann als Methode zur Visualisierung des Lernens und der Störung des Lernens verwendet werden. Der Vergleich von Lerntrajektorien eines bestimmten Netzwerks unter verschiedenen äußeren Randbedingung erlaubt einen direkteren Vergleich als etwa nur der Vergleich der Fehler E (2.6) in Abhängigkeit vom Lernschritt. Die Ursache für Stagnation des Fehlers E während des Lernens kann sowohl eine lokales Minimum als auch ein Plateau sein.

Läuft der Backpropagation-Algorithmus beim Lernvorgang über ein Plateau, also ein flaches Gebiet der Fehlerfunktion mit nahezu konstantem Gradienten, so wird sich der Gesamtgewichtsvektor (4.2) mit relativ konstanter Richtung relativ gradlinig fortbewegen. Eine gradlinige Bewegung des Gesamtgewichtsvektors wird als gradlinige Lerntrajektorie dargestellt (s. Abbildung 4.5). Erfährt der Lernvorgang durch Unebenheiten oder lokale Minima der Fehlerfunktion eine Richtungsänderung, so spiegelt sich eine solche Richtungsänderung als Knick in der korrespondierenden Lerntrajektorie wieder. Spiralförmige, kreisförmige oder oszillierende Bewegungen des Gesamtgewichtsvektors führen zu zyklischen Lerntrajektorien. Anhand des Gesamtfehlers E kann der Unterschied zwischen Plateau und lokalem Minimum unter Umständen nur schwer beobachtet werden, es sei denn, es kommt zu Oszillationen, die häufig deutlich am Gesamtfehler E ebenfalls als Oszillationen in Erscheinung treten.

4.2.3 Veranschaulichung

Die folgenden Beispiele veranschaulichen die Visualisierung höherdimensionaler Trajektorien in zwei Dimensionen und verdeutlichen Korrespondenzen zwischen höherdimensionalen Trajekto-

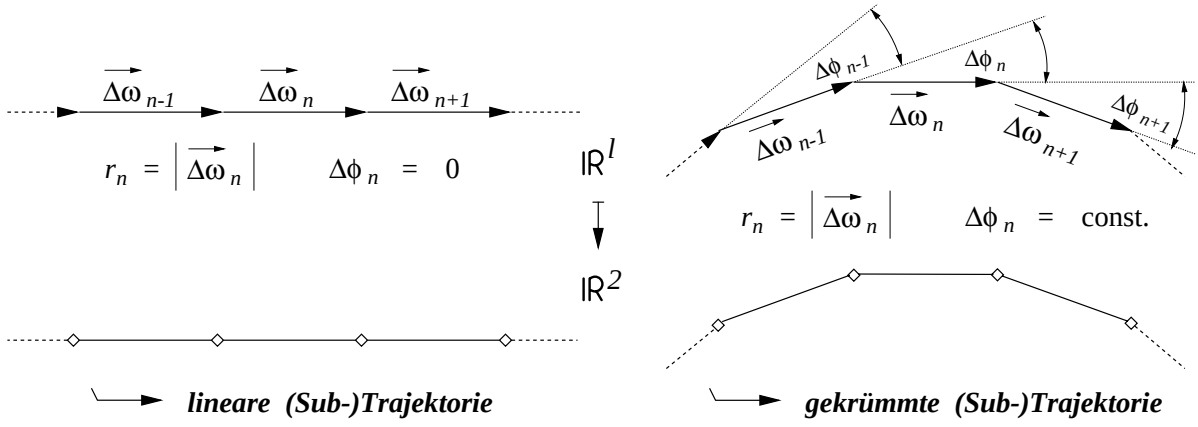


Abbildung 4.5: **Lineare und gekrümmte Trajektorien.**

rien und der zweidimensionalen Darstellung. Der einfachste Fall ist die gradlinige Trajektorie. Dieses erste Beispiel ist jedoch für die Visualisierung des Backpropagation-Lernens im Bereich von Plateaus wichtig. Das zweite Beispiel zeigt die Abbildung einer Trajektorie in drei Dimensionen, die "zick-zack-artig" entlang der Kanten eines Würfels verläuft. Das dritte Beispiel entspricht grundsätzlich dem zweiten Beispiel, nur mit dem Unterschied, daß die auf zwei Dimensionen abgebildete Trajektorie entlang der Kanten eines 50-dimensionalen Hyperwürfel verläuft. Ein weiteres Beispiel demonstriert die Abbildung der Lerntrajektorien von dreidimensionaler lokaler Suche nach einem Extremum einer gebrochen rationalen Funktion mit dem Gradientenverfahren auf die zweidimensionale Darstellung. Das Gradientenfeld wird dabei in unterschiedlicher Weise, durch Hinzufügen von Störtermen, gestört, was in der zweidimensionalen Darstellung deutlich wird. Anschließend wird Backpropagation-Lernen durch Lerntrajektorien visualisiert. Dazu werden zwei Backpropagation-Netzwerke von sehr unterschiedlicher Größe trainiert – ein Netzwerk mit einigen hundert synaptischen Gewichten zur einfachen Erkennung von Buchstaben und ein Netzwerk mit einigen hunderttausend synaptischen Gewichten zur Erkennung von Graubildern – jeweils mit verschiedenen arithmetischen Vereinfachungen. Ferner wird das Training des N-2-N-Encoder-Problems mit unterschiedlichen Parametern durch Lerntrajektorien visualisiert. Das N-2-N-Encoder-Problem ist sehr gut für derartige Untersuchungen geeignet, da es sich sehr einfach durch geringfügige Änderungen von einem durch ein Backpropagation-Netz leicht erlernbaren Problem zu einem, bei endlicher numerischer Genauigkeit, sogar bei Fließkommadarstellung nicht mehr erlernbaren Problem variieren läßt [Bakker *et al.*, 1993], obwohl Sätze synaptischer Gewichte als Lösungen existieren.

Gradlinige Trajektorie

Ein Beispiel für eine gradlinige Trajektorie $\vec{\omega}_n$ in einem beliebig dimensionalen Raum wäre etwa $\vec{\omega}_n = n \cdot \vec{e}_{x_i}$ mit $n = 1, 2, 3, \ldots$ in Richtung $\vec{e}_{x_i}$. In drei Dimensionen könnte also beispielsweise folgende gradlinige Trajektorie vorliegen

$$
\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
\vec{\omega}_1 &=& (0,1,0) & & & & & & \\
 & & & \Delta\vec{\omega}_2 &=& (0,1,0) & & & \\
\vec{\omega}_2 &=& (0,2,0) & & & & \Delta\phi_3 &=& \angle(\Delta\vec{\omega}_3, \Delta\vec{\omega}_2) = 0^o \\
 & & & \Delta\vec{\omega}_3 &=& (0,1,0) & r_3 &=& |\Delta\vec{\omega}_3| = 1 \\
\vec{\omega}_3 &=& (0,3,0) & & & & \Delta\phi_4 &=& \angle(\Delta\vec{\omega}_4, \Delta\vec{\omega}_3) = 0^o \\
 & & & \Delta\vec{\omega}_4 &=& (0,1,0) & r_4 &=& |\Delta\vec{\omega}_4| = 1 \\
\vec{\omega}_4 &=& (0,4,0) & & & & & & \\
\vdots & & \vdots & & & & & & \\
\vec{\omega}_n &=& (0,n,0) & & & & & &
\end{array}
\tag{4.9}
$$

Eine derartige Trajektorie wird in der zweidimensionalen Darstellung als gerade Trajektorie dargestellt, da $\Delta\varphi_n = 0$. Darüber hinaus ergeben sich äquidistante Stützpunkte P_n, da $r_n = 1$. Eine gradlinige Trajektorie entlang der Richtung $\vec{e}_{x_j}$ wird beschrieben durch

$$\begin{aligned} \vec{\omega}_n &= \sum_{i=1}^{n} c \cdot \vec{e}_{x_j} \\ &= n \cdot c \cdot \vec{e}_{x_j} \end{aligned} \tag{4.10}$$

Dabei sei c eine Konstante, welche die Schrittweite bestimmt, also die Abstände r zwischen aufeinander folgenden Differenzvektoren $\Delta\vec{\omega}_n$.

Würfel

Durch Akkumulation zueinander orthogonaler Vektoren läßt sich eine Folge von Vektoren konstruieren, die im Raum und in der Abbildung als Lerntrajektorie zu einer "zick-zack-förmigen" Trajektorie führen (Abb. 4.6). Bei diesem Beispiel werden drei orthogonale Einheitsvektoren $\vec{e}_{x_i}$ gemäß

$$\vec{\omega}_n = \sum_{i=1}^{n} \vec{e}_{x_i} \qquad \text{hier mit} \qquad n = 3 \tag{4.11}$$

akkumuliert. Diese Akkumulation resultiert in der Folge von Vektoren

$$\begin{array}{lllllll} \vec{\omega}_1 & = & (1,0,0) & & & & \\ & & & \Delta\vec{\omega}_2 & = & (0,1,0) & r_2 = |\Delta\vec{\omega}_2| = 1 \\ \vec{\omega}_2 & = & (1,1,0) & & & & \Delta\phi_3 = \measuredangle(\Delta\vec{\omega}_3, \Delta\vec{\omega}_2) = 90^o \\ & & & \Delta\vec{\omega}_3 & = & (0,0,1) & r_3 = |\Delta\vec{\omega}_3| = 1 \\ \vec{\omega}_3 & = & (1,1,1) & & & & \end{array} \tag{4.12}$$

zwischen aufeinander folgenden Differenzvektoren $\Delta\vec{\omega}_n$. Es handelt sich dabei also um eine Trajektorie mit drei Stützpunkten $\vec{\omega}_1$, $\vec{\omega}_2$, $\vec{\omega}_3$ in drei Dimensionen (Abb. 4.6). Die drei Vektoren

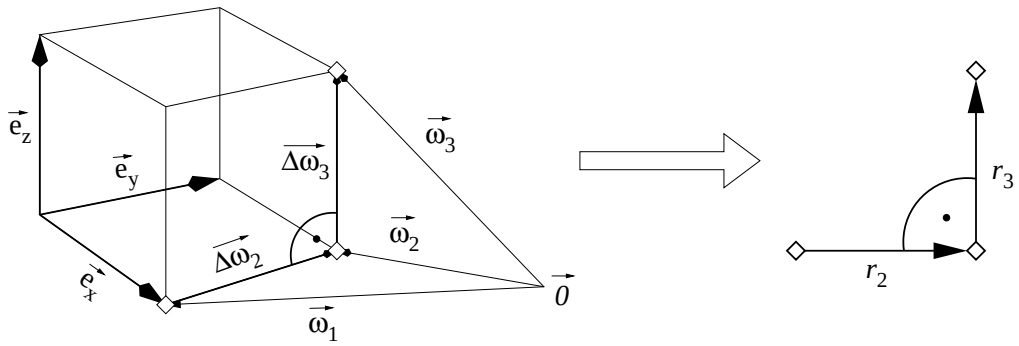


Abbildung 4.6: **Trajektorie in drei Dimensionen** *in [Larsson, 1997]. Die Trajektorie weist drei Stützpunkte $\vec{\omega}_1$, $\vec{\omega}_2$, $\vec{\omega}_3$ auf. Der Winkel zwischen den Differenzvektoren $\Delta\vec{\omega}_2$ und $\Delta\vec{\omega}_3$ beträgt 90^o, die Länge der beiden Differenzvektoren beträgt jeweils $r_2 = r_3 = |\vec{e}_{x_i}| = 1$. Die Zustandsvektoren $\vec{\omega}_1$, $\vec{\omega}_2$ und $\vec{\omega}_3$ gehen nicht vom Ursprung aus, um die Abbildung übersichtlicher zu halten.*

$\vec{\omega}_1$, $\vec{\omega}_2$, $\vec{\omega}_3$ stehen hier stellvertretend für Zustandsvektoren, die drei verschiedene Sätze synaptischer Gewichte repräsentieren. Der euklidische Abstand aufeinander folgender Differenzvektoren (4.12) ist hier die Länge der Einheitsvektoren (4.12), und die Winkel zwischen aufeinander folgenden Differenzvektoren betragen 90^0. Die resultierende Lerntrajektorie besteht aus zwei zueinander senkrechten Einheitsvektoren, wie in Abbildung 4.6 dargestellt.

Hyperwürfel

Dieses Beispiel demonstriert den Abbildungsmechanismus der Lerntrajektorien von einer hochdimensionalen "zick-zack-förmigen" Bewegung, der eine gradlinige Bewegung überlagert ist. Die Akkumulation von dreidimensionalen Einheitsvektoren gemäß (4.11) läßt sich leicht auf höherdimensionale Einheitsvektoren erweitern. Die resultierende Trajektorie verläuft dann entlang der Kanten eines Hyperwürfels. Dieses Beispiel dient der Veranschaulichung der Zusammenhänge zwischen einer Trajektorie im hochdimensionalen Raum und der korrespondierenden zweidimensionalen Darstellung der resultierenden Lerntrajektorie. Die Teilstücken $\Delta\vec{\omega}_n$ haben ebenfalls die Länge $|\Delta\vec{\omega}_n| = 1$. Mit $\Delta\phi_n = 90^o$ wird die Lerntrajektorie für $n > 4$ als Quadrat (bei gleicher Skalierung von X-Achse und Y-Achse) dargestellt werden. Ist $0 \neq \Delta\phi_n \neq 90^o$, so ergibt sich eine zyklische Figur. Wird zu (4.11) ein gradliniger Teil (4.10) in der Art

$$\vec{\omega}_n = \sum_{i=1}^{n} (\vec{e}_{x_i} + c \cdot \vec{e}_{x_j}) \tag{4.13}$$

hinzugefügt, so ergibt sich eine Folge von Zustandsvektoren der Form

$$\begin{array}{rcl@{\quad}l@{\quad}l@{\quad}l@{\quad}l}
\vec{\omega}_0 & = & (0,0,0,\ldots,0) & + & (0,\ldots, & 0, & \ldots,0) \\
\vec{\omega}_1 & = & (1,0,0,\ldots,0) & + & (0,\ldots, & c, & \ldots,0) \\
\vec{\omega}_2 & = & (1,1,0,\ldots,0) & + & (0,\ldots, & 2c, & \ldots,0) \\
\vec{\omega}_3 & = & (1,1,1,\ldots,0) & + & (0,\ldots, & 3c, & \ldots,0) \\
 & \vdots & & & \vdots & & \\
\vec{\omega}_{i<j} & = & (1,\ldots,0) & + & (0,\ldots, & i \cdot c, & \ldots,0) \quad \text{für} \quad i < j \\
 & \vdots & & & \vdots & & \\
\vec{\omega}_{i\geq j} & = & (1,\ldots,0) & + & (0,\ldots, & 1 + i \cdot c, & \ldots,0) \quad \text{für} \quad i \geq j \\
 & \vdots & & & \vdots & & \\
\vec{\omega}_{n-3} & = & (1,\ldots,0,0,0) & + & (0,\ldots, & 1 + (n-3) \cdot c, & \ldots,0) \\
\vec{\omega}_{n-2} & = & (1,\ldots,1,0,0) & + & (0,\ldots, & 1 + (n-2) \cdot c, & \ldots,0) \\
\vec{\omega}_{n-1} & = & (1,\ldots,1,1,0) & + & (0,\ldots, & 1 + (n-1) \cdot c, & \ldots,0) \\
\vec{\omega}_n & = & (1,\ldots,1,1,1) & + & (0,\ldots, & 1 + n \cdot c, & \ldots,0)
\end{array} \tag{4.14}$$

Dabei bestimmt j in (4.13) die Position der akkumulierten Konstante c. Die damit erzeugten Zustandsdifferenzvektoren $\Delta\omega$ sind fast senkrecht zueinander. Daraus ergeben sich die in den Abbildungen 4.7 und 4.8 dargestellten Lerntrajektorien. Für $i \neq j$ sind die Differenzvektoren von der Form $\Delta\omega_{i\neq j} = \vec{e}_i + c \cdot \vec{e}_i$. Für $i = j$ ist der Differenzvektor jedoch $\Delta\omega_{i=j} = \vec{e}_i + (1+c) \cdot \vec{e}_i$.

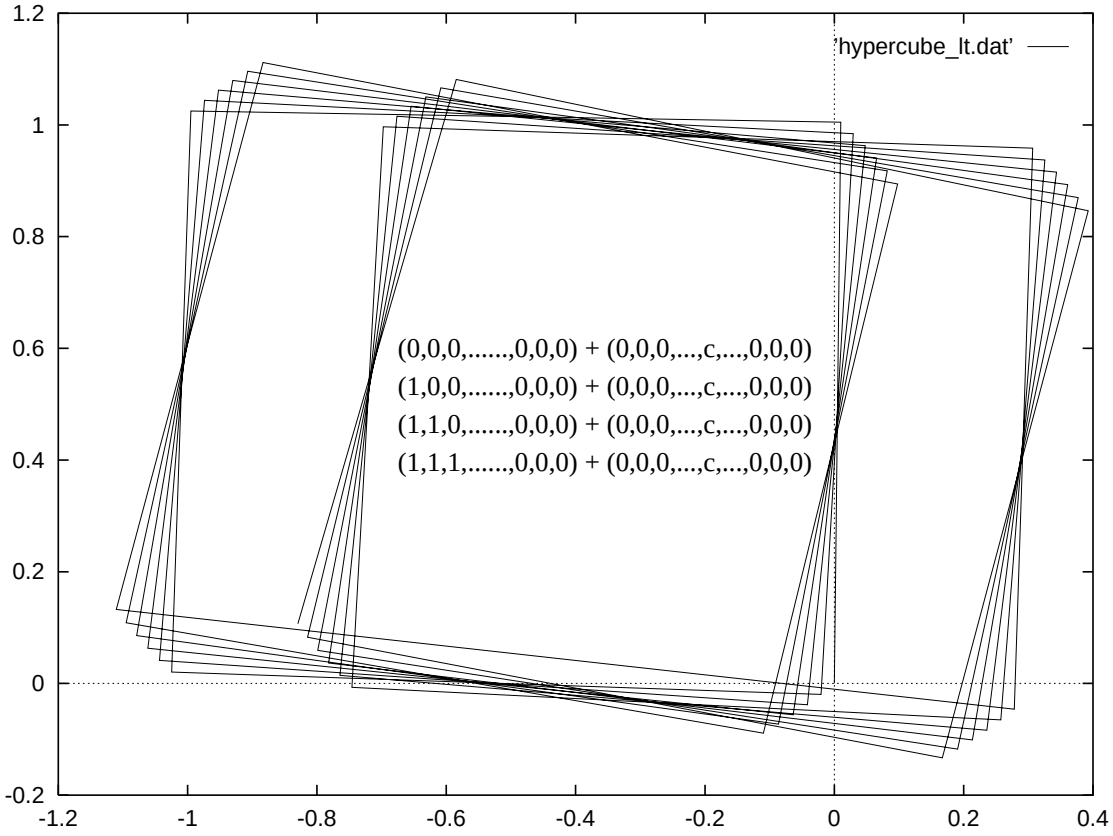


Abbildung 4.7: **Trajektorie** *entlang der Kanten eines 50-dimensionalen Hyperwürfels überlagert mit einer Drift in Richtung $\vec{e}_{x_j}$ [Larsson, 1999a]. Dabei ist $c = 0,1$ und $j = 25$ gewählt.*

Daher kommt es bei $i = j$ in der Darstellung der Lerntrajektorie zu einem sprunghaften Versatz der zyklischen Darstellung (s. Abbildung 4.7 und Abbildung 4.8).

Die Visualisierung mit Hilfe der Lerntrajektorien ist als ein Fenster in hochdimensionale Räume zu verstehen. Um mit Hilfe der zweidimensionalen Darstellung der Lerntrajektorien eine Vorstellung von der Gestalt der erzeugenden hochdimensionalen Trajektorien zu bekommen, ist es notwendig, eine Anschauung der Korrespondenzen zwischen den hochdimensionalen Trajektorien und ihrer Darstellung als zweidimensionale Lerntrajektorien zu erlangen. Die Intention dieses Beispiels ist die Veranschaulichung der Zusammenhänge zwischen hochdimensionalen Trajektorien und der zweidimensionalen Abbildung als Lerntrajektorien anhand eines Beispiels, das gerade solche Lerntrajektorien liefert, die auch für das Lernen mit dem Backpropagation-Algorithmus typisch sind. Die gemäß (4.14) konstruierten Zustandsvektorfolgen stellen eine Superposition von gradliniger Veränderung in eine Richtung $c \cdot \vec{e}_{x_j}$ und ständiger Richtungsänderung gemäß $\sum_{i=1}^{n} \vec{e}_{x_i}$ dar. Das Beispiel des 50-dimensionalen Hyperwürfels veranschaulicht so den Zusammenhang zwischen gradlinigen Trajektorien, wie sie beim Backpropagation-Lernen im Bereich von Plateaus auftreten, und zyklischen Trajektorien mit starken Richtungsänderungen, wie sie durch ”Unebenheiten” der Fehlerfunktion hervorgerufen werden. Darüber hinaus zeigt das Beispiel beim Übergang vom Beispiel des dreidimensionalen Würfels die Invarianz des

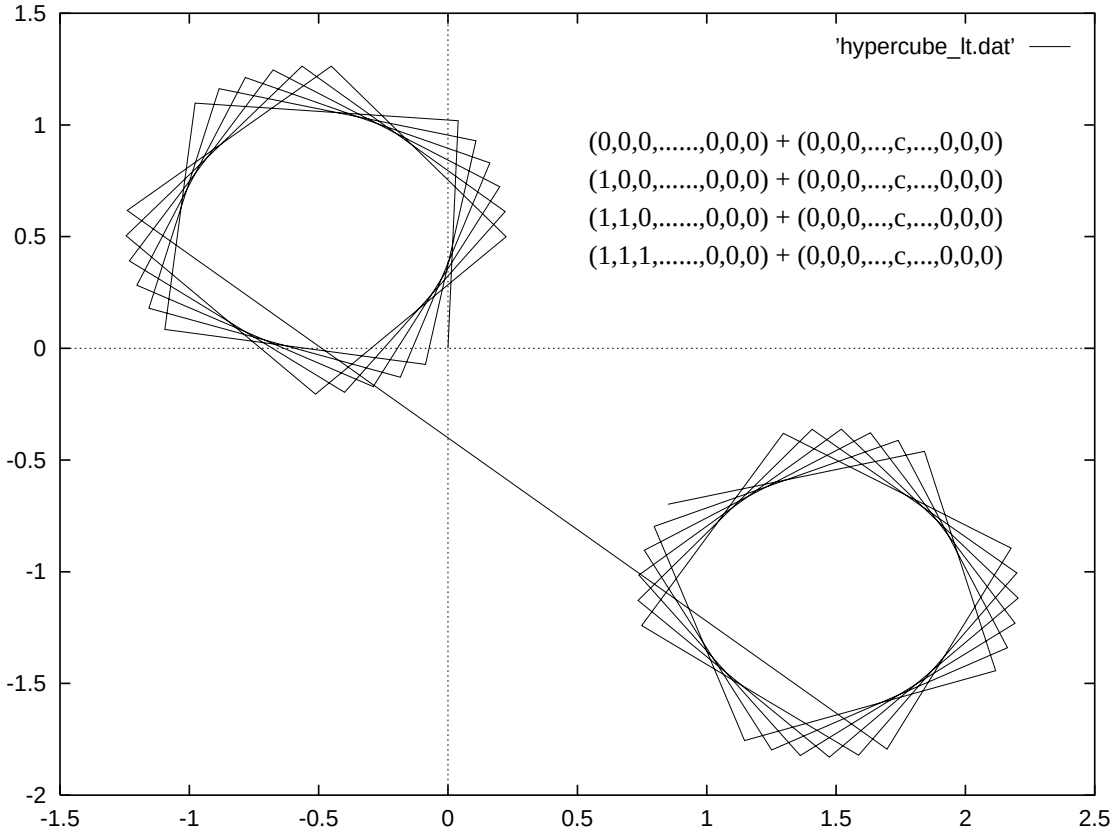


Abbildung 4.8: **Trajektorie** *entlang der Kanten eines 50-dimensionalen Hyperwürfels wie Abb. 4.7, aber mit* $c = 0, 2$ *statt* $c = 0, 1$. *In dieser Abbildung überlappen die zyklischen Figuren, die durch einen Versatz bei* $i = j$ *räumlich getrennt sind, nicht.*

Verfahrens gegenüber der Zahl der Dimensionen.

Es sei an dieser Stelle vorgreifend bemerkt, daß beim Backpropagation-Lernen solche Figuren, wie in den Abbildungen 4.7 und 4.8 dargestellt, tatsächlich auftreten. Es treten sogar kreisförmige Teil-Lerntrajektorien über mehrere Lernschritte auf – dabei sind dann die Längen r der Zustandsdifferenzvektoren und die Winkel $\Delta\varphi$ über mehrere Lernschritte nahezu konstant. Die Krümmung einer (zweidimensionalen) Lerntrajektorie kann als ein Maß für die Richtungsänderungen und damit für den Zerklüftungsgrad der Fehlerfunktion – im hochdimensionalen Raum verstanden werden. Der Abstand zwischen auf einander folgenden Stützpunkten kann als ein Maß für den Betrag des Gradienten – die Steilheit der Fehlerfunktion im hochdimensionalen Raum – verstanden werden. Unter der Annahme, daß es sich bei Plateaus um flache Bereiche mit konstantem Gradienten handelt, lassen sich so mit Hilfe der Lerntrajektorien Plateaus identifizieren. Dagegen ändert sich der Gradient im Bereich von Extrema, was die Form der Lerntrajektorien entsprechend beeinflußt. Um Bereiche mit konstantem r zu identifizieren, könnte statt der Lerntrajektorien auch die graphische Darstellung von r als Funktion der Lernschritte dienen. Die Darstellung in der Art der Lerntrajektorien scheint dem Autor jedoch eine direktere Abbildungsform hochdimensionaler Trajektorien zu sein.

Gebrochenrationale Funktion

Das folgende Beispiel demonstriert die Abbildung eines Pfades – einer Trajektorie f_n – von drei Dimensionen auf die zweidimensionale Darstellung durch Lerntrajektorien p_n. Die Trajektorie ist dabei das Ergebnis lokaler Suche mit dem Gradientenverfahren, wie sie auch beim Lernen beim Backpropagation-Algorithmus erfolgt. Dazu wird eine rationale Funktion mit einem globalen Maximum und lokalen Minima *konstruiert*. Der Vorteil des im folgenden beschriebenen konstruktiven Vorgehens ist, daß von Anfang an bekannt ist, wo das globale Maximum ist, wieviele lokale Minima existieren und wo sich diese befinden. Die konstruierte Funktion wird durch verschiedene numerische Vereinfachungen gestört. Die Störungen werden in dreidimensionaler Darstellung, im Gradientenfeld und anhand der Lerntrajektorien visualisiert. Die Gegenüberstellung der unterschiedlichen Darstellungen ist bei der konstruierten, dreidimensionalen Funktion problemlos möglich. Bei Anwendung des Gradientenverfahrens auf höherdimensionale Funktionen sind solche vergleichende Darstellungen nicht mehr möglich – nur die Visualisierung durch Lerntrajektorien ist für beliebig hoch dimensionale Funktionen geeignet. Um jedoch einen Nutzen aus Lerntrajektorien ziehen zu können, ist eine anschauliche Vorstellung der Abbildung von Trajektorien in hochdimensionalen Räumen auf die zweidimensionale Darstellung als Lerntrajektorie erforderlich. Die folgende rotationssymmetrische Funktion

$$f(r) = \frac{1}{1+r^2} \qquad \text{mit} \qquad r = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{4.15}$$

hat genau ein Maximum $f(0) = 1$ bei $r = 0$, also für $x_i = 0$, was leicht durch Kurvendiskussion verifizierbar ist. Außerdem ist

$$\lim_{|r \to \infty|} f(r) = 0, \tag{4.16}$$

und die Funktion hat keine lokalen Extrema. Die Rotationssymmetrie der Funktion (4.15) ist für die Wahl von Startpunkten für das Gradientenverfahren vorteilhaft. Durch die Quadratwurzel in (4.15) werden jedoch die Ausdrücke der partiellen Ableitungen $\partial / \partial x_i$ verglichen mit einer ähnlichen Funktion der Form

$$f(x,y) = \frac{1}{1+x^2+y^2} \tag{4.17}$$

etwas umfangreicher, weshalb die Funktion (4.17) im folgenden verwendet wird. Diese zweidimensionale Funktion hat bei $(0,0)$ ebenfalls genau ein Maximum $f(0,0) = 1$ (vgl. Anhang, S. 164). Durch additive Verschiebungskonstanten Δx und Δy läßt sich das Maximum der Funktion (4.17) von $(0,0)$ an jede beliebige Stelle $(-\Delta x, -\Delta y)$ mit

$$\tilde{f}(x,y) = \frac{1}{1+(x+\Delta x)^2+(y+\Delta y)^2} \tag{4.18}$$

verschieben. Durch Addition von Termen der Form (4.18) läßt sich eine Funktion mit beliebig vielen Maxima konstruieren. Durch einen Koeffizienten $c \in [0,\ldots,1[$ können die Maxima zu lokalen Maxima

$$\tilde{f}(x, y, \Delta x, \Delta y) = c \cdot \frac{1}{1 + (x + \Delta x)^2 + (y + \Delta y)^2} \tag{4.19}$$

gemacht werden. Für die Funktion

$$f(x, y) = \frac{1}{1 + x^2 + y^2} \qquad \text{ist} \qquad \vec{\nabla} f(x, y) = \left(\frac{-2x}{(1 + x^2 + y^2)^2}, \frac{-2y}{(1 + x^2 + y^2)^2} \right). \tag{4.20}$$

Diese Funktion und ihr Gradient lassen sich leicht auf beliebig viele n Dimensionen erweitern

$$f(x_1, \ldots, x_n) = \frac{1}{1 + \sum_{i=1}^{n} x_i^2} \qquad \text{und} \qquad \frac{\partial}{\partial x_i} f(x_1, \ldots, x_n) = \frac{-2x_i}{\left(1 + \sum_{i=1}^{n} x_i^2\right)^2}. \tag{4.21}$$

Mit Verschiebungskonstanten $(\Delta x_1, \ldots, \Delta x_n)$ ergibt sich für

$$\tilde{f}(x_1, \ldots, x_n, \Delta x_1, \ldots, \Delta x_n) = \frac{1}{1 + \sum_{i=1}^{n} (x_i + \Delta x_i)^2}, \tag{4.22}$$

und für die Komponenten $\frac{\partial}{\partial x_i}$ des Gradienten $\vec{\nabla} = (\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_i})$ gilt dann

$$\frac{\partial}{\partial x_i} \tilde{f}(x_1, \ldots, x_n, \Delta x_1, \ldots, \Delta x_n) = \frac{-2 \cdot (x_i + \Delta x_i)}{\left(1 + \sum_{i=1}^{n} (x_i + \Delta x_i)^2\right)^2}. \tag{4.23}$$

Zur Demonstration der Abbildung einer Trajektorie $f(t) \in I\!R^3 \rightarrow p_n \in I\!R^2$, die das Ergebnis eines Gradientenaufstiegs ist, wird die Funktion

$$\begin{array}{rcccc} F(x, y) & = & \underbrace{\tilde{f}(x, y, 0, 0)} & + & \frac{1}{4} \cdot (\sum_{i=0}^{3} \underbrace{\tilde{f}(x, y, \Delta x_i, \Delta y_i)}) \\ & & \text{Globales Maximum} & & \text{Lokale Maxima} \end{array} \tag{4.24}$$

mit

$$(\Delta x_i, \Delta y_i) \quad \in \quad \{(3, 3), (3, -3), (-3, 3), (-3, -3)\} \tag{4.25}$$

zugrunde gelegt. Die Höhe c der lokalen Maxima in (4.24) ist willkürlich auf $c = 1/4$ der Höhe ($= 1$) des globalen Maximums bei $(0, 0)$ festgelegt, weil damit eine übersichtliche Darstellung erreicht werden kann. Die Abbildung 4.9 zeigt den Gradientenaufstieg $f_n \in I\!R^3$ auf der Funktion

$F(x,y)$ mit Fließkommazahlendarstellung im ungestörten Gradientenfeld $\vec{\nabla}F(x,y)$ der Funktion (4.17), ausgehend von einem willkürlich gewählten Startpunkt $(x = 0.5, y = -10)$, zusammen mit der resultierenden Lerntrajektorie $p_n \in I\!R^2$. Das zugehörige Gradientenfeld $\vec{\nabla}F(x,y)$ ist in Abbildung 4.11 dargestellt. Der Startpunkt ist dabei so gewählt, daß der Einfluß eines lokalen Maximums deutlich wird. Bei Startpunkten, die auf der X-Achse oder auf der Y-Achse liegen, würde die Existenz der lokalen Maxima nicht direkt sichtbar in Erscheinung treten. Liegt ein Startpunkt jedoch zu sehr in der Nähe eines lokalen Maximums, so endet der Gradientenaufstieg im betreffenden lokalen Maximum. Dabei ist der Startpunkt mit $(x = 0,6785, y = -10)$ gegenüber Gradientenaufstieg mit dem Startpunkt $(x = 0,5, y = -10)$ nur geringfügig verschoben (Abb. 4.9). Die Verschiebung wurde empirisch so gewählt, daß der Einfluß des globalen Ma-

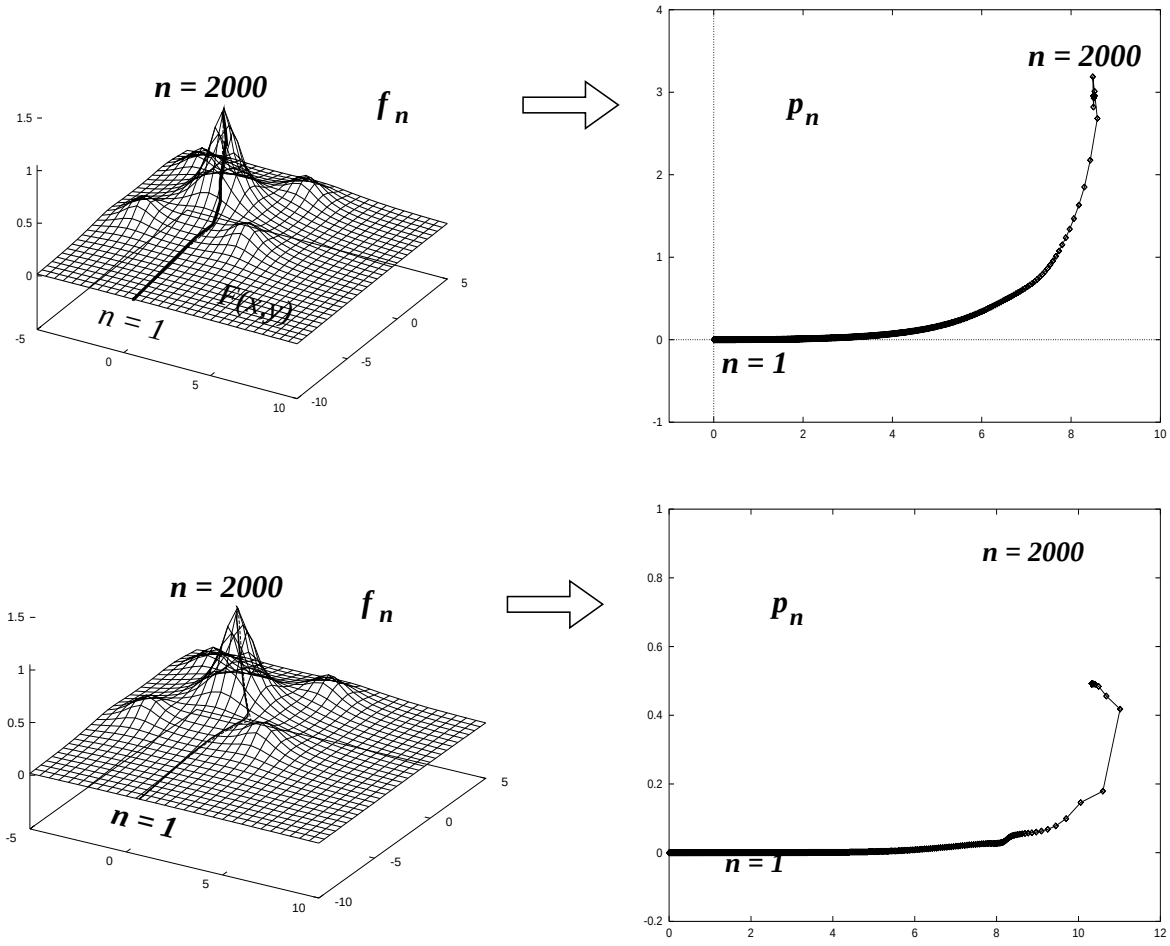


Abbildung 4.9: **Dreidimensionale Trajektorien f_n und zweidimensionale Lerntrajektorien p_n** *Bei der oberen Abbildung [Larsson, 1997] ist der Startpunkt $(x = 0.5; y = -10)$. Bei der unteren Abbildung ist der Startpunkt $(x = 0,6785, y = -10)$. Dadurch wird der Einfluß des lokalen Maximums bei $(3, -3)$ stärker deutlich. Für $x \in [0,680 \ldots 0,685]$ läuft der Gradientenaufstieg in das lokale Maximum bei $(3, -3)$. Die Parameter $\alpha = 0.1$ und $\eta = 0.9$ sind für beide Fälle gleich.*

ximums deutlich erkennbar ist. Bei Startpunkten, die auf Winkelhalbierenden des kartesischen Koordinatensystems liegen, würde die Existenz des globalen Maximums nicht deutlich sichtbar in Erscheinung treten, wenn es durch die lokalen Maxima bei $(3, 3)$, $(3, -3)$, $(-3, 3)$, $(-3, -3)$ verdeckt wird.

Eine typische Störung des Gradientenabstiegs, wie sie beim Lernen mit dem Backpropagation-Algorithmus durch zufällige Auswahl der Lernmuster auftritt, ist die Überlagerung des Gradientenabstiegs mit zufälligen Richtungsänderungen. Die Zufallskomponente kann die unerwünschte Terminierung des Lernvorgangs in einem *lokalen* Extremum verhindern, kann aber auch in unerwünschter Weise in ein lokales Extremum führen, wenn zufällige Richtungsänderungen zu sehr in die Nähe eines lokalen Extremums führen. Dies wird im folgenden durch zwei weitere Beispiele veranschaulicht. Bei der Implementierung des Gradientenaufstiegs auf der Funktion (4.24) werden für beide Koordinaten $x = x_1$ und $y = x_2$ zunächst die im n-ten Iterationsschritt durchzuführenden finiten Änderungen Δx und Δy, also Δx_i für $i = \{1, 2\}$, –

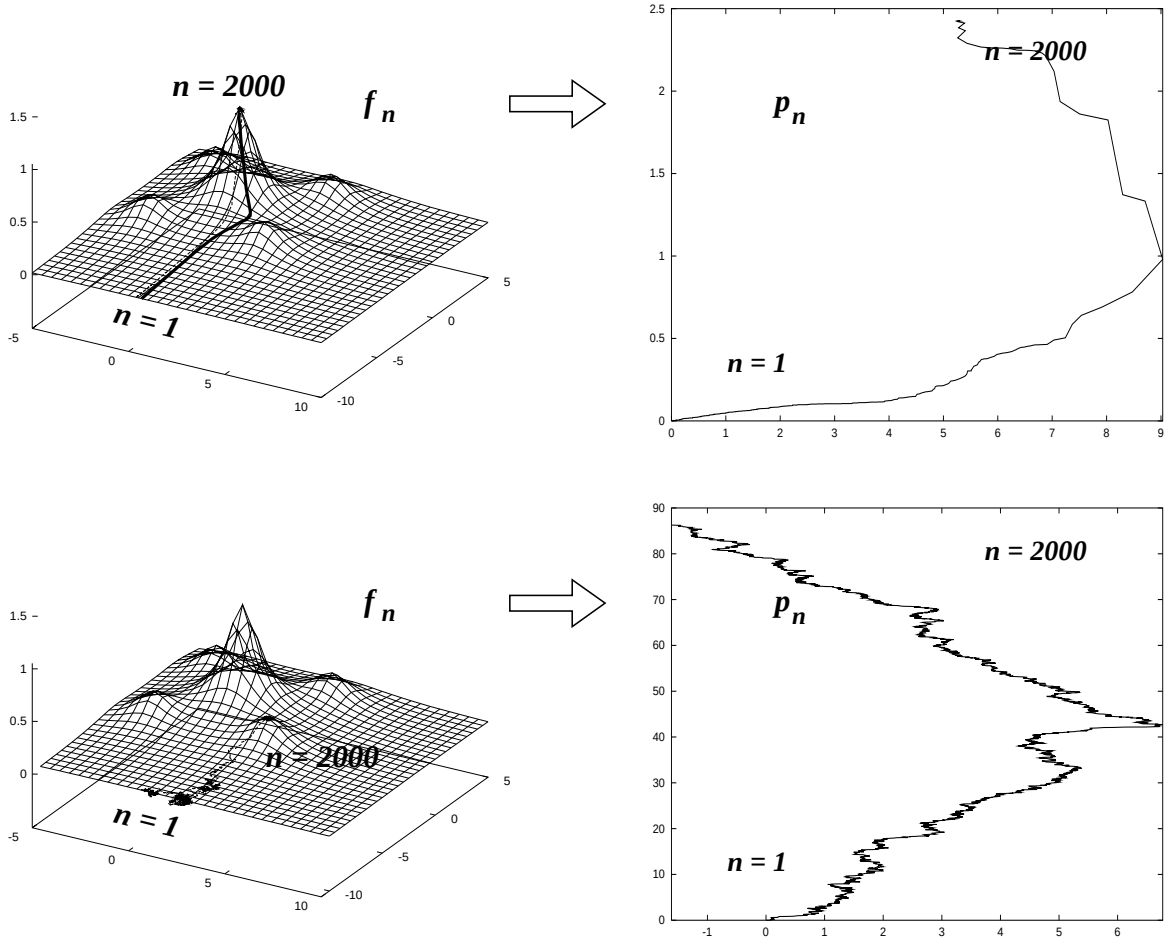


Abbildung 4.10: **Gestörter Gradientenaufstieg** *durch zur berechneten Änderung proportionale Zufallswerte (oben) und durch Addition von Zufallswerten (unten) zur berechneten Änderung.*

nicht zu verwechseln mit den Verschiebungskonstanten aus (4.24) – gemäß

$$(\Delta x_i)_n \quad = \quad \eta \cdot \frac{\partial}{\partial x_i} F(x_1, x_2) + \alpha \cdot (\Delta x_i)_{n-1} \qquad \text{mit} \qquad i = 1, 2 \tag{4.26}$$

berechnet. Dabei ist $(\Delta x_i)_{n-1}$ die im vorangegangenen Iterationsschritt berechnete Änderung. Damit wird eine Folge

$$\begin{array}{rcl} (x)_n & := & (x)_{n-1} + (\Delta x)_{n-1} \\ (y)_n & := & (y)_{n-1} + (\Delta y)_{n-1} \end{array} \tag{4.27}$$

erzeugt, die dann n Stützpunkte einer Trajektorie $f_n = F(x, y)_n$ liefert. Zur Störung des Gradientenaufstiegs wurden die Änderungen Δx und Δy gemäß

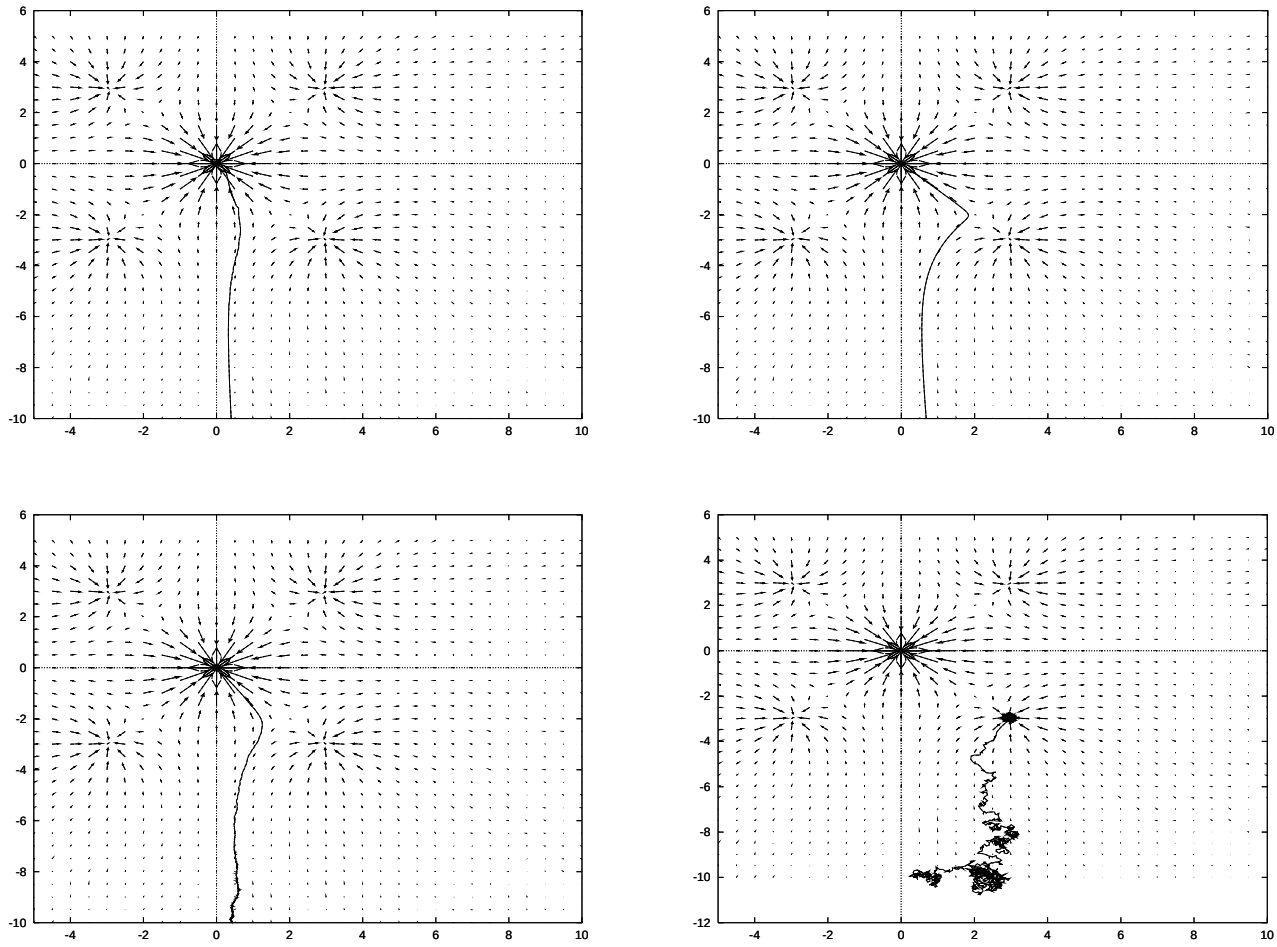


Abbildung 4.11: **Trajektorien in Gradientenfeldern** *zu den Abbildungen 4.9 (oben links), 4.9 (oben rechts), 4.10 (unten links), 4.10 (unten rechts).*

$$\begin{aligned} \Delta x &:= \Delta x + \Delta x \cdot \texttt{pmrand}\,(R) \\ \Delta y &:= \Delta y + \Delta y \cdot \texttt{pmrand}\,(R) \end{aligned} \tag{4.28}$$

gestört. Zur Erzeugung von Pseudozufallszahlen [Press *et al.*, 1992] wird die Funktion `rand` (R) der Standard-C-Library [Plauger, 1992] von der Funktion `pmrand` (R) aufgerufen. Die Funktion `rand()` liefert bei jedem Aufruf eine ganzzahlige Pseudozufallszahl aus dem Bereich `[0 ... RAND_MAX[`. Die Funktion `pmrand` (R) ist so skaliert, daß sie Pseudozufallszahlen im Fließkommaformat aus dem Bereich $]-R,\ldots,+R[$ liefert.

Als Startpunkt für den Gradientenaufstieg wurde der Punkt $(x = 0,4, y = -10)$ vom Autor gewählt, und es wurde $R = 1$ gewählt. Die Störung von (4.28) durch die zufällige Komponente ist proportional zu (Δx_i). Dadurch ergeben sich in Bereichen, in denen der Gradient klein ist, auch nur kleine zufällige Störungen. Das Ergebnis ist in Abbildung 4.10 dargestellt. In einem weiteren Beispiel wurde der Gradientenaufstieg additiv gemäß

$$\begin{aligned} \Delta x &:= \Delta x + \texttt{pmrand}\,(R) \\ \Delta y &:= \Delta y + \texttt{pmrand}\,(R) \end{aligned} \tag{4.29}$$

gestört. Als Startpunkt für den Gradientenaufstieg wurde der Punkt $(x = 0.4, y = -10)$ gewählt. Mit einem Parameter $R = 0,1$ läuft der Gradientenaufstieg bei dem gewählten Startpunkt in das lokale Maximum (s. Abb. 4.10) bei $(3,-3)$, mit $R = 0,01$ wird jedoch das globale Maximum bei $(0,0)$ erreicht (s. Abb. 4.10). In der Abbildung 4.11 sind die Gradientenfelder zusammen mit der Trajektorie für die Störungen (4.28) und (4.29) dargestellt.

4.2.4 Numerische Vereinfachungen

Wie schon eingangs im Kapitel 2 beschrieben, sind zur Hardware-Implementierung numerischer Operationen im allgemeinen numerische Vereinfachungen bezüglich der verwendeten Zahlendarstellung notwendig, um gegenüber einer entsprechenden Software-Implementierung auf einem Standard-Rechner eine Steigerung der Verarbeitungsgeschwindigkeit bei vertretbarem Schaltungsaufwand erreichen zu können. Numerische Näherung bedeutet in diesem Zusammenhang meist Verzicht auf die Fließkommadarstellung und Übergang zu einer Festkommadarstellung oder Ganzzahldarstellung.

Die für eine Hardware-Implementierung des Backpropagation-Algorithmus erforderlichen Operationen sind die Addition, die Multiplikation und die Sigmoidfunktion (s. Abschnitt 2.2). Die Addition ist im allgemeinen effizient – bezüglich der Verarbeitungsgeschwindigkeit und der für die Implementierung notwendigen Chip-Fläche – möglich. Bei der Multiplikation hängt der Implementierungsaufwand sehr stark von der gewählten Implementierungsalternative ab. Ein sequentieller Multiplizierer benötigt verglichen mit einem parallelen Multiplizierer deutlich weniger Chip-Fläche, weist aber auch eine entsprechend geringere Verarbeitungsgeschwindigkeit auf [Omondi, 1994].

Fließkommadarstellung

Simulationen mit Fließkommadarstellung (`float` in der Programmiersprache C) liegt die semilogarithmische Darstellung mit Mantisse m und Exponent e gemäß

$$f = m \cdot 2^e \qquad \text{mit} \qquad m \in [0, \ldots, 10[\qquad \text{und} \qquad e \in [-38, \ldots, +38] \tag{4.30}$$

zugrunde. Für Fließkommadarstellung (`double` in der Programmiersprache C) ist $e \in [-308, \ldots, +308]$. Die Fließkommadarstellung zeichnet sich – verglichen mit der Festkommadarstellung bei gleicher Operandenwortbreite – durch eine wesentlich bessere Zahlendarstellungsdynamik aus. Sie ist aber auch nur eine numerische Näherung reeller Zahlen, jedoch für viele Anwendungen eine hinreichend gute. Die Zahlenbereichsgrenzen der Fließkommadarstellung treten meist nicht in Erscheinung, wenn sich durchgeführte Berechnungen im zulässigen Zahlenbereich bewegen, auch wenn (akkumulierte) Rundungsfehler zu falschen (End-)Ergebnissen führen können. Die Implementierung von Rechenwerken mit Fließkommadarstellung hoher Genauigkeit, wie sie von Standardprozessoren verwendet werden, kommt für Hardware-Implementierungen meist nicht in Frage, wie in Kapitel 2 beschrieben.

Festkommadarstellung

Die Verwendung der Festkommadarstellung zur Implementierung neuronaler Algorithmen ist eine numerische Näherung der reellen Zahlen durch eine *endliche* Teilmenge der rationalen Zahlen. Die mathematische Formulierung des Backpropagation-Algorithmus basiert auf der Menge der reellen Zahlen. Der Backpropagation-Algorithmus spezifiziert Regeln, wie endliche Änderungen der synaptischen Gewichte zu berechnen sind, was für sich genommen schon eine numerische Näherung darstellt. Die Lernrate η stellt einen Parameter dar, mit dem sich der Übergang von infinitisimalen Änderungen zu endlich großen Änderungen einstellen läßt.

Die Festkommadarstellung stellt aufgrund der schlechteren Zahlendarstellungsdynamik eine wesentlich stärkere Störung des Backpropagation-Algorithmus dar als die Fließkommadarstellung. Die Festkommadarstellung ist jedoch zur Hardware-Implementierung besser geeignet als die Fließkommadarstellung. Da die Festkommadarstellung bei vergleichbarer Wortbreite eine schlechtere Darstellungsdynamik als die Fließkommadarstellung aufweist, kann es bei Festkommadarstellung leicht zum numerischen Überlauf kommen, wenn dies nicht durch Bereichsbegrenzungen verhindert wird. Zur Demonstration der Visualisierung des Backpropagation-Lernens durch Lerntrajektorien werden numerische Zwischenergebnisse, die mit Fließkommadarstellung errechnet werden gemäß

$$I(x) = \begin{cases} -1 \cdot (2^l - 1) & \text{für} \quad x < -1 \cdot (2^l - 1) \\ \dfrac{\lfloor (2^n - 1) \cdot x \rfloor}{2^n - 1} & \\ 2^l - 1 & \text{für} \quad x > 2^l - 1 \end{cases} \qquad \text{mit} \quad n = 15 \quad \text{und} \quad l = 4 \tag{4.31}$$

quantisiert, wodurch sich eine Art Festkommadarstellung ergibt. Dabei repräsentiert n die Nachkommastellen-Bits und l die Bits vor dem Dezimalkomma. Das Vorzeichen stellt ein zusätzliches Bit dar. Dieser Weg wurde hier aufgrund der einfacheren Handhabung, verglichen mit "echter" Zweierkomplementfestpunktdarstellung, wie sie für Hardware-Implementierungen verwendet wird, gewählt. Zur Validierung von Hardware-Implementierungen ist die System-Simulation ohnehin aussagekräftiger als detailgetreue Nachbildung einer Ziel-Hardware in einer Software-Simulation.

Logarithmische Quantisierung

Die Multiplikation $a \cdot 2^n$ einer ganzzahligen Dualzahl a mit einer ganzzahligen Zweierpotenz 2^n ist sehr einfach durch bitweises Verschieben der Dualzahl a um n Stellen zu realisieren. Eine Komponente, die diese Aufgabe parallel ausführt, wird als *Barrel-Shifter* bezeichnet [Weste und Eshraghian, 1993]. Es sei an dieser Stelle bemerkt, daß die Hardware-Implementierung eines Barrel-Shifters nicht in dem Maße effizienter als ein paralleler Multiplizierer ist, wie vielleicht zunächst angenommen werden könnte, wenn ein solcher aus bestehenden Multiplizierern, wie sie in Zellbibliotheken meist als fertige, flächenoptimierte Komponenten zur Verfügung stehen, zusammengesetzt wird, und ein Barrel-Shifter im Gegensatz dazu aus vielen kleinen Multiplexern zusammengesetzt werden muß, wodurch dann gegenüber dem Multiplizierer ein erhöhter Flächenbedarf durch einen erhöhten Verdrahtungsaufwand zu berücksichtigen ist. Die Reduzierung der Wortbreite eines Faktors durch Repräsentation dieses Faktors durch einen Exponenten ist jedoch für eine Hardware-Implementierung vorteilhaft, wenn Operanden über Chip-Anschlüsse (Pads) von außen an ein Rechenwerk herangeführt werden müssen. Dem Problem der

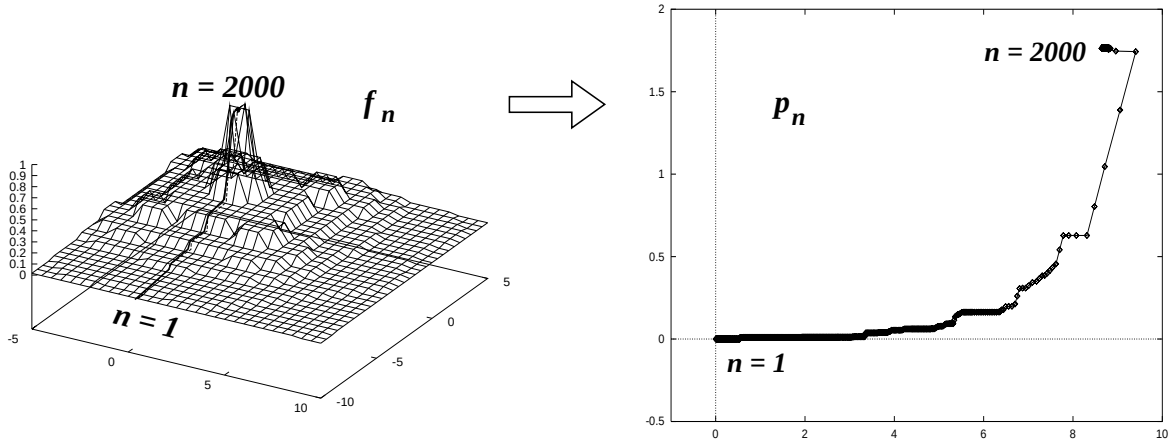


Abbildung 4.12: **Logarithmisch quantisierte Funktion F(x,y)** *mit $\alpha = 0.1$ und $\eta = 0.9$ ausgehend vom Startpunkt $(x = 0.5; y = -10)$. Beiden Funktionsdarstellungen liegt die Funktion $F(x, y)$ zugrunde. Der Gradientenaufstieg erfolgt auf Grundlage der Funktion $F(x, y)$, die resultierende Trajektorie ist jedoch mit der Funktion $G(x)$ gemäß (4.32) logarithmisch quantisiert.*

Kommunikation mit externen Speichern wurde durch ein besonderes Schema zur Speicherung vieler synaptischer Gewichte in externem RAM beim Konzept der NeNEB-Architektur [Larsson *et al.*, 1996, 1997a,b] Rechnung getragen, die im Rahmen einer Diplomarbeit [Krol, 1996] implementiert wurde.

Die Lerntrajektorien wurden als ein Hilfsmittel zur qualitativen Beurteilung von numerischen Störungen, die etwa Folge von numerischen Vereinfachungen sein könnten, eingeführt. Der Übergang von Fließkommadarstellung zu Ganzzahldarstellung stellt eine numerische Störung dar. Die weitere Vereinfachung der Multiplikation durch Multiplikation mit einer Zweierpotenz stellt eine stärkere Störung dar. Die Quantisierung von Fließkommazahlen durch Quantisierung des Logarithmus des Betrags dieser Zahlen auf ganze Zahlen stellt eine Störung dar, die mit den im vorangegangenen aufgeführten Störungen vergleichbar ist. Diese Störung wird im folgenden als *logarithmische Quantisierung* bezeichnet. Es handelt sich dabei um eine Art Fließkommadarstellung mit relativ großer Zahlendarstellungsdynamik. Zur Veranschaulichung der Visualisierung des Lernens mit einem durch numerische Modifikationen gestörten Backpropagation-Algorithmus durch Lerntrajektorien dienen einige Beispiele mit logarithmischer Quantisierung der synaptischen Gewichte des betreffenden Backpropagation-Netzwerks. Bei dieser Zahlendarstellung lassen sich leicht weitere Einschränkungen, wie die Beschränkung auf eine bestimmte Wortbreite durch Festlegen von Zahlenbereichsbegrenzungen formulieren, wodurch ein Übergang von einer Fließkommadarstellung zu einer Festkommadarstellung erreicht werden kann. In den folgenden Beispielen erfolgt die logarithmische Quantisierung gemäß

$$G(\omega) = \begin{cases} -2^{\lfloor \log_2(|\omega|) \rfloor} & \text{für} \quad \omega < 0 \\ 0 & \text{für} \quad \omega = 0 \\ 2^{\lfloor \log_2(|\omega|) \rfloor} & \text{für} \quad \omega > 0 \end{cases} \tag{4.32}$$

zur Störung der Funktion (4.24). Durch Wahl der Basis ist der Grad der Störung kontinuierlich variierbar. Im Kontext des Entwurfs digitaler Schaltungen wurde *zwei* als Basis gewählt. Das Ergebnis ist in der Abbildung 4.12 dargestellt. Dabei erfolgte der Gradientenaufstieg auf der Funktion (4.24). Die resultierende Trajektorie ist mit der Funktion $G(x)$ gemäß (4.32) logarithmisch quantisiert, wodurch die Trajektorie stufig wird. Die Stufen finden sich in der korrespondierenden Lerntrajektorie wieder. Dieses Beispiel demonstriert, wie die Existenz von Stufen in einer Fehlerfunktion, die durch numerische Vereinfachungen hervorgerufen werden können, in der korrespondierenden Lerntrajektorie zu sehen sind.

4.2.5 Backpropagation

Die im Rahmen der vorliegenden Arbeit eingeführten Lerntrajektorien sind zweidimensionale Abbildungen von Kurven aus höherdimensionalen Räumen. Im Kontext des Backpropagation-Lernens ergeben sich solche Kurven, genauer gesagt deren Stützpunkte, aus der schrittweisen Veränderung des Satzes der synaptischen Gewichte eines lernenden Backpropagation-Netzwerks. Um die Semantik der zweidimensionalen Lerntrajektorien-Darstellung zu veranschaulichen, wurden vom Autor zunächst solche Probleme analysiert, die eine räumliche Vorstellung zulassen. Dabei wurde auf Backpropagation-Netzwerke verzichtet, da selbst das kleinste,

nicht-triviale Backpropagation-Netzwerk – das XOR-Netzwerk (vgl. Anfang Kapitel 2, S. 7 ff.) – schon über neun synaptische Gewichte verfügt, die während des Lernens verändert werden, so daß das Training eines XOR-Netzwerks bereits einen Gradientenabstieg in einem neundimensionalen Raum darstellt und sich somit der direkten räumlichen Anschauung entzieht. Im folgenden Abschnitt wird nun die Visualisierung des Lernens durch Lerntrajektorien anhand von Backpropagation-Netzwerken von sehr unterschiedlicher Größe demonstriert. Dabei erfolgt das Lernen mit unterschiedlichen numerischen Vereinfachungen, die den Lernvorgang in unterschiedlicher Weise beeinflussen, was anhand der resultierenden Lerntrajektorien visualisiert wird.

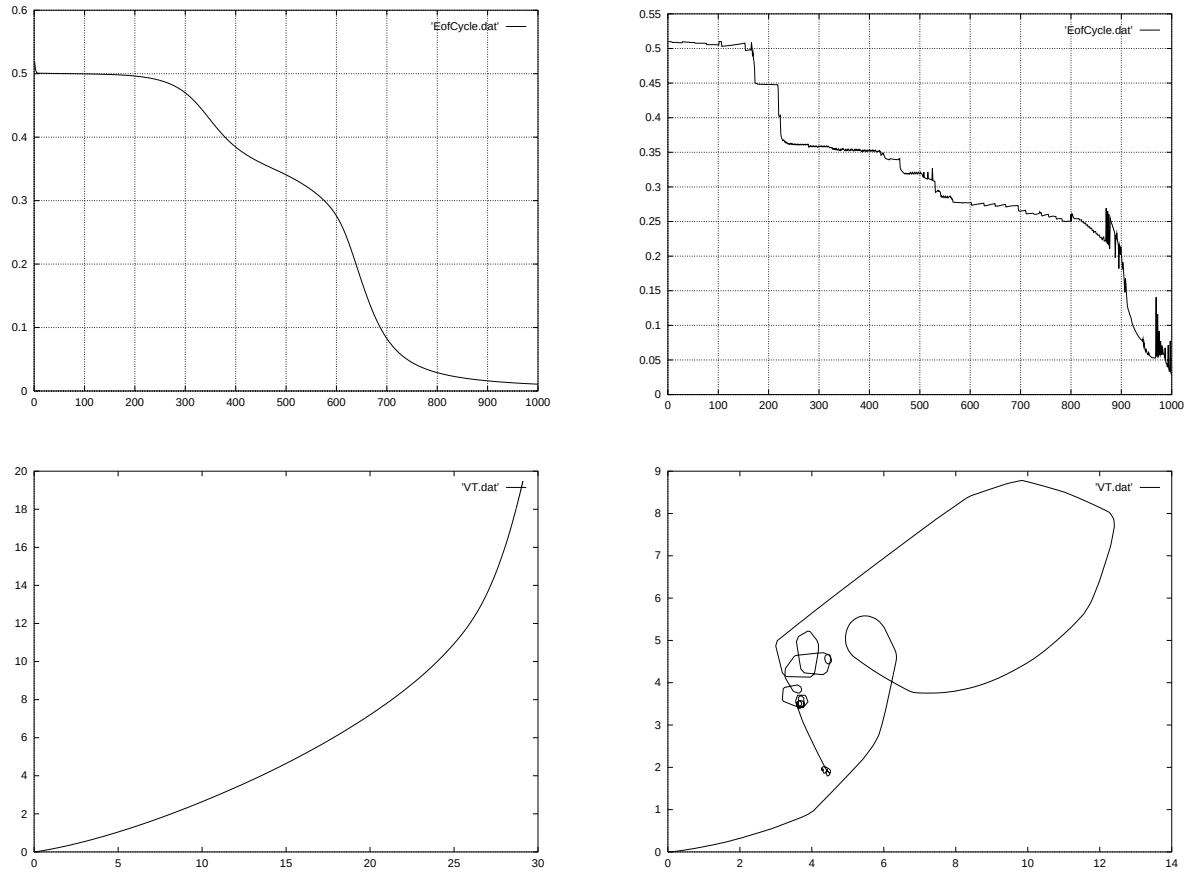


Abbildung 4.13: **2-2-1-XOR** *Vergleich des Konvergenzverhaltens durch Visualisierung mit Lerntrajektorien für ein 2-2-1-Backpropagation-Netzwerk beim Lernen der XOR-Funktion mit einer Lernrate von $\eta = 0,2$, einem Impulsterm $\alpha = 0,2$ und einem Temperaturparameter $\tau = 0,5$. Die synaptischen Gewichte wurden mit Zufallszahlen in dem Intervall $[-0,5 \ldots 0,5]$ initialisiert. Das Training erfolgte über 1000 Lernzyklen. Die Fließkommadarstellung und die Festkommadarstellung lieferten bei diesem Problem gleiche graphische Darstellungen. Daher sind nur die Ergebnisse der Fließkommadarstellung und der logarithmischen Quantisierung abgebildet.*

XOR

Als erstes Beispiel zur Demonstration der Lerntrajektorien dient das Erlernen der XOR-Funktion $y(x_1, x_2) = x_1 \oplus x_2 = \overline{x_1}x_2 \vee x_1\overline{x_2}$ mit einem [2-2-1]-Backpropagation-Netzwerk wie in [Rumelhart *et al.*, 1986a]. Als Lernmuster dienen dabei vier Vektorpaare $(\vec{x};\vec{y})_{n=1\ldots4} \in \{ ((0,0);(0))_1, ((0,1);(1))_2, ((1,0);(1))_3, ((1,1);(0))_4 \}$ als Lernmuster. Während des Lernens werden $(i+1)\cdot j + (j+1)\cdot k = (2+1)\cdot 2 + (2+1)\cdot 1 = 9$ synaptische Gewichte modifiziert, um die Fehlerfunktion E zu minimieren – also ein Gradientenabstieg in 9 Dimensionen. Gelernt wurde mit Fließkommadarstellung, Festkommadarstellung und mit logarithmisch quantisierten synaptischen Gewichten mit einer Lernrate von $\eta = 0,05$, einem Impulsterm $\alpha = 0,9$ und einem Temperaturparameter $\tau = 0,5$. Die synaptischen Gewichte wurden mit Zufallszahlen aus dem Intervall $[-0,5 \ldots 0,5]$ initialisiert, jedoch alle mit demselben Satz synaptischer Gewichte. Das Lernen beginnt also in allen drei Fällen in der Umgebung desselben Startpunkts. Die Umgebung ist dabei durch die numerischen Vereinfachungen festgelegt. Die Lerntrajektorien sind in Abbildung 4.13 nebeneinander dargestellt. Nach 1000 Lernzyklen konvergiert der Backpropagation-Algorithmus mit allen drei numerischen Vereinfachungen für die gewählten Lernparameter. Mit einer Lernrate von $\eta = 0,2$ und einem Impulsparameter $\alpha = 0,2$ konvergiert der Backpropagation-Algorithmus dagegen nicht bei logarithmisch quantisierten synaptischen Gewichten.

N-2-N-Encoder

Der N-M-N-Encoder ist eine klassische Aufgabe für Backpropagation-Netzwerke und kann zum Test von Variationen des Backpropagation-Algorithmus verwendet werden [Hertz *et al.*, 1991]. Der N-2-N-Encoder ist ein Spezialfall des N-M-N-Encoders, mit nur $M = 2$ Hidden-Neuronen. Lerntrajektorien wurden zur Beobachtung des Verhaltens des Backpropagation-Algorithmus eingeführt. Die Visualisierung des Lernens von Encoder-Aufgaben mit Hilfe der Lerntrajektorien liegt daher nahe.

Beim N-M-N-Encoder sind N orthogonale Eingangsvektoren auf N orthogonale Ausgangsvektoren abzubilden. Ein N-M-N-Encoder-Netzwerk mit $M \approx \log_2(N)$ kann durch binäre Kodierung in der verdeckten Schicht die gestellte Aufgabe auch für größere N lernen [Rumelhart und McClelland, 1986]. Dagegen wird das Lernen der Abbildung orthogonaler Eingangsvektoren auf orthogonale Ausgangsvektoren durch ein N-2-N-Backpropagation-Netzwerk mit nur zwei Hidden-Neuronen für $N > 8$ schwierig, obwohl Lösungen für endlich große N existieren. Wird jedoch, wie in [Bakker *et al.*, 1993] beschrieben, ein Codierungsschema nicht-orthogonaler Vektoren verwendet, so ist das Problem auch für große N lernbar.

1-aus-N-Codierung		N/2-aus-N-Codierung	
$\vec{x}$	$\vec{y}$	$\vec{x}$	$\vec{y}$
$(1,0,0,0,\ldots,0)$	$(1,0,0,0,\ldots,0)$	$(1,1,1,1,0,\ldots,0)$	$(1,1,1,1,0,\ldots,0)$
$(0,1,0,0,\ldots,0)$	$(0,1,0,0,\ldots,0)$	$(0,1,1,1,1,\ldots,0)$	$(0,1,1,1,1,\ldots,0)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(0,\ldots,0,0,1,0)$	$(0,\ldots,0,0,1,0)$	$(0,\ldots,1,1,1,1,0)$	$(0,\ldots,1,1,1,1,0)$
$(0,\ldots,0,0,0,1)$	$(0,\ldots,0,0,0,1)$	$(0,\ldots,0,1,1,1,1)$	$(0,\ldots,0,1,1,1,1)$

(4.33)

Beim Lernen des N-2-N-Encoder-Problems mit dem Backpropagation-Algorithmus hängt das Konvergenzverhalten fraglos von numerischen Vereinfachungen ab und ist schon für $N > 8$ auch bei Fließkommadarstellung für eine 1-aus-N-Codierung sehr stark gestört, wogegen das Kon-

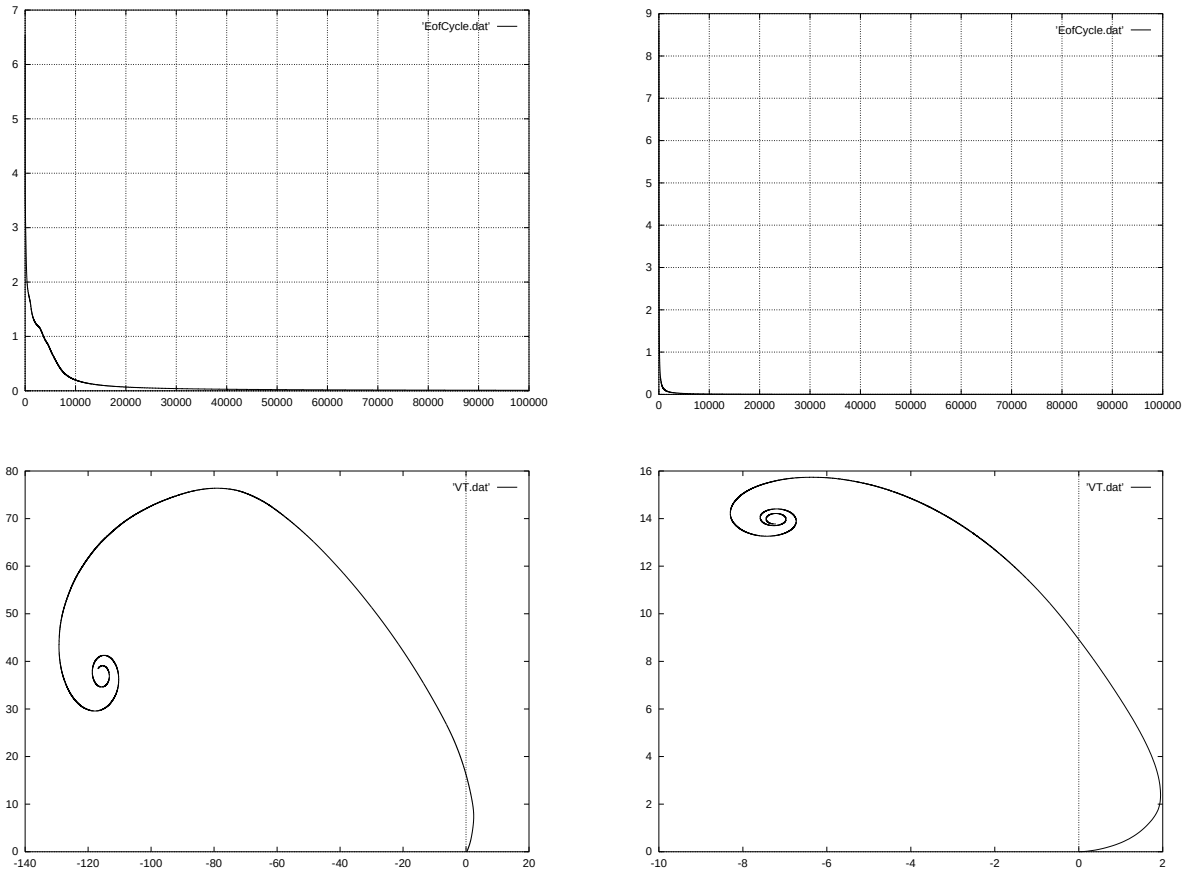


Abbildung 4.14: **8-2-8-Encoder** *Vergleich des Konvergenzverhaltens durch Visualisierung mit Lerntrajektorien für 1-aus-N-Codierung (links) und N/2-aus-N-Codierung (rechts) der Lernmuster. Es wurden* 100.000 *Lernzyklen mit einer Lernrate* $\eta = 0,1$*, Impulsparameter* $\alpha = 0,2$ *und* $\tau = 0,5$ *durchgeführt. Die synaptischen Gewichte wurden mit Zufallszahlen aus dem Intervall* $[-0,5 \ldots 0,5]$ *initialisiert. Das Training erfolgte mit Fließkommadarstellung.*

vergenzverhalten gutartig wird und die Lerngeschwindigkeit des Backpropagation-Algorithmus erhöht wird, wenn statt einer 1-aus-N-Codierung eine N/2-aus-N-Codierung für die Lernmuster gewählt wird. Damit kann sogar ein 50-2-50-Encoder und ein 100-2-100-Encoder gelernt werden, was mit einer 1-aus-N-Codierung der Lernmuster nicht mehr funktioniert [Bakker *et al.*, 1993]. Das N-2-N-Encoder-Problem ist daher gut geeignet, um das Konvergenzverhalten mit Hilfe von Lerntrajektorien zu visualisieren.

Vereinfachte Zeichenerkennung

Im vorliegenden Beispiel wird zur Erkennung von Großbuchstaben, dargestellt in einer 5×7-Punktmatrix, ein [35-10-26] Netzwerk mit $26 \cdot (10 + 1) + 10 \cdot (35 + 1) = 646$ synaptischen Gewichten verwendet (s. Abb. 4.16). Dieses Beispiel demonstriert die Visualisierung des Lernvorgangs eines Backpropagation-Networks mit einer deutlich größeren Zahl synaptischer Gewichte, als bei den vorangegangenen Beispielen. Es wurden 10.000 Lernzyklen mit einer Lern-

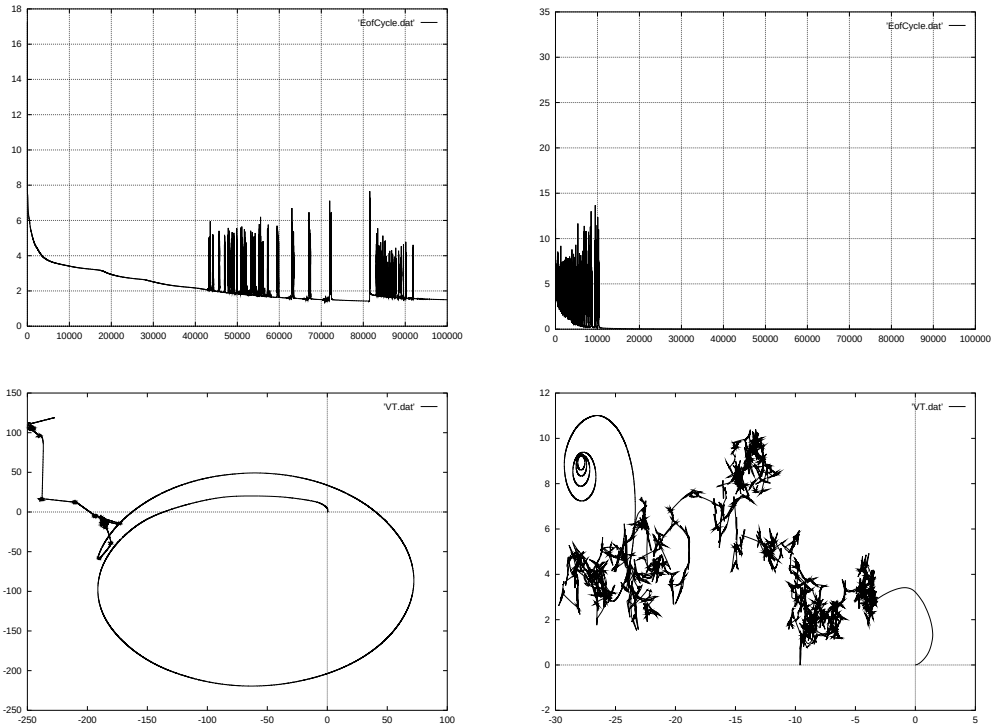


Abbildung 4.15: **16-2-16-Encoder** *Vergleich des Konvergenzverhaltens durch Visualisierung mit Lerntrajektorien für eine 1-aus-N-Codierung (links) und eine N/2-aus-N-Codierung (rechts) der Lernmuster. Es wurden* 100.000 *Lernzyklen mit einer Lernrate* $\eta = 0,1$*, Impulsparameter* $\alpha = 0,2$ *und* $\tau = 0,5$ *durchgeführt. Die synaptischen Gewichte wurden mit Zufallszahlen auf dem Intervall* $[-0,5 \ldots 0,5]$ *initialisiert. Das Training erfolgte mit Fließkommadarstellung.*

rate $\eta = 0,05$ und einem Impulsparameter $\alpha = 0,1$ und einem Temperaturparameter $\tau = 0,5$ durchgeführt. Initialisiert wurde das Backpropagation-Netzwerk mit zufälligen Werten aus dem Intervall $[-0,5 \ldots +0,5]$. In Abbildung (4.17) sind die Lerntrajektorien p_n und Fehler E für unterschiedliche numerische Vereinfachungen gegenübergestellt. Zum Vergleich wird Training mit Fließkommadarstellung, Festkommadarstellung und mit logarithmischer Quantisierung durchgeführt. Diese drei numerischen Vereinfachungen stören den Lernvorgang unterschiedlich stark.

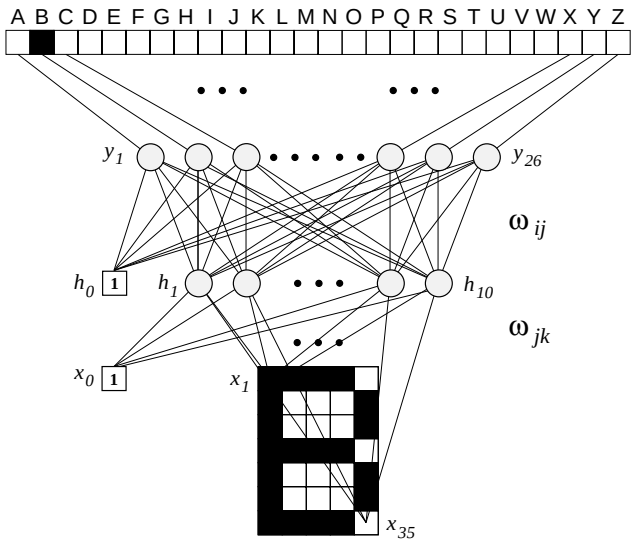


Abbildung 4.16: **Zeichenerkennung mit 35-10-26-Backpropagation-Netzwerk** *(Abb. aus [Larsson, 1997]). Die zu erkennenden Zeichen liegen als* 5×7*-Punktmatrix vor und werden auf einen 1-aus-n-Code abgebildet. Das Netzwerk hat* $26 \cdot (10+1) + 10 \cdot (35+1) = 646$ *synaptische Gewichte. Ähnliche Netzwerke findet man in der Literatur beispielsweise zur Erkennung von handschriftlichen Postleitzahlen [Hertz* et al.*, 1991].*

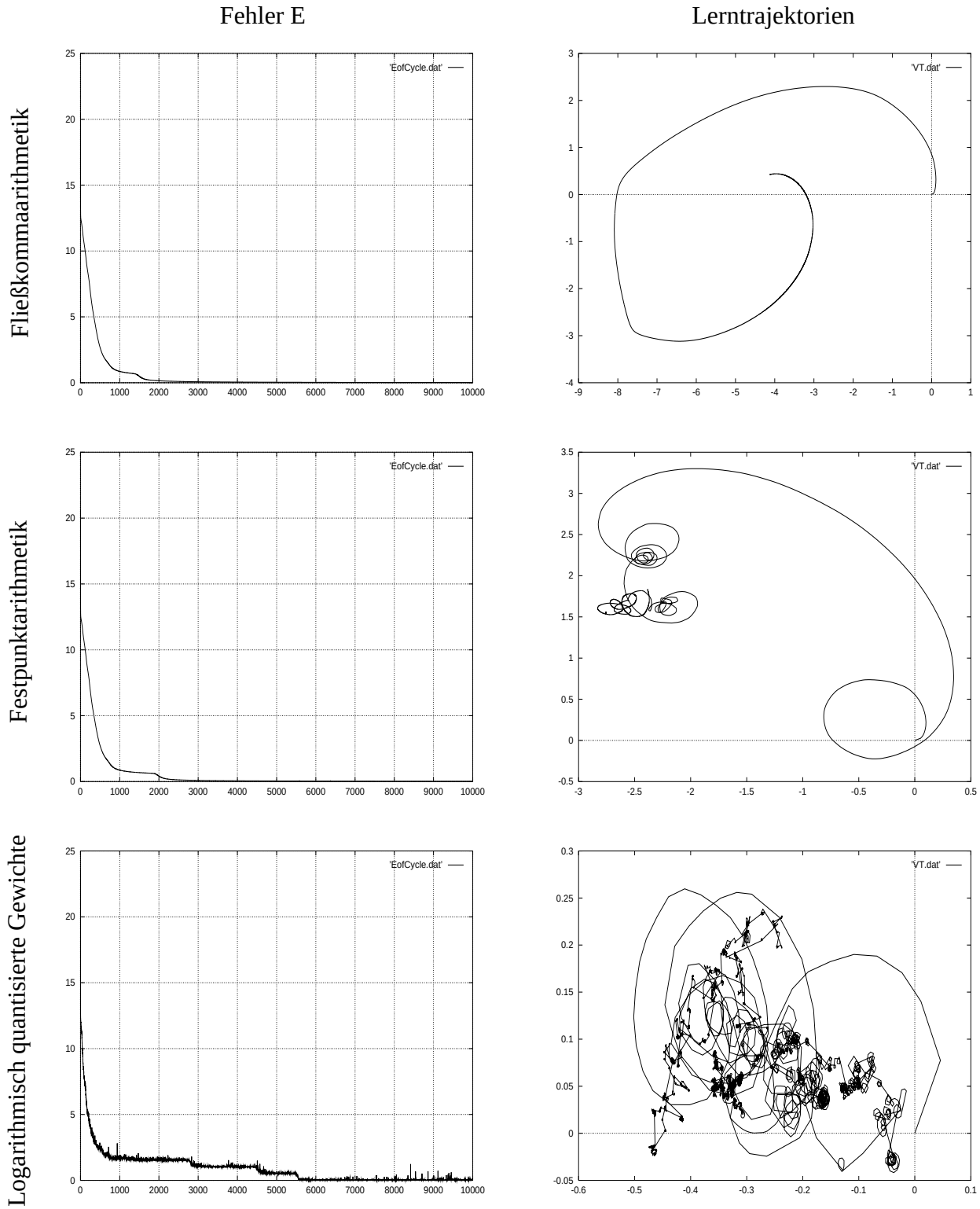


Abbildung 4.17: **Zeichenerkennung - Lerntrajektorien** *des [35-10-26]-Backpropagation-Netzwerks (Abb. 4.16) aus [Larsson, 1997].*

Der Lernvorgang wird vergleichend durch den totalen Fehler E sowie durch Lerntrajektorien visualisiert. Bei diesem Beispiel wird deutlich, daß die Visualisierung des Lernvorgangs mit Hilfe der Lerntrajektorien eine direktere Anschauung bezüglich der Störungen des Lernvorgangs zu vermitteln vermag, als die Protokollierung des totalen Fehlers E als Funktion des Lernzyklus.

Lernen mit *ganzzahlig quantisierten Gewichten I(x)* in Festkommadarstellung wird durch Quantisierung gemäß (4.31) erreicht. Die Quantisierung gemäß (4.31) stört den Lernvorgang in geringerem Maße als die logarithmische Quantisierung der synaptischen Gewichte ω_{ij} und ω_{jk} gemäß (4.32). Die Störung des Lernvorgangs tritt beim Vergleich des Gesamtfehlers E in Abbildung 4.17 nicht so deutlich in Erscheinung wie beim Vergleich der korrespondierenden Lerntrajektorien. Lernen mit logarithmisch quantisierten Gewichten (4.32) stellt eine für digitale Schaltungsimplementierung denkbare Alternative zur Multiplikation dar, bewirkt aber sehr stark gestörtes Lernverhalten, wenn auch der Lernvorgang konvergiert. Der Unterschied beider numerischen Vereinfachungen bezüglich des Lernverhaltens wird anhand der eingeführten Lerntrajektorien, verglichen mit der graphischen Darstellung des über dem Lernschritt protokollierten Fehlers E, in stärkerem Maße deutlich.

Sigmoidfunktionsnäherungen

Die Sigmoidfunktion (2.3) und die erste Ableitung der Sigmoidfunktion (2.9) erfordern für eine effiziente Hardware-Implementierung starke numerische Vereinfachungen gegenüber Software-Implementierungen. Die Berechnung der Sigmoidfunktion

$$\texttt{g(x)} = \texttt{1/(1 + exp(-x/t);} \tag{4.34}$$

erfordert, wenn sie in dieser direkten Weise formuliert wird, die Änderung des Vorzeichens von `x` nach `-x`, eine Division `/t`, die Berechnung der Exponentialfunktion `exp()`, eine Addition `1 +` und eine Reziprokwertberechnung `1/(...)`. Die Änderung des Vorzeichens ist bei Darstellung von `x` im Zweierkomplement einfach zu implementieren. Alle anderen Operationen lassen sich jedoch bei direkter Implementierung der mathematischen Operationen in Hardware nur mit

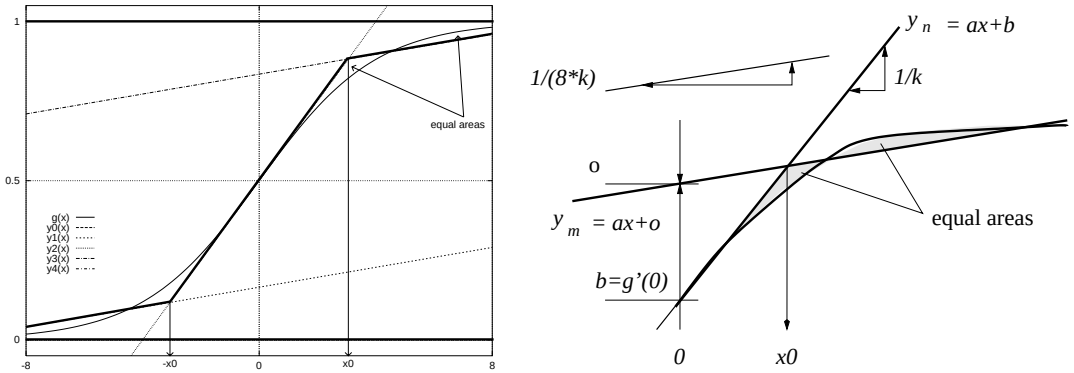


Abbildung 4.18: **Stückweise lineare Interpolation der Sigmoidfunktion.** *Aus [Larsson* et al.*, 1997a].*

großem Aufwand realisieren. Die Implementierung der Exponentialfunktion als Taylor-Reihe $\exp(x) = \sum_{n=0}^{N} x^n/n!$ ist für eine direkte Hardware-Implementierung zu aufwendig. Die Berechnung der Sigmoidfunktion $g(x)$ über eine Taylor-Reihe wurde in [Krol, 1996] untersucht. Diese Variante erfordert jedoch viele Operationen und darüber hinaus liefert die Taylor-Reihe [Press *et al.*, 1986] der gebrochen rationalen Sigmoidfunktion nur in der Nähe der Entwicklungsstelle eine hinreichend gute Näherung. Mit rationalen Funktionen [Press *et al.*, 1986] lassen sich möglicherweise bessere Ergebnisse erzielen, die jedoch – aufgrund vieler notwendiger Rechenoperationen – einen hohen Aufwand für eine Hardware-Implementierung zur Folge hätten. Diese Alternative wurde daher nicht im Detail untersucht. In Anlehnung an [Cloutier und Simrad, 1994] wurde zur Hardware-Implementierung der Vorwärtsphase des Backpropagation-Algorithmus für die Sigmoidfunktion in [Larsson *et al.*, 1996, 1997a,b] eine Näherung durch Geradenstücke in der Form

$$y(\mathsf{x}) = \begin{cases} 0 & \text{wenn} \quad \frac{\mathsf{x}}{(8*\mathsf{k})} + o_- < 0 \\ \frac{\mathsf{x}}{(8*\mathsf{k})} + o_- & \text{wenn} \quad \mathsf{x} \leq -\mathsf{x0} \\ \frac{\mathsf{x}}{\mathsf{k}} & \text{wenn} \quad -\mathsf{x0} < \mathsf{x} < +\mathsf{x0} \\ \frac{\mathsf{x}}{(8*\mathsf{k})} + o_+ & \text{wenn} \quad \mathsf{x} \geq +\mathsf{x0} \\ 1 & \text{wenn} \quad \frac{\mathsf{x}}{(8*\mathsf{k})} + o_+ > 1 \end{cases} \tag{4.35}$$

gewählt, wie in Abbildung 4.18 dargestellt. Anstelle des kontinuierlichen Temperaturparameters τ ist dabei ein inverser Temperaturparameter $c = 1/\tau$ mit $c \in \{0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8\}$ wählbar. Dieser wird durch eine natürliche Zahl, den Steilheitsparameter $\mathsf{s} \in \{0, 1, 2, 3, 4, 5, 6, 7\}$, zusammen mit additiven Konstanten $\mathsf{x0}$ bestimmt. Damit kann die Sigmoidfunktion in einem weiten Bereich interpoliert werden. Die stückweise lineare Näherung der Sigmoidfunktion (4.35) wird mit dem reziproken Steigungsparameter $\mathsf{k} = 2^{\mathsf{s}-1}$ eingestellt. Der Parameter s steuert einen Barrel-Shifter.

Um eine gegebene Sigmoidfunktion durch die stückweise lineare Funktion (4.35) zu nähern, sind die Interpolationsparameter zu bestimmen. Dies geschieht analytisch in zwei Schritten. Zuerst wird der reziproke Steigungsparameter so gewählt, daß die reziproke Steigung der stückweise linearen Näherung der Sigmoidfunktion (4.35) so gut wie möglich an $1/g'(x)$ angenähert ist. Im zweiten Schritt sind die additiven Konstanten $\mathsf{x0}$ (und $-\mathsf{x0}$) zu bestimmen. Dies kann analytisch durch Minimierung der Flächendifferenz

$$\begin{aligned} F(x_0) &= |G(x_m) - Y(x_0, x_m)| \\ &= |\textstyle\int_0^{x_m} g(x)dx - \int_0^{x_m} y(x_0, x)dx| \\ &\approx |(x_m + \tau \cdot \log_2(\tfrac{1}{2})) - (\textstyle\int_0^{x_0} y_1(x)dx + \int_{x_0}^{x_1} y_2(x)dx + \int_{x_1}^{x_m} y_3(x)dx)| \end{aligned} \tag{4.36}$$

zwischen der Sigmoidfunktion $g(x)$ und der stückweise linear angenäherten Funktion $y(x)$ im Intervall $[0, \ldots, x_{max}]$ erfolgen. Aufgrund der Symmetrie der Sigmoidfunktion und ihrer Ableitung bei $x = 0$ genügt es, nur das Intervall $[0, \ldots, x_{max}]$ anstelle von $[x_{min}, \ldots, x_{max}]$ zu

betrachten. Die Integrationsgrenze $x_m < \infty$ dient der Integralkonvergenz. Als Ergebnis ergibt sich

$$\begin{array}{rcl} \mathsf{k} & = & \frac{1}{g'(0)} \quad \text{mit } \mathsf{k} \in \{\frac{1}{2}, 1, 2, 4, 8, 16, 32, 64\} \\ \mathsf{x0} & \approx & 0,3825 \cdot \mathsf{k} \\ a & = & \frac{7}{8 \cdot \mathsf{k}} \cdot \mathsf{x0} \qquad \text{und} \qquad b = g(0) \\ o_- & = & b - a \qquad\quad \text{und} \quad o_+ = b + a. \end{array} \tag{4.37}$$

Diese Näherung für die Sigmoidfunktion wurde im Rahmen einer Diplomarbeit [Krol, 1996] als synthetisierbares VHDL-Modell implementiert. Es gilt

$$g'(x) = \frac{1}{\tau} \cdot g(x) \cdot g(-x). \tag{4.38}$$

Die Herleitung befindet sich im Anhang auf S. 163. Wird nun zum Lernen für $g(x)$ die Näherung (4.35) verwendet und basierend auf dieser zusammen mit dem Ausdruck (4.38) die Ableitung $g'(x)$ berechnet, so stellt das zwar eine Störung des Lernverhaltens dar, der Backpropagation-Algorithmus findet jedoch auch damit – die Wahl geeigneter Lernparameter vorausgesetzt – richtige Lösungen. In [Cloutier und Simrad, 1994] zeigen die Autoren die Eignung verschiedener, zur Hardware-Implementierung gut geeigneter, numerischer Vereinfachungen durch Lernen verschiedener Probleme anhand des erreichten, totalen Fehlers E. Die Abbildungen 4.19 und 4.20 zeigen die Visualisierung des Lernens mit Lerntrajektorien bei der einfachen Buchstabenerkennung für Lernen mit reeller (Fließkommadarstellung) Sigmoidfunktion $g(x)$ und deren Ableitung $g'(x)$, Lernen mit der stückweise linearen Näherung (4.35) und Lernen mit ganzzahlig quantisierter Sigmoidfunktion gemäß

$$G(x) = I(g(I(x)) \qquad \text{und} \qquad G'(x) = I(\frac{1}{\tau} g(x) \cdot g(-x)). \tag{4.39}$$

Die ganzzahlige Quantisierung (4.39) mit $I(x)$ gemäß (4.31) entspricht der Näherung der Sigmoidfunktion und ihrer Ableitung durch eine *Look Up Table* (LUT), die für eine effiziente Hardware-Implementierung in Form eines Schaltnetzes geeignet ist (vgl. Abschnitt 5.4, S. 151).

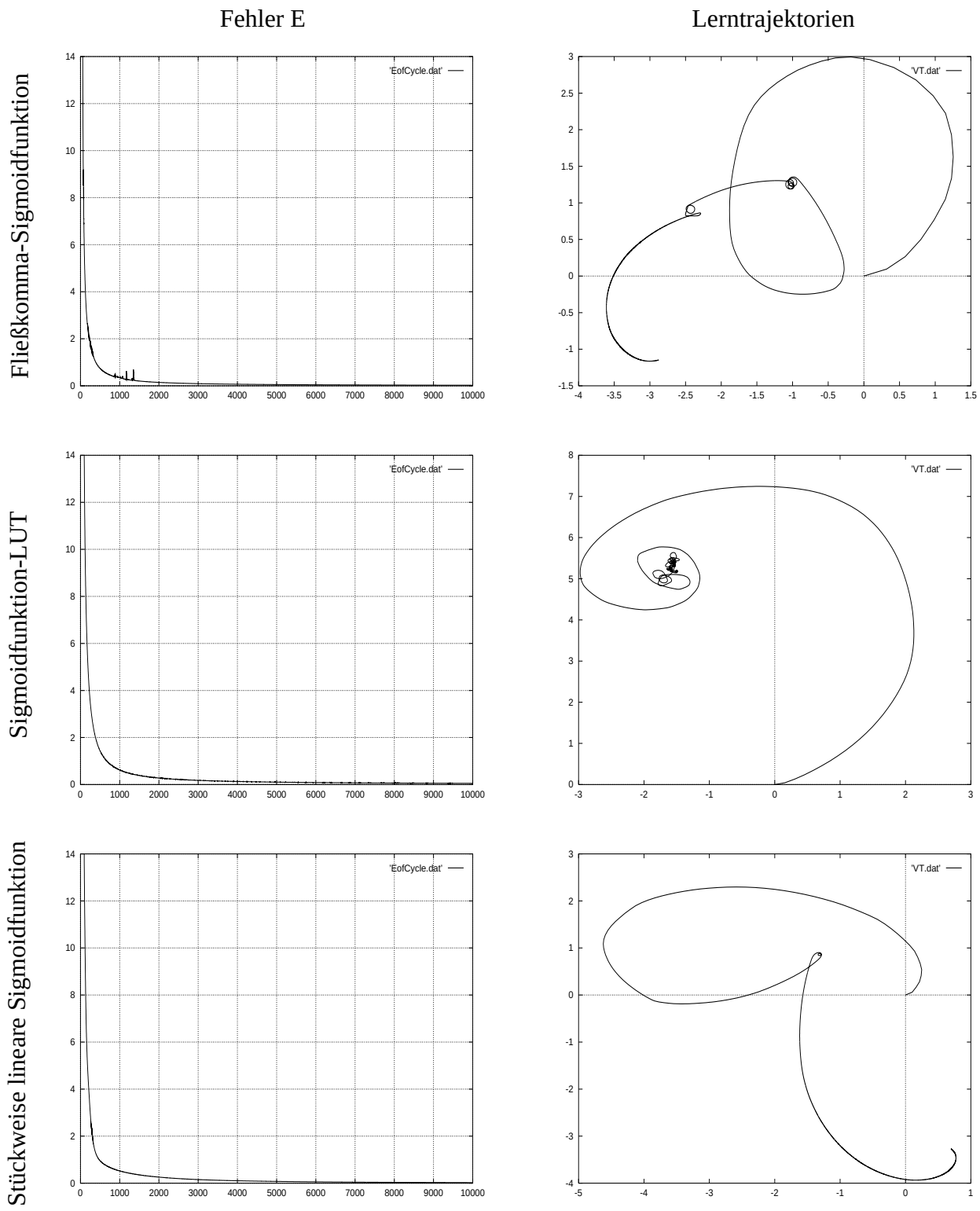


Abbildung 4.19: **Zeichenerkennung mit orthogonalen Ausgangsvektoren - Lerntrajektorien** *des [35-10-26]-Backpropagation-Netzwerks mit verschiedenen numerischen Vereinfachungen der Sigmoidfunktion (Fließkommadarstellung, 8-Bit-Sigmoidfunktion-LUT, stückweise lineare Geradennäherung der Sigmoidfunktion). 10.000 Lernzyklen, mit* $\eta = 0,2$ *und* $\alpha = 0,2$ *und* $\tau = 0,5$.

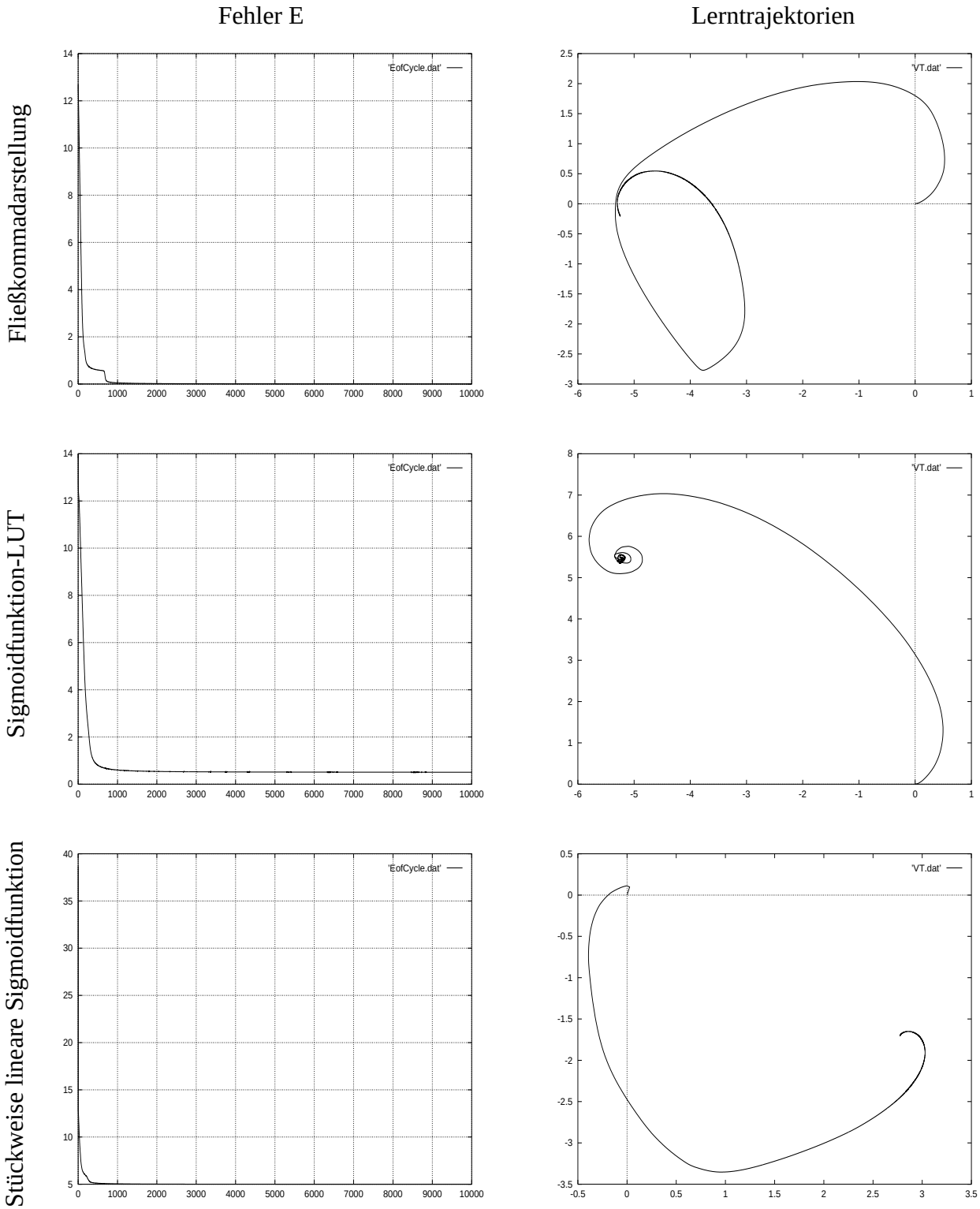


Abbildung 4.20: **Zeichenerkennung mit N/2-Codierung der Ausgangsvektoren - Lerntrajektorien** *des [35-10-26]-Backpropagation-Netzwerks mit verschiedenen numerischen Vereinfachungen der Sigmoidfunktion (Fließkommadarstellung, 8-Bit-Sigmoidfunktion-LUT, stückweise lineare Geradennäherung der Sigmoidfunktion). 10.000 Lernzyklen, mit* $\eta = 0,1$ *und* $\alpha = 0,2$ *und* $\tau = 0.5$.

Bildklassifizierung

Abschließend wird ein sehr großes Backpropagation-Netzwerk trainiert, verschiedene Graustufenbilder, mit 2^8 Graustufen, zu klassifizieren. Es handelt sich um ein [6300-32-10]-Netzwerk mit ($(6300 \cdot (32+1) + 32 \cdot (10+1) = 201962$ synaptischen Gewichten), das zehn verschiedene Graustufenbilder auf zehn nicht-orthogonale Ausgangsvektoren (vgl. N-2-N-Encoder) abbilden soll. Zwar existieren Abschätzungen bezüglich der Zahl der benötigten synaptischen Gewichte [Hertz *et al.*, 1991] in Abhängigkeit von der Zahl der Lernmustern, die bei diesem Beispiel jedoch nicht zu Rate gezogen wurden. Vielmehr dient dieses abschließende Beispiel der Demonstration der Eignung der Lerntrajektorien für ein sehr großes Backpropagation-Netz. Die Graustufenbilder liegen als Pixelgraphik mit $k = 90 \times 70$ Bildpunkten vor. Kleinere Bilder werden zentriert mit einem schwarzen Rahmen hinterlegt. Schwarz ist einer Aktivierung 0 zugeordnet. Die Bilder bilden die Eingangsvektoren mit $k = 90 \cdot 70 = 6300$ Bildpunkten als Vektorkomponenten x_k, wie in Abb. 4.21 skizziert wurde. Das Training erfolgt über 1.000 Lernzyklen mit einer Lernrate $\eta = 0,1$, einem Impulsparameter $\alpha = 0,5$ und einem Temperaturparameter $\tau = 0,5$. Dabei wurde mit Fließkommadarstellung und mit logarithmisch quantisierten synaptischen Gewichten gemäß (4.32) gelernt. In Abbildung 4.22 sind der totale Fehler und die korrespondierenden Lerntrajektorien dargestellt.

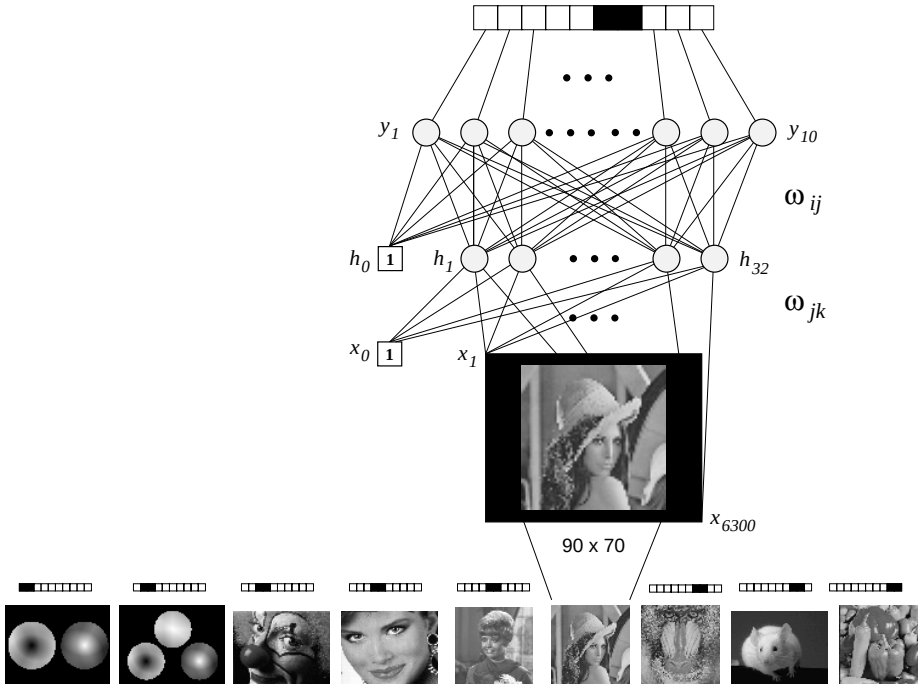


Abbildung 4.21: **Klassifizierung von Graustufenbildern** *mit 8-Bit-Graustufen und* 90×70 *Bildpunkten mit [6300-32-10]-Backpropagation-Netzwerk.*

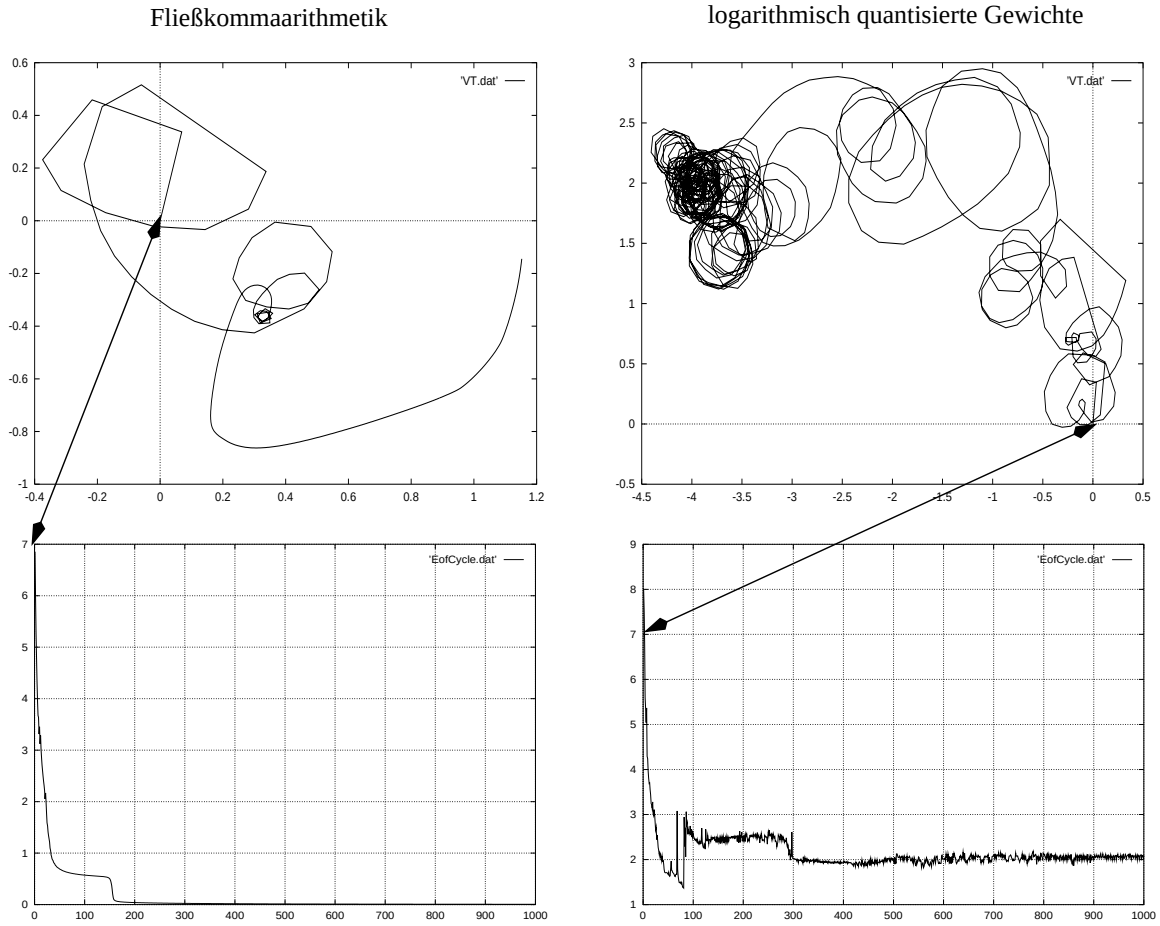


Abbildung 4.22: **Klassifizierung von Graustufenbildern - Lerntrajektorien** *und korrespondierende Fehler E. Gelernt wurde über 1000 Lernzyklen mit einer Lernrate von $\eta = 0,1$, einem Impulsparameter $\alpha = 0,5$ und einem Temperaturparameter $\tau = 0,5$. Die numerische Näherung mit logarithmischer Quantisierung stört den Lernvorgang zu stark.*

4.2.6 Schlußfolgerungen

Lerntrajektorien visualisieren den zeitlichen Verlauf von Parameteränderungen in hochdimensionalen Räumen, wie sie in Heuristiken – wie dem Backpropagation-Algorithmus – erfolgen, durch eine zweidimensionale Darstellung. Der Verlauf solcher Parameteränderungen hängt von der vorliegenden Optimierungsaufgabe ab. Außerdem hängt der Verlauf – beim Gradientenverfahren und damit auch beim Backpropagation-Lernen – vom gewählten Startpunkt ab (s. Beispiel Abb. 4.9). Ein Extrembeispiel dafür ist etwa die Wahl eines Startpunktes in einem lokalen Maximum.

Das konkrete Aussehen einer Lerntrajektorie hängt also von der Form der Fehlerfunktion und vom – beim Backpropagation-Algorithmus zur Initialisierung der synaptischen Gewichte – zufällig gewählten Startpunkt ab. Lerntrajektorien besitzen daher Aussagekraft im Vergleich untereinander, um verschiedene numerische Vereinfachungen und Implementierungen auf ver-

schiedenen Plattformen *untereinander* zu vergleichen. Dies gilt auch für die Wahl des Startpunkts, wie mit der aus Termen (4.18) zusammengesetzten Funktion (4.24) gezeigt. Der Startpunkt ist von dem verwendeten Zufallszahlengenerator abhängig. Daher ist es unter Umständen für den Vergleich von unterschiedlichen Implementierungen erforderlich, die zu vergleichenden Implementierungen mit demselben zufälligen Startpunkt zu initialisieren. Die Methode der Lerntrajektorien ist ein auf numerische Störungen, wie sie durch numerische Vereinfachungen hervorgerufen werden können, sensibles Visualisierungsverfahren.

So ist die Darstellung des Gesamtfehlers beim Backpropagation-Lernen unzureichend, um zwei Implementierungen bezüglich ihrer funktionalen Äquivalenz auf der Systemebene beurteilen zu können. Auch die Bewertung eines Endergebnisses ist in diesem Sinne unzureichend. Lerntrajektorien sind ein weitergehendes Hilfsmittel, um zu entscheiden, ob eine Implementierung eines Backpropagation-Netzwerks für die Lösung bestimmter Aufgaben geeignet ist. Lerntrajektorien sind geeignet, Implementierungen des Backpropagation-Algorithmus auf verschiedenen Plattformen qualitativ zu vergleichen, indem sie Verhalten der betreffenden Implementierungen auf einer hohen Abstraktionsebene – der Systemebene – visualisieren. Das mag vielleicht aus mathematischer Sicht unvollkommen erscheinen, ist aber insbesondere im Kontext des Entwurfs digitaler Systeme ein gangbarer Weg, komplexe Systeme in Bezug auf ihre Robustheit gegenüber unterschiedlichen Eingaben zu validieren.

Implementierungen von Algorithmen, die mathematisch auf Basis der reellen Zahlen formuliert sind, werden auf verschiedenen Plattformen immer zu leicht unterschiedlichem Verhalten neigen. Das gilt insbesondere dann, wenn ein Algorithmus in Software und in Hardware implementiert wird. Bei Software-Implementierungen wird in der Regel auf Funktionsbibliotheken zurückgegriffen. Für effiziente Hardware-Implementierungen von Algorithmen sind meist numerische Vereinfachungen bezüglich der Zahlendarstellung von Operanden notwendig. Das Ziel der Implementierung eines Algorithmus ist jedoch immer, Strukturen zu erstellen, die ein gewünschtes Verhalten zeigen. Die Implementierung des Backpropagation-Algorithmus soll bestimmte Aufgaben anhand von Beispielen erlernen können und dabei *robust* gegenüber Abweichungen von den Lernmustern sein.

Die Beobachtung des Verhaltens und der Wechselwirkung neu entwickelter Komponenten auf der Systemebene stellt ein wichtiges Hilfsmittel zur Validierung und zur Untersuchung der Robustheit neu entwickelter Komponenten dar, insbesondere bei der Realisierung von System-on-a-Chip-Entwürfen [Keating und Bricaud, 1998]. Der Nachweis funktionaler *Äquivalenz* unterschiedlicher Implementierungen ist solange von untergeordneter Bedeutung, solange die Spezifikation nicht den Anforderungen in Bezug auf Robustheit in einer realen Anwendungsumgebung genügt. Stören geringfügige Störungen einer entwickelten Komponente, wie das Kippen einzelner Bits, das Verhalten eines Gesamtsystems nicht, so können unterschiedliche Implementierungen einer solchen Komponente, die sich in ihrem Verhalten nur geringfügig unterscheiden, unter Umständen als funktional gleichwertig angesehen werden, obwohl sie nicht funktional äquivalent sind. Die funktionale Äquivalenz unterschiedlicher Repräsentationen (z.B. Verhaltensbeschreibung, Gatternetzliste und Chip-Layout) wird dann wichtig, wenn die entwickelte Komponente gestellten Anforderungen in Bezug auf Robustheit in einer (realen) Systemumgebung genügt und etwa Änderungen an den unterschiedlichen Repräsentationen vorgenommen werden müssen [Keating und Bricaud, 1998].

Die vorliegende Arbeit stellt Methoden als Hilfsmittel zur Überprüfung, Verfeinerung und Validierung von Spezifikationen für eine bestimmte Klasse von Systemen zur Verfügung. Neuronale Netzwerke und Einheiten zur digitalen Verarbeitung analoger Videosignale sind Beispiele für solche Systeme.

4.2.7 Ausblick

Für Lerntrajektorien sind, über die Visualisierung des Backpropagation-Lernens hinaus gehende, weitere Anwendungen denkbar. So könnten Lerntrajektorien unter Umständen in einem allgemeineren Sinne zur Verhaltensbeobachtung von anderen Vielparametersystemen und anderen Algorithmen, die einen Satz von vielen Parametern quasi gleichzeitig verändern, eingesetzt werden. Wird ein Bild als ein Punkt in einem hochdimensionalen Raum verstanden, so stellt sich eine Sequenz von aufeinander folgenden Bildern als eine Folge von Punkten in einem hochdimensionalen Raum dar, die mit Hilfe der Lerntrajektorien visualisiert werden, und könnte für bestimmte Klassen von Bildsequenzen charakteristische Lerntrajektorien liefern. Eine mit der Methode der Lerntrajektorien derart stark abstrahierend vorverarbeitete Bildsequenz könnte dann einem neuronalen Netzwerk als Eingabe dienen und so zur Identifikation von Bildsequenzen eingesetzt werden. So lassen sich aber unter Umständen auch bestimmte Bildtransformationen – wie Translation, Rotation, Skalierung von Gesamtbildern oder von Bildobjekten – und Bewegungsabläufe in Bildsequenzen mit Hilfe der Methode der Lerntrajektorien identifizieren. So könnten Lerntrajektorien als stark abstrahierende Vorverarbeitung von Bildsequenzen etwa zur Erkennung von Gesten, wie bestimmten Handbewegungen oder Kopfbewegungen, einsetzbar sein. Auch die Erkennung von Objekten, die sich in bestimmter Weise durch ein Bild bewegen, wie etwa Personen, könnte mit Hilfe der Lerntrajektorien unter Umständen vorverarbeitet werden. Wenn sich für bestimmte Bildtransformationen charakteristische Lerntrajektorien ergeben, so könnten diese möglicherweise für Bewegungsvorhersagen, wie sie bei der Kompression von Bildsequenzen für digitale Videoübertragung [Jack, 1996; Reimers, 1997] verwendet werden, von Nutzen sein.

4.3 VidTrans

Ausgehend von dem Systemkonzept einer Architektur zur Echtzeitverarbeitung von Videobildern mit neuronalen Netzwerken (NeNEB, s. Abb. 3.4, S. 73) wurde die rein digitale Verarbeitung analoger Videosignale in dieses Systemkonzept eingeschlossen. Die rein digitale Videosignalverarbeitung hat gegenüber analoger Verarbeitung mehrere Vorteile. Die digitale Signalverarbeitung hat grundsätzlich den Vorteil gegenüber der analogen Signalverarbeitung, daß die Verarbeitung weitgehend unabhängig von elektrischen Parametern der zur Signalverarbeitung verwendeten Komponenten ist. Darüber hinaus ist die digitale Signalverarbeitung auch unempfindlich gegenüber Änderungen elektrischer Parameter, wie etwa durch Alterung. Außerdem ermöglicht die rein digitale Verarbeitung analoger Videosignale SoC-Entwürfe mit rein digitalen Technologien, also auch mit FPGAs.

Die Entwicklung digitaler Komponenten zur Verarbeitung von analogen Videosignalen stellt jedoch eine Herausforderung in Bezug auf die Validierung solcher Komponenten dar. Wechsel-

wirkungen mit einer realen Anwendungsumgebung sind in besonderer Weise zu berücksichtigen. Die Validierung von Spezifikationen videosignalverarbeitender Komponenten erfordert besondere Methoden. Die im Rahmen der vorliegenden Arbeit vorgeschlagene Lösung zur Validierung videosignalverarbeitender Komponenten ist eine bidirektionale Schnittstelle – ein auf EPLDs basierender Transienten-Rekorder, namens *VidTrans* – zum Austausch von Videosignalen, d.h. kurzen Videosequenzen, zwischen Simulationen videosignalverarbeitender Komponenten und realen, analogen Videogeräten [Larsson, 1996]. Die Schaltung wurde auf verschiedenen internationalen Fachmessen (*Hannover-Messe Industrie'97* und *Elektronica'98* in München) ausgestellt sowie bei verschiedenen regionalen Veranstaltungen (*Handelskammer Hamburg* im Juni'97 und bei der Auftaktveranstaltung der *Hamburger Mikroelektronik Initiative* der TU-TECH GmbH (TU-Hamburg-Harburg) im September'97) praktisch demonstriert.

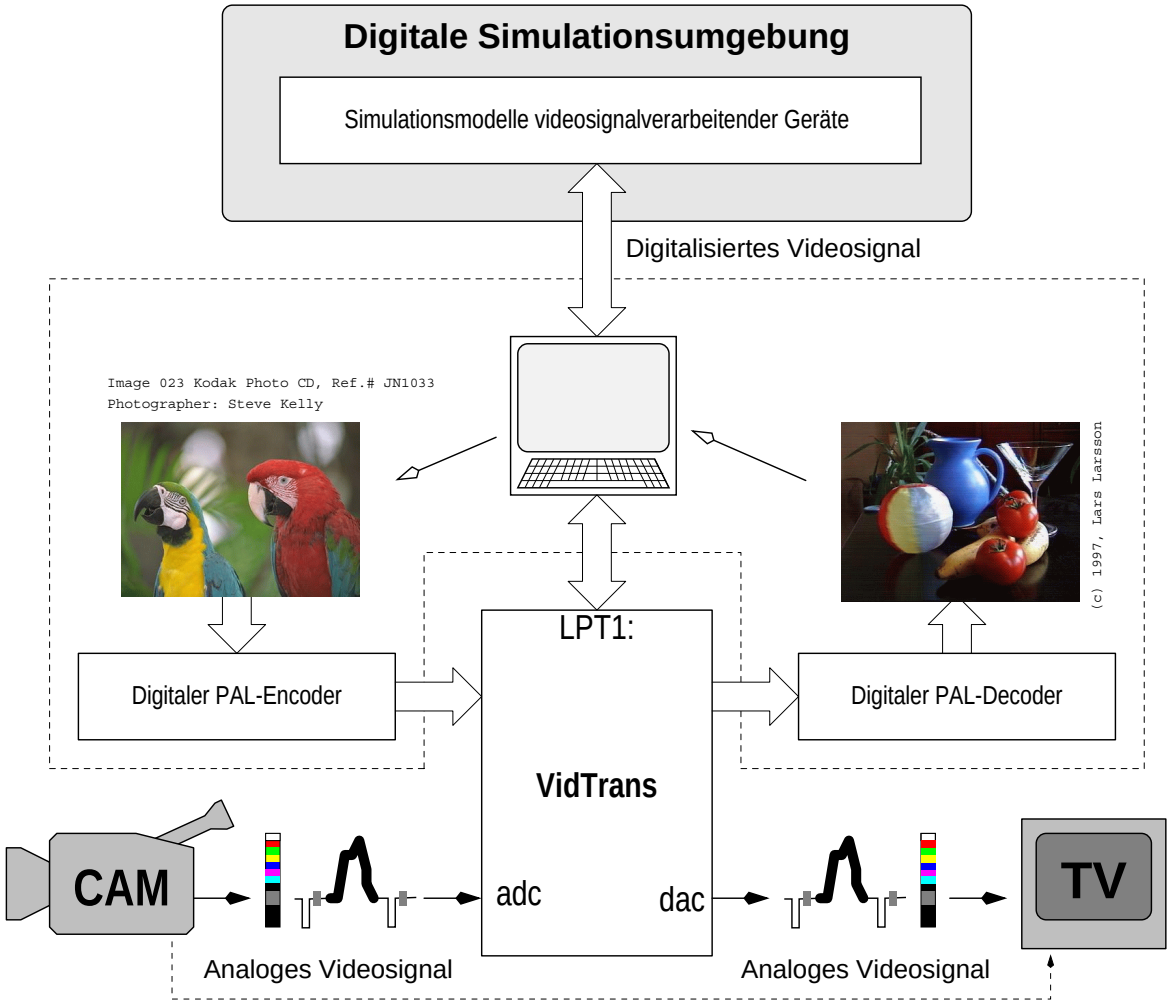


Abbildung 4.23: **Bidirektionale Simulationsschnittstelle - VidTrans.** *Der Austausch von realen (analogen) Videosignalen mit Simulationsmodellen (digitaler) videosignalverarbeitender Komponenten ermöglicht besonders systemnahe Simulationen, wodurch die Robustheit entwickelter Komponenten schon in frühen Entwurfsphasen untersucht werden kann.*

Durch den Austausch von Videosignalen zwischen Simulation und realen Geräten kann schon in frühen Entwurfsphasen eine relativ große Nähe zwischen entwickelten Komponenten und der realen Anwendungsumgebungen erreicht werden. Zur Realisierung von *VidTrans* wurden EPLDs eingesetzt. Durch die Verwendung von EPLDs kann sofern erwünscht die Ansteuerung verwendeter Analog-Digital-Konverter (ADC) und Digital-Analog-Konverter (DAC) bei systemnahen Simulationen mit berücksichtigt werden. Ebenso können mögliche Verfälschungen der analogen Signale, die unter Umständen durch die verwendeten analogen Komponenten im Videosignalpfad hervorgerufen werden können, bei den Simulationen und damit auch beim Entwurf der Komponenten zur rein digitalen Verarbeitung analoger Videosignale berücksichtigt werden, da die analogen Videosignale die gleichen Komponenten und analogen Signalpfade durchlaufen wie später im Zielsystem. Solch eine Nähe zum Zielsystem wird durch die Verwendung von EPLDs ermöglicht, da erst damit der Aufwand für die Realisierung einer anwendungsspezifischen Analogschnittstelle in einem praktikablen Rahmen gehalten werden kann. *VidTrans* steht zwischen reiner Simulation und In-Circuit-Prototyping und ermöglicht so Quasi-In-Circuit-Prototyping zu einem Zeitpunkt, als FPGAs für direktes In-Circuit-Prototyping mit hinreichend vielen Gatter-Äquivalenten bei hinreichend geringen Gatterlaufzeiten noch nicht zur Verfügung standen, um eine Aufgabe, wie die Realisierung eines digitalen PAL-Farbvideosignal-Encoders [Larsson, 1999b], mit geringem Aufwand, d.h. mit nur einem FPGA, zu lösen.

Mit Hilfe von *VidTrans* wurden ausführbare Spezifikationen von digitalen PAL-Encodern und digitalen PAL-Encodern realisiert, die nun als virtuelle Videogeräte für die Entwicklung videosignalverarbeitender Komponenten zur Verfügung stehen. Die sehr gute Übereinstimmung mit realen Videogeräten verdanken diese ausführbaren Spezifikationen der bidirektionalen Videosignalschnittstelle *VidTrans*. Das Verhaltensmodell eines rein digitalen PAL-Encoders erzeugt einen Videosignaldatenstrom, der sich bezüglich der erreichten Bildqualität, mit *VidTrans* auf einem analogen Farbfernseher dargestellt, fast nicht von der Bildqualität handelsüblicher Videokameras unterscheidet. Das Verhaltensmodell eines digitalen PAL-Decoders liefert ebenfalls Bilder von sehr hoher Bildqualität, wenn das Modell auch etwas empfindlicher als analoge Farbfernseher auf Rauschen reagiert. Bei der Validierung eines digitalen PAL-Encoders ist die stärkere Rauschempfindlichkeit des implementierten PAL-Decoders jedoch ein Vorteil: Liefert die ausführbare Spezifikation des PAL-Decoders, anhand von durch Schaltungssimulation eines PAL-Encoder-Entwurfs gewonnenen Videosignalsequenzen, Bilder mit gewünschter Bildqualität, so ist der betreffende PAL-Encoder-Entwurf in Bezug auf analoge Farbfernsehgeräte auf der sicheren Seite.

Die, auf den bei der Software-Implementierung eines rein digitalen PAL-Encoders gewonnenen Erkenntnissen, basierende Hardware-Implementierung eines digitalen PAL-Encoders mit Hilfe eines FPGAs verdeutlicht in anschaulicher Weise die Schwierigkeiten, die beim Übergang von einer Software-Implementierung auf eine Hardware-Implementierung aufgrund von notwendigen numerischen Vereinfachungen auftreten.

Mit *VidTrans* kann das Verhalten von videosignalverarbeitenden Komponenten auf der Systemebene beobachtet werden, ähnlich wie die Beobachtung des Lernens von Backpropagation-Netzwerken durch *Lerntrajektorien* auf einer hohen Abstraktionsebene. Durch Beobachtung des Verhaltens auf der Systemebene und Bewertung kann die den Simulationen zugrunde liegende Spezifikation mit der Systemumgebung abgeglichen und verfeinert werden, bis die Spezifikation einer in der Entwicklung befindlichen Komponente den gestellten Anforderungen genügt. Insbe-

sondere lassen sich mit *VidTrans* die Auswirkungen von für den Schaltungsentwurf notwendigen numerischen Vereinfachungen auf der Systemebene direkt beobachten und so bewerten.

4.3.1 Digitale Videosignalverarbeitung

Erste Untersuchungen zur rein digitalen Extraktion von Videobildern aus einem digitalisierten analogen Videosignaldatenstrom wurden im Rahmen einer vom Autor betreuten Studienarbeit [Hahn, 1995] durchgeführt. Durch diese Ergebnisse bezüglich der digitalen Extraktion von Graubildern ermutigt, wurde die digitale Verarbeitung analoger Farbvideosignale im Rahmen einer vom Autor betreuten Studienarbeit [Jürgens, 1996] und einer Diplomarbeit [Jürgens, 1999] untersucht. Zu Beginn der Untersuchungen zur rein digitalen Verarbeitung analoger Farbvideosignale gab es von keinem Halbleiterhersteller rein digitale Farbvideosignal-Encoder oder rein digitale Farbvideosignal-Decoder. Das hat sich mittlerweile geändert. So bieten mehrere große Halbleiterhersteller derartige Komponenten an [Fairchild Semiconductor, 1998; Harris Semiconductor, 1998; OKI Semiconductor, 1998; Philips Semiconductors, 1996a; Sony Semiconductor, 1998; Texas Instruments, 1998]. Die Erarbeitung von neuen Ansätzen ist für zukünftige Entwicklungen sinnvoll, da die steigende Integrationsdichte, insbesondere von programmierbaren Logikbausteinen, in steigendem Maße die Entwicklung anwendungsspezifischer Entwürfe betrifft und durch Integration digitaler PAL-Encoder und digitaler PAL-Decoder neue Anwendungsbereiche auch für anwendungsspezifische Entwicklungen erschlossen werden können. Auch wenn der Trend bei Videogeräten zur Digitaltechnik geht, stehen analoge Farbfernseher doch als fast allgegenwärtige graphische Ausgabegeräte zur Verfügung.

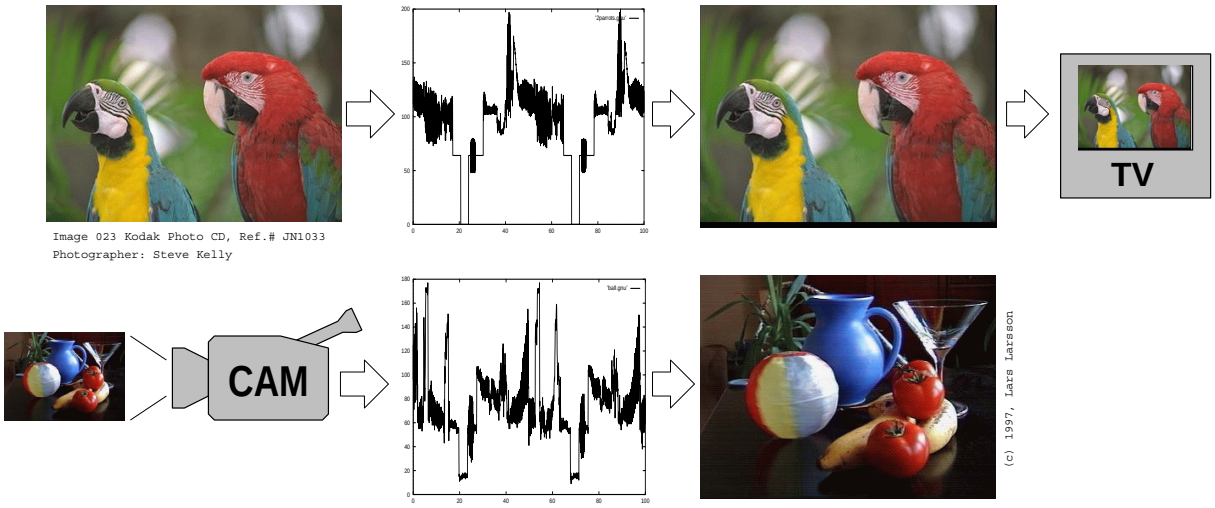


Abbildung 4.24: **Systemnahe Simulation von PAL-Encoder und PAL-Decoder.** *Erst der Austausch von Videosignalen zwischen Simulationen und analogen Videogeräten ermöglicht aussagekräftige Bewertung von Simulationsergebnissen. Die Signaleigenschaften des digital erzeugten Videosignals (oben) unterscheiden sich deutlich von den Signaleigenschaften (Rauschen, Offset, Verzerrung) des digitalisierten analogen Videosignals (unten). Beide Videosignale besitzen jedoch im Systemkontext Gültigkeit.*

Mittlerweile findet man zwar Literatur zur digitalen Videosignalverarbeitung [Jack, 1996] und zur digitalen Fernsehtechnik [Reimers, 1997]. Zur Implementierung sind jedoch derzeit noch eigene Untersuchungen und Entwicklungen notwendig. So ist zur Implementierung digitaler videosignalverarbeitender Komponenten das Studium von Literatur zu Grundlagen der analogen Fernsehtechnik [Limann, 1978; Kirsch, 1993; Morgenstern, 1994; Mäusel, 1995] für die Erstellung einer Spezifikation hilfreich. Werke wie [Jack, 1996] und [Reimers, 1997] beschreiben zwar detailliert die Funktionsweise videosignalverarbeitender Komponenten, jedoch nicht direkt in algorithmischer Form, die für eine direkte Umsetzung in Signalverarbeitungsalgorithmen zur Software-Implementierung oder zur Hardware-Implementierung wünschenswert wäre. Außerdem ist eine direkte Übertragung von Blockschaltbildern (s. Abb. 2.16, S. 43 und Abb. 2.17, S. 45) in eine Schaltungsbeschreibungssprache aufwendiger als eine Verhaltensbeschreibung auf algorithmischer Abstraktionsebene, wie beim Vergleich der Vorgehensweise bei der Implementierung von *VidTrans* [Larsson, 1996] mit der Vorgehensweise bei der Implementierung von *PalCo* [Larsson, 1999b] deutlich wurde. Die Implementierungen *VidTrans* und *PalCo* sind im Kapitel 5 beschrieben. Bei der Implementierung von *VidTrans* wurde zunächst ein Blockschaltbild auf Register-Transfer-Ebene erstellt und anschließend in VHDL beschrieben. Dagegen wurde bei der Implementierung von *PalCo* auf ein Blockschaltbild verzichtet – vielmehr wurde das Verhalten

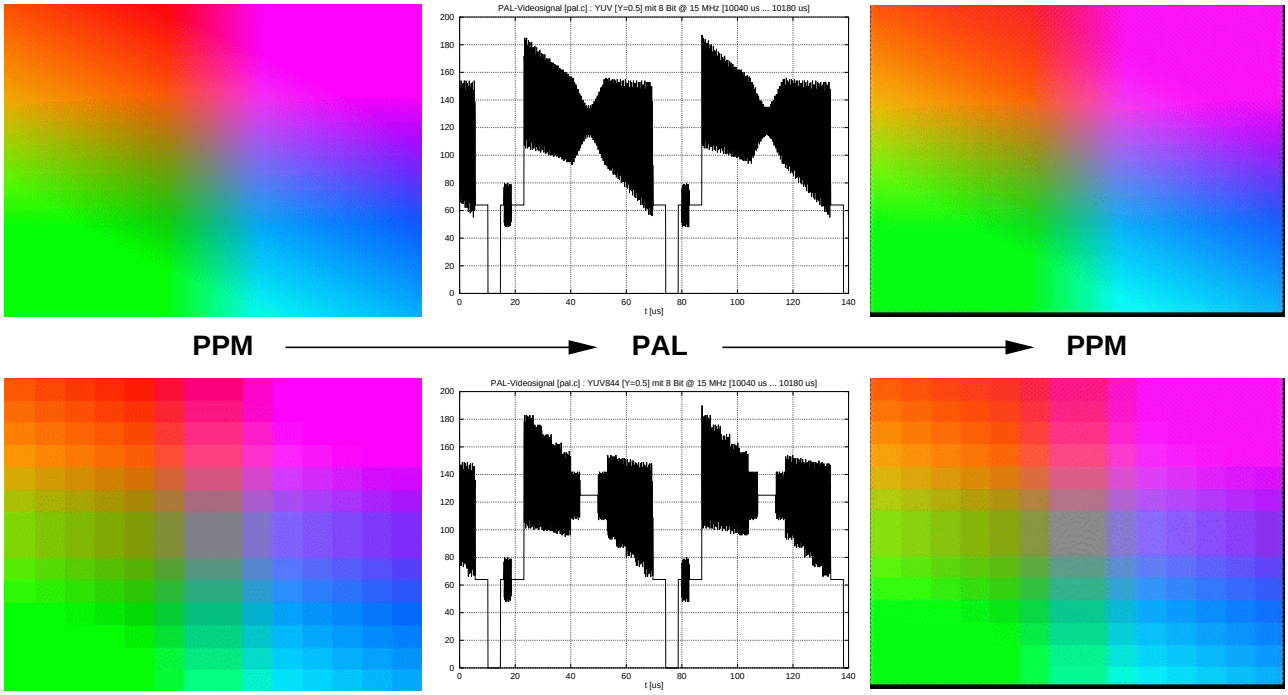


Abbildung 4.25: **Software-PAL-Encoder und Software-PAL-Decoder** *zur Evaluation der Videosignalverarbeitungsaufgabe. Im oberen Teil der Abbildung wurde eine Ebene des YUV-Farbraums mit einem kontinuierlichen Farbverlauf in ein PAL-Videosignal konvertiert. Im unteren Teil der Abbildung wurde die gleiche Ebene in ein PAL-Videosignal konvertiert, jedoch mit der Darstellungsgenauigkeit von u und v, wie sie bei der VHDL-Implementierung des PAL-Encoders verwendet wurde.*

eines PAL-Encoders, wie im Abschnitt 2.4.1 nicht-formal, verbal spezifiziert und dann direkt in eine synthetisierbare VHDL-Beschreibung umgesetzt.

Die textuelle Verhaltensbeschreibung mit einer Schaltungsbeschreibungssprache wie VHDL mit darauf aufsetzender Logiksynthese stellt gegenüber der Schaltplaneingabe einen Paradigmenwechsel dar. Die Vorteile textueller Schaltungsbeschreibungen kommen erst dann zum Tragen, wenn der Entwerfende sich dieses Paradigmenwechsels bewußt wird und bei der Beschreibung einer Schaltung mit funktionalen Grundelementen – Logikgattern, Flipflops, Registern, Schieberegistern, Zählern, etc. – zu abstrakteren, algorithmischen Verhaltensbeschreibungen einer Schaltung übergeht. Dabei ist zu beachten, daß das, was beim Entwurf als Verhaltensbeschreibung formal spezifiziert wird, nach der Logiksynthese eine Schaltungsstruktur ergibt, die das Verhalten der spezifizierten Verhaltensbeschreibung repräsentiert, die sich jedoch aus funktionalen Grundelementen – Logikgattern, Flipflops, Registern, Schieberegistern, Zählern, etc. – zusammensetzt.

Ein besonderer Vorteil bei der Verwendung eines durch FPGA-Prototyping validierten VHDL-Modells eines PAL-Encoders anstelle eines handelsüblichen digitalen PAL-Encoders, ist die Möglichkeit der Wiederverwendbarkeit für SoC-Designs. Darüber hinaus bietet es Unabhängigkeit von Herstellern digitaler PAL-Encoder, was bei Verwendung des betreffenden VHDL-Modells anstelle eines Chips unter Umständen ein Redesign bei Einstellung der Produktion eines verwendeten Chips vermeiden hilft [Stogdill, 1999]. Außerdem ermöglichen solche VHDL-Modelle Systemsimulationen, wie sie mit handelsüblichen Chips nur dann möglich sind, wenn die Hersteller der betreffenden Chips dem Anwender entsprechende Verhaltensmodelle zu Verfügung stellen würden. Die Verwendung von FPGAs für Prototyping, durch Einbau von FPGAs in ein Zielsystem, bietet zusätzlich die Möglichkeit, einen digitalen Entwurf in realer Systemumgebung unter realen Echtzeitbedinungen zu validieren.

Digitale Farbvideosignalerzeugung

Die digitale Erzeugung eines PAL-Farbvideosignals ist eine notwendige Vorarbeit für die Realisierung rein digitaler Extraktion von Farbbildern aus einem digitalisierten Videosignaldatenstrom, um die zeitliche Struktur des PAL-Videosignals bis ins letzte Detail analysieren zu können. Darüber hinaus wird bei der Implementierung digitaler PAL-Encoder die Bedeutung systemnaher Validierung durch Beobachtung von Systemverhalten bei der Wechselwirkung mit einer realen Systemumgebung und realen Systemkomponenten – hier Farbfernseher – besonders deutlich. Die digitale Erzeugung eines PAL-Farbvideosignals kann jedoch als eine eigenständige Entwurfsaufgabe verstanden werden. Der im Rahmen der vorliegenden Arbeit implementierte digitale PAL-Decoder, der seinerseits mit Hilfe von *VidTrans* validiert wurde, steht nun als Simulationsmodell als Ersatz für reale Farbfernsehgeräte zur Validierung von digitalen PAL-Encodern zur Verfügung. Nun steht dieser digitale PAL-Decoder als Grundlage für die Weiterentwicklung hin zu einem generischen Verhaltensmodell analoger Farbfernseher zur Verfügung, wodurch sich der Entwurfsvorgang und die Validierung digitaler PAL-Encoder erheblich vereinfachen ließe. Außerdem besteht im Bereich der digitalen Videosignalverarbeitung Interesse, Qualitätsmeßgeräte für die qualitative Beurteilung von Videogeräten zu haben [Hewlett-Packard, 1997]. Die Kombination von neuronalen Netzwerken und digitaler Videosignalverarbeitung, wie im Rah-

men der vorliegenden Arbeit erforscht, kann dafür als wissenschaftliche Grundlage dienen.

Die größte Schwierigkeit beim ersten Schritt der Realisierung eines digitalen PAL-Encoders besteht zunächst einmal darin, ein korrektes Zusammenspiel verschiedener Signalkomponenten eines PAL-Videosignals zu realisieren. Dabei ist das Zusammenspiel notwendiger Signalkomponenten untereinander derart, daß etablierte Entwurfsstrategieen – Hierarchie, Regularität, Modularität und Lokalität [Weste und Eshraghian, 1993] – nicht fruchten, da die Erzeugung eines PAL-Videosignals ein vielschichtiges Problem ist. Erst ein weitgehend vollständig implementierter PAL-Encoder erzeugt ein PAL-Videosignal, das überhaupt eine Bewertung der Implementierung erlaubt. Die Bewertung des PAL-Videosignals erfordert darüber hinaus Beobachtungszeitintervalle, die sich über viele Zeitskalen (≈ 100 ns $\ldots 100$ ms) erstrecken.

Die im Rahmen der vorliegenden Arbeit konzipierte und realisierte Software-Implementierung und Hardware-Implementierung digitaler PAL-Encoder weisen die in Abbildung 4.26 skizzierte Blockstruktur auf. Diese Blockstruktur ist für den Entwurf digitaler Schaltungen angepaßt und keine direkte Umsetzung einer analogen PAL-Encoder-Architektur (vgl. Abb. 2.16, S. 43) aus der Literatur [Mäusel, 1995] in eine digitale Architektur.

Die digitalen PAL-Encoder bestehen aus vier funktionalen Komponenten. Die Komponente TIMER hat die Funktion eines Steuerwerks, und die Komponenten DDS, QAM und MIXER haben zusammen die Funktion eines Operationswerks. Die Komponente YUV-DEMO ist nicht Teil des PAL-Encoders, sondern dient bei der Hardware-Implementierung des FPGA-Prototypen nur als Bildmustergenerator. Dabei werden $u(x)$ und $v(y)$ als Funktion von Bildschirmspalte x und Bildschirmzeile y erzeugt, und die Luminanz $y(f)$ wird abhängig von einem Bildzähler f variiert. Damit wird der gesamte adressierbare YUV-Farbraum zyklisch durchlaufen und dargestellt.

Der als Software implementierte PAL-Encoder arbeitet auf Basis der RGB-Darstellung digitaler Bilddateien – die lineare Farbraumtransformation (2.43) ist darin integriert. Bilder im

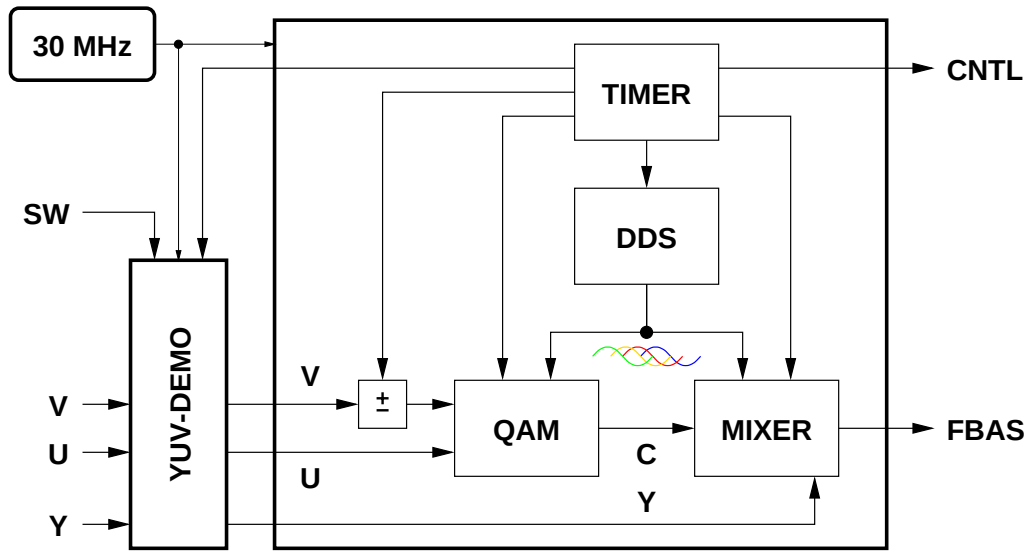


Abbildung 4.26: **Implementierte PAL-Encoder-Architektur** *[Larsson, 1999b]*

PPM-Format (portable pixmap file format, Jef Poskanzer, 1989) können vom Software-PAL-Encoder direkt verarbeitet werden. Dadurch ist es mit dem PAL-Encoder möglich, Realbilder direkt in ein PAL-Videosignal zu überführen und dieses über die Videosignalschnittstelle *VidTrans* direkt auf analogen Farbfernsehgeräten zu bewerten. Bei der Hardware-Implementierung des digitalen PAL-Encoders *PalCo* als FPGA-Prototyp stand zunächst die Validierung der Implementierung in Bezug auf das Zusammenspiel der Teilkomponenten und eine Aufwandsabschätzung im Vordergrund. Daher wurde zunächst auf die Hardware-Implementierung der linearen Farbraumtransformation (2.43) verzichtet. Außerdem liegen digitale Bilddatenströme nach dem MPEG-2-Standard auf Basis der YUV-Farbraumrepräsentation [Jack, 1996; Reimers, 1997] vor. MPEG-2 ist derzeit der etablierte Standard für digitale Fernsehübertragung bei Rundfunkanstalten [Hewlett-Packard, 1997].

Die Erzeugung des PAL-Farbvideosignals durch die implementierten PAL-Encoder erfolgt nach folgendem Schema

$$\begin{aligned}
\text{fbas}(t) &= y(t) + c(t) + black + \text{burst}(t, \Phi(t)) - \text{sync}(t) \\
c(t) &= u(t) \cdot \text{dds}(t, 0^o) \pm v(t) \cdot \text{dds}(t, 90^o)) \\
black &= konstant \\
\text{burst}(t) &= \text{gate}(t) \cdot \text{dds}(t, \pm 135^o) \\
\text{csync}(t) &= \text{hsync}(t) \oplus \text{vsync}(t) \\
\text{sync}(t) &= black \cdot (\text{csync}(t) \wedge (\text{preamble}(t) \vee \text{postamble}(t)))
\end{aligned} \tag{4.40}$$

Dabei erfolgt die Auswahl der Signalkomponenten durch das Modul TIMER. Das Zeilensprungverfahren (vgl. Abschnitt 2.4.1, S. 39) impliziert ein halbzeiliges H-Sync-Schema, so daß intern ein "halbzeiliges" H-Sync-Schema Verwendung findet. Das Modul TIMER erzeugt Steuersignale von der Dauer einer Taktperiode, die Signalgeneratoren starten, welche die benötigte Signalfolgen erzeugen. So erzeugt ein Teilmodul autonom, getriggert vom Modul TIMER durch ein Signal `new_line`, die zur Erzeugung einer Bildzeile benötigten Signalfolge. Ein anderes Teilmodul erzeugt, getriggert von einem Signal `new_frame`, eine V-Sync-Sequenz zusammen mit Preamble- und Postamble-Signalen (Vor- und Nachtrabanten [Kirsch, 1993; Mäusel, 1995]). Die Komponente DDS erzeugt sinusförmige Signale, wie im Abschnitt 2.3.4 ab S. 35 beschrieben. Durch die Komponente MIXER erfolgt das Zusammenmischen der verschieden PAL-Videosignalkomponenten.

Vereinfachungen zur Hardware-Implementierung des PAL-Encoders

Die Farbdifferenzsignale u und v werden üblicherweise vor der QAM-Modulation auf den Farbträger bandbegrenzt [Kirsch, 1993; Morgenstern, 1994; Mäusel, 1995], da die Übertragung des Chrominanz-Signals $c(t)$ mit geringerer Bandbreite als die Übertragung des Luminanz-Signals $y(t)$ erfolgt, was wahrnehmungsphysiologisch aufgrund der geringeren Farbauflösung im Vergleich zur Helligkeitsauflösung des menschlichen Auges gerechtfertigt ist [Morgenstern, 1994]. Auf die Bandbegrenzung der Signale u und v wurde jedoch bei den Implementierungen der digitalen PAL-Encoder verzichtet, um den Implementierungsaufwand – bzgl. Verarbeitungszeit bei der Software-Implementierung und bzgl. der Komponentenzahl bei der Hardware-Implementierung – zu reduzieren. Durch diese Vereinfachung werden Filter eingespart, deren

Implementierungsaufwand leicht den Implementierungsaufwand der PAL-Encoder selbst um ein Vielfaches übersteigt. Beim Software-PAL-Decoder wird nämlich der Hauptanteil der Rechenzeit für Berechnung diskreter Faltungen (2.23) zur Repräsentation von FIR-Filtern benötigt. Im Falle der Hardware-Implementierung in Form einer echtzeitfähigen Architektur gemäß Abbildung 2.8, S. 29 ergibt sich ein entsprechend hoher Schaltungsaufwand zur Realisierung vieler Multiplizierer und Register. Der Schaltungsaufwand läßt sich zwar durch Multiraten-Filter und geschickte Wahl der Übergangsfrequenzen in Form von (kaskadierten) Halbbandfiltern deutlich reduzieren [Fliege, 1993], weist aber auch dann noch einen vergleichsweise hohen Implementierungsaufwand auf, wenn man bedenkt, daß für die Implementierung einer Stufe eines FIR-Filters ein ähnlicher Realisierungsaufwand wie für einen QAM-Modulator (2.44) gemäß der linken Seite der Abbildung 2.10 auf Seite 33 erforderlich ist.

Für die *Direkte Digital Synthese* (DDS) wurde zur Hardware-Implementierung eine Sinus-Tabelle mit zunächst nur 32 Werten und 5 Bit Auflösung gewählt, die als *Schaltnetz* in VHDL beschrieben wurde, weil damit die Integration in einem einzigen FPGA mit nur 10.000 Gatter-Äquivalenten (FPGA Typ Altera EPF10K10LC84 [Altera Corporation, 1998b]) möglich war. Dadurch ergibt sich zwar eine Bildqualität, die schlechter ist, als die mit der Software-Implementierung des digitalen PAL-Encoders erreichte Bildqualität. Da aber bei der Hardware-Implementierung des digitalen PAL-Encoders die Machbarkeit, die Aufwandsabschätzung und der Vergleich mit der Software-Implementierung durch Beobachtung von Verhalten der Implementierungen auf der Systemebene im Vordergrund stand, war die geringere Bildqualität der Hardware-Implementierung zunächst von untergeordneter Bedeutung. Eine mit der Software-Implementierung vergleichbare Bildqualität wäre allenfalls für einen entsprechenden kommerziellen Entwurf von Bedeutung gewesen.

Digitale Videofarbbildextraktion

Die Farbbildextraktion aus einem analogen digitalisierten PAL-Farbvideosignal mit Methoden der digitalen Signalverarbeitung ist in Bezug auf die Robustheit einer Implementierung aus mehreren Gründen schwieriger zu realisieren als die digitale PAL-Farbvideosignalerzeugung. Über das Quantisierungsrauschen hinaus kann ein analoges PAL-Videosignal durch Rauschquellen, die in den analogen Signalpfad einstrahlen, zusätzlich gestört sein. Starkes Rauschen verschlechtert nicht nur die Qualität extrahierter Bilder, sondern kann auch die Bildsynchronisation erheblich stören und sogar verhindern, wenn das vom FBAS-Signal separierte C-Sync-Signal – als unverrauscht angenommen – durch einfache Automaten verarbeitet wird, die sich effizient als Hardware implementieren lassen. Niederfrequente Störungen können zu Fluktuationen des, als konstant angenommenen, Schwarzpegels führen. Die genaue Kenntnis des Schwarzpegels ist jedoch für die Rekonstruktion des Luminanz-Signals $y(t)$ und der Farbdifferenzsignale $u(t)$ und $v(t)$ erforderlich, da das FBAS-Signal durch Summation von konstantem Schwarzpegel, Luminanz-Signal $y(t)$ und Chrominanz-Signal $c(t)$ erzeugt wird und der Schwarzpegel zur Rekonstruktion des Signalgemisches $y(t) + c(t)$ vom FBAS-Signal abzuziehen ist. Außerdem ist der Pegelmittelpunkt der Burst-Signale gleich dem Schwarzpegel (vgl. Abbildung 2.15, S. 42).

Die Burst-Signale werden zur Rekonstruktion des PAL-Farbträgers und des von Bildzeile zu Bildzeile wechselnden Vorzeichens des Farbdifferenzsignals $v(t)$ benötigt. Die Frequenz

und Phase des PAL-Farbträgers ($f_{PAL} = 4,43361875$ MHz) ist bei der Videofarbbildextraktion sehr genau zu rekonstruieren, da schon geringe Phasenfehler zu wahrnehmbaren Farbfehlern führen [Mäusel, 1995]. Die Farbträgerfrequenz, eigentlich auf 0,25 Hz genau spezifiziert, kann aber deutlich abweichen, wie durch genaue Messungen festgestellt wurde. Die Rekonstruktion des Farbträgers kann mit Hilfe einer PLL (vgl. Abschnitt 2.3.3, S. 34) erfolgen. Werden dabei mögliche Abweichungen der Farbträgerfrequenz vom Sollwert nicht in Betracht gezogen, so führt das zur Fehlfunktion der betreffenden PLL, wenn etwa zur hardware-gerechten Vereinfachung eine feste und genaue Farbträgerfrequenz angenommen wird und nur die Phase des PLL-Oszillators verändert wird. So wurden mit *VidTrans* und einem Software-PAL-Decoder (namens *CFrame*) bei PAL-Videosignalen einiger Rundfunkanstalten Frequenzabweichungen im Bereich von $1 \ldots 10$ Hz gemessen. Bei CamCordern, die als Videosignalquellen verwendet wurden, wurden Abweichungen der Sollfrequenz des PAL-Farbträgersignals von über 100 Hz gemessen. Als Frequenzreferenz diente bei allen Videoquellen der 30.000.000-Hz-Oszillator von *VidTrans*, aus dem die Abtastfrequenz $f_s = 15$ MHz durch Teilung gewonnen wird.

Die Extraktion eines Farbbilddatenstroms $y(t), u(t), v(t)$ aus einem digitalisierten PAL-Videosignal $\text{fbas}(t)$ erfolgt bei der Software-Implementierung des digitalen PAL-Encoders nach folgendem Schema

$$\begin{aligned} y(t) &= \text{tp}(\text{fbas}(t) - \text{black}(t)) \\ u(t) &= 2.0 \cdot \text{tp}(((y(t) + c(t)) \cdot \text{plldds}(t,\ 0^o)) - y(t) \\ v(t) &= \pm 2.0 \cdot \text{tp}(((y(t) + c(t)) \cdot \text{plldds}(t, 90^o)) - y(t) \\ \text{csync}(t) &= \text{syncsep}(t) \\ \text{black}(t) &= \text{blacksep}(t) \\ \text{plldds}(t) &= \text{plldds}(\text{burst}(t), \text{gate}(t), \text{sync}(t)) \end{aligned} \tag{4.41}$$

Der im Rahmen der vorliegenden Arbeit als Software implementierte digitale PAL-Decoder weist die in Abbildung 4.27 dargestellte Blockstruktur aus. Dieser PAL-Decoder wurde jedoch vor

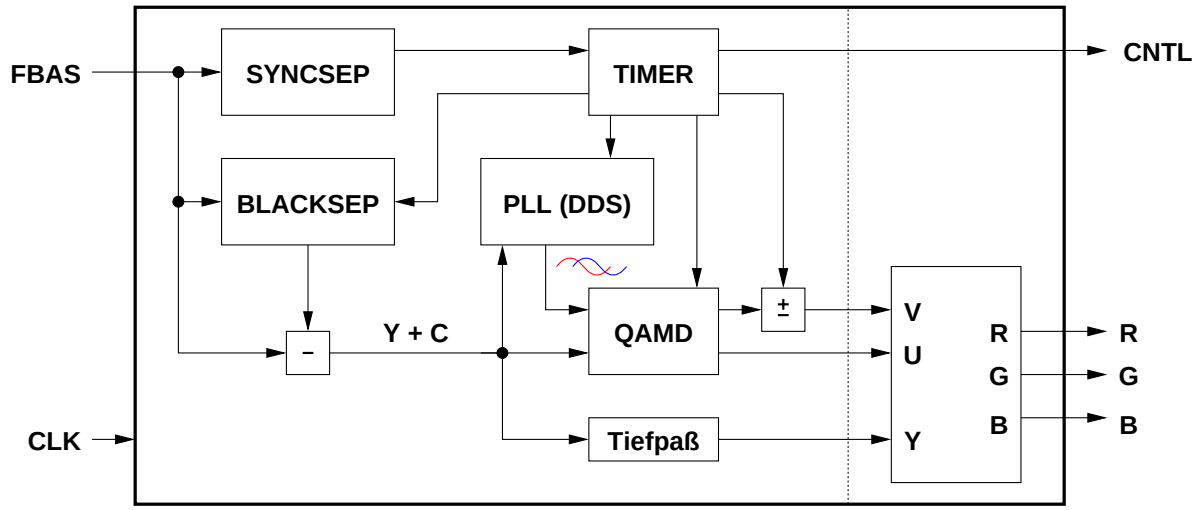


Abbildung 4.27: **Implementierte PAL-Decoder-Architektur**

dem Hintergrund einer Hardware-Implementierung realisiert, d.h. die Software-Implementierung entspricht funktional einer Hardware-Implementierung mit der dargestellten Blockstruktur.

Zunächst wird aus dem FBAS-Signal ein C-Sync-Signal durch die Komponente SYNCSEP (vgl. Abbildung 4.27) separiert. Dazu wird ein Schwellwert dynamisch und adaptiv bestimmt. Dieser Schwellwert läuft *langsam inkrementell* bis zu einem festgelegten Grenzwert, wenn das *Videosignal größer* als der aktuelle Schwellwert ist. Der Schwellwert läuft aber *schnell dekrementell* bis minimal auf null, wenn das *Videosignal kleiner* als der aktuelle Schwellwert ist. Dadurch variiert der Schwellwert in einem Bereich zwischen minimalem Videosignalpegel und Schwarzschulter. Mit einem Komparator, der den Schwellwert mit dem Videosignaldatenstrom vergleicht, wird ein 1-bit-breites Signal csync(t) gewonnen. Diese Aufgabe erledigt die Komponente SYNCSEP. Der aktive Pegel ist dabei auf null festgesetzt.

Aus dem digitalen csync(t) werden durch geeignete Automaten zunächst H-Sync und V-Sync extrahiert. Mit einer fallenden Flanke des C-Syncs wird dazu ein Zähler gestartet, der mit der Abtastfrequenz inkrementiert wird, so daß der Zählerstand ein Zeitintervall seit der fallenden Flanke des C-Syncs repräsentiert. Geht das C-Sync-Signal auf eins (inaktiv), wird der betreffende Zähler zurück gesetzt. Durch Auswertung des Zählerstandes wird zwischen H-Syncs und V-Syncs unterschieden. Ein Zähler wird mit der Abtastfrequenz inkrementiert, so daß dieser Bildpunkte innerhalb der Bildzeilen repräsentiert. Dieser Zähler wird mit Vorderflanken der H-Syncs zurückgesetzt. Ein Zeilenzähler wird mit den Vorderflanken der H-Syncs inkrementiert. Der Zeilenzähler wird durch V-Syncs zurückgesetzt. Damit lassen sich dann schon Graubilder aus einem digitalisierten Videosignal extrahieren. Diese Teilaufgabe übernimmt, neben weiteren Aufgaben, die Komponente TIMER. Der TIMER hat – wie bei den implementierten PAL-Encodern – die Funktion eines Steuerwerks. Die übrigen in Abbildung 4.27 dargestellten Komponenten SYNCSEP, BLACKSEP, PLL(DDS), QAMD, Tiefpaß, $\boxed{-}$ und $\boxed{\pm}$ haben zusammen die Funktion eines Operationswerks.

Für die Extraktion der Farbinformation ist die Rekonstruktion des Farbträgers erforderlich. Dazu ist das Zeitfenster der Burst-Signale innerhalb einer Zeile zu extrahieren, damit ein Farbträgeroszillator während der Burst-Zeitfenster (Burst-Gate) synchronisiert werden kann. Der Farbträgeroszillator im PAL-Decoder wird nur während des kurzen Burst-Zeitfensters synchronisiert und läuft während der übrigen Zeit frei. An das Burst-Zeitfenster anschließend, wird die Schwarzschulter des Videosignals durch die Komponente BLACKSEP bestimmt, gespeichert und steht für die weitere Signalverarbeitung während der sichtbaren Zeile zur Verfügung.

Mit dem rekonstruierten PAL-Farbträger, der von der Komponente PLL(DDS) zur Verfügung gestellt wird, kann die QAM-Demodulation des Chrominanz-Signals c(t) gemäß (2.33), wie in Abschnitt 2.3.2, S. 32 beschrieben, erfolgen. Dazu ist das Chrominanz-Signal $c(t)$ mit $\cos(2\pi \cdot f_{PAL})$ und mit $\sin(2\pi \cdot f_{PAL})$ zu multiplizieren. Um die QAM-Demodulation direkt mit dem Chrominanz-Signal $c(t)$ zu machen, ist das Chrominanz-Signal $c(t)$ zunächst durch einen Hochpaß aus dem FBAS-Signal herauszufiltern (vgl. Abb. 2.17, 45). Die Farbdifferenzsignale $u(t)$ und $v(t)$ werden durch Tiefpaßfilterung der Signale $c(t) \cdot cos(\omega_{PAL} \cdot t)$ und $c(t) \cdot \sin(\omega_{PAL} \cdot t)$ rekonstruiert, wobei $\cos(\omega_{PAL} \cdot t)$ und $\sin(\omega_{PAL} \cdot t)$ den in (Kreis-)Frequenz $\omega_{PAL} = 2\pi \cdot f_{PAL}$ und Phase rekonstruierten PAL-Farbträger repräsentieren. Die QAM-Demodulation der Farbdifferenzsignale erfolgt durch die Komponente QAMD.

Wesentlicher Bestandteil analoger PAL-Decoder ist der *PAL-Laufzeit-Decoder*. Die Funktionsweise des PAL-Laufzeit-Decoders ist in der Literatur [Limann, 1978; Kirsch, 1993; Morgenstern, 1994; Mäusel, 1995] eingehend beschrieben. Im folgenden werden daher nur die für eine Hardware-Implementierung relevanten Parameter und die sich daraus ergebenden technischen Randbedingungen skizziert. Die Entwurfsentscheidung für oder gegen die Hardware-Implementierung eines PAL-Laufzeit-Decoders hat für die Hardware-Implementierung eines PAL-Decoders weitreichende Konsequenzen. Ohne einen PAL-Laufzeit-Decoder ähnelt ein PAL-Decoder eher einem NTSC-Decoder. Der Verzicht auf den PAL-Laufzeit-Decoder bei der Implementierung eines PAL-Decoders stellt zwar einen erheblichen Eingriff bezüglich der Robustheit des PAL-Decoders dar, reduziert aber den Realisierungsaufwand deutlich, da auf die (digitale) Zwischenspeicherung einer Bildzeile und auf die dabei unter Umständen notwendige Interpolation von Abtastwerten verzichtet werden kann. Ob sich der Verzicht auf einen PAL-Laufzeit-Decoder rechtfertigen läßt, konnte anhand von Simulationen mit realen Videosignalen untersucht werden.

Im PAL-Laufzeit-Decoder wird das Chrominanz-Signal $c(t)$ um die Dauer einer Bildzeile verzögert. Der PAL-Laufzeit-Decoder verarbeitet das Chrominanz-Signal $c(t)$ separat. Dazu muß das Chrominanz-Signal $c(t)$ zur Verarbeitung durch den PAL-Laufzeit-Decoder zunächst durch Hochpaßfilterung aus dem FBAS-Signalgemisch extrahiert werden. Die Verzögerungszeit $\tau_{PAL-Laufzeit-Decoder} = 63,94325$ μs beträgt *genau* ein halbzahliges Vielfaches [Morgenstern, 1994] der Periodenlänge des Farbträgers ($283,5$ PAL-Farbträgerperioden, $f_{PAL} = 4,43361875$ MHz), oder die Verzögerungszeit $\tau_{PAL-Laufzeit-Decoder} = 63,056$ μs beträgt *genau* ein ganzzahliges Vielfaches [Mäusel, 1995] der Periodenlänge des Farbträgers (284 PAL-Farbträgerperioden, $f_{PAL} = 4,43361875$ MHz). Bei der Software-Implementierung des PAL-Laufzeit-Decoders wurde eine Verzögerung um $283,5$ Perioden des Farbträgers realisiert. Das verzögerte Chrominanz-Signal $c_d(t) = c(t - \tau_{PAL-Laufzeit-Decoder})$ wird im PAL-Laufzeit-Decoder zum unverzögerten Chrominanz-Signal $c(t)$ addiert bzw. subtrahiert

$$\begin{aligned} c_d(t) &= c(t - \tau_{PAL-Laufzeit-Decoder_{283,5}}) \\ c_u(t) &= \tfrac{1}{2} \cdot (c(t) + c_d(t)) \\ c_v(t) &= \tfrac{1}{2} \cdot (c(t) - c_d(t)) \end{aligned} \quad . \tag{4.42}$$

Die Signale $c_u(t)$ und $c_v(t)$ sind die durch den PAL-Laufzeit-Decoder phasenfehlerkompensierten Farbdifferenzsignale $u(t)$ und $v(t)$. Das PAL-Verfahren ist ein gegenüber Phasenfehlern des PAL-Farbträgers *robustes* Übertragungsverfahren. Das PAL-Verfahren ist gegen Phasenfehler von mehr als 60^o unempfindlich [Morgenstern, 1994], unter der Annahme, daß sich die Chrominanz aufeinander folgender Zeilen nicht oder allenfalls nur geringfügig unterscheidet. Die Ausdrücke (4.42) bzw. (4.43) sind entsprechenden Blockschaltbildern aus der Literatur [Limann, 1978; Kirsch, 1993; Morgenstern, 1994; Mäusel, 1995] entnommen. Dabei ist jedoch zu beachten, daß der Ausdruck (4.42) für eine Verzögerung um $283,5$ Perioden des Farbträgers gültig ist. Bei einer Verzögerung um 284 Perioden sind die Vorzeichen zu ändern

$$\begin{aligned} c_d(t) &= c(t - \tau_{PAL-Laufzeit-Decoder_{284}}) \\ c_u(t) &= \tfrac{1}{2} \cdot (c(t) - c_d(t)) \\ c_v(t) &= \tfrac{1}{2} \cdot (c(t) + c_d(t)) \end{aligned} \quad . \tag{4.43}$$

Dieses Implementierungsdetail sei an dieser Stelle erwähnt, da falsche Vorzeichen in (4.42) bzw. in (4.43) zur fast vollständigen Auslöschung der Farben, aus einem digitalisierten PAL-Videosignaldatenstrom, extrahierter Bilder führen. Ein solcher Vorzeichenfehler bei der Implementierung digitaler PAL-Decoder fällt zwar bei der Beobachtung des Verhaltens der Implementierungen auf der Systemebene deutlich auf, würde aber ohne direkte Darstellung extrahierter Farbbilder mit großer Wahrscheinlichkeit verborgen bleiben.

Zwecks Evaluation wurde ein PAL-Laufzeit-Decoder für die Software-Implementierung eines PAL-Decoders realisiert. Um die Verzögerung um eine Bildzeile zu erreichen, wurde ein Ringpuffer realisiert, der genau eine Bildzeile speichern kann. Um die Unabhängigkeit bei der Wahl der Systemtaktfrequenz, so wie bei der Erzeugung des Farbträgersignals durch Direkte Digital Synthese (DDS), zu erhalten, wurde die lineare Interpolation von Abtastwerten, die zwischen den äquidistanten Abtastwerten liegen, implementiert. Der Ringpuffer verzögert das Chrominanz-Signal um ein ganzzahliges Vielfaches der Abtastfrequenz ($f_s = 15$ MHz). Für den PAL-Laufzeit-Decoder muß das Signal jedoch um ein ganz- oder halbzahliges Vielfaches der Periodenlänge des Farbträgersignals (f_{PAL} = 4,43361875 MHz) verzögert werden.

Es sind also Abtastwerte, die zwischen zwei auseinander folgenden Abtastzeitpunkten liegen, zu interpolieren. Diese Zwischenwerte lassen sich numerisch auf Basis des Abtasttheorems mit Termen der Form $\sin(x)/x$ interpolieren [Press *et al.*, 1992], da sich – nach dem Abtasttheorem – ein bandbegrenztes Signal aus vielen verschobenen gewichteten $\sin(x)/x$-Funktionen

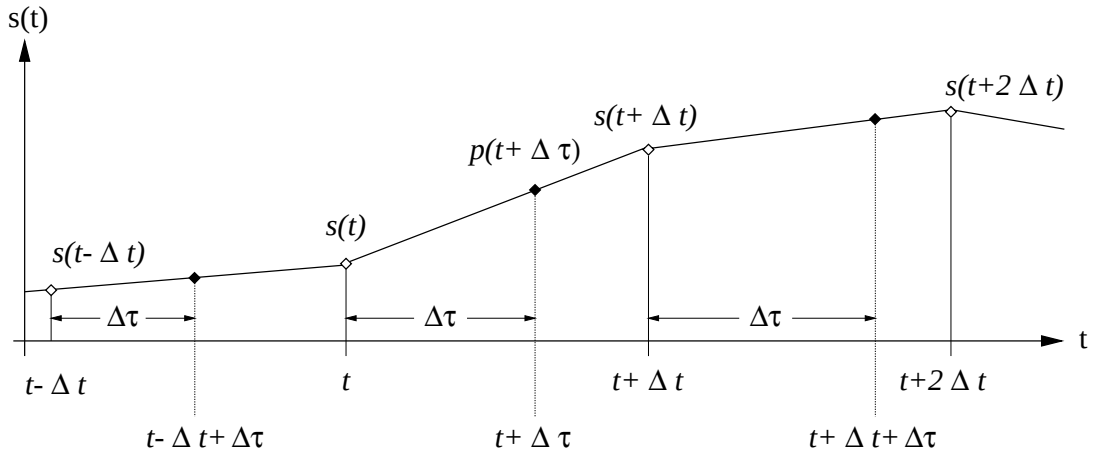


Abbildung 4.28: **Lineare Interpolation** *von Abtastwerten $p(t + \Delta\tau)$, die zwischen die zwischen zwei aufeinander folgenden Abtastwerten $s(t)$ und $s(t + \Delta t)$ liegen.*

zusammensetzen läßt [Girod *et al.*, 1997], was die Rückgewinnung eines, dem Abtasttheorem genügenden, bandbegrenzten Signals durch Interpolationsfilter [Hess, 1993] ist.

Es wurde jedoch zunächst eine lineare Interpolation von Zwischenwerten implementiert, da sich diese – vor dem Hintergrund einer möglichen Hardware-Implementierung – mit geringerem Aufwand als ein Interpolationsfilter realisieren läßt. Außerdem konnte auch schon ohne PAL-Laufzeit-Decoder eine Qualität extrahierter Farbbilder erreicht werden, die keine wahrnehmbaren Farbverfälschungen aufweisen. Bei der analogen Signalverarbeitung des Farbsignals können Phasenfehler auftreten, die ohne PAL-Laufzeit-Decoder zu Farbfehlern führen können [Morgenstern, 1994], die bei digitaler Signalverarbeitung nicht auftreten. Phasenfehler, die den Burst am Bildzeilenanfang und die darauf folgende Bildzeile gleichermaßen betreffen, können durch Synchronisation des rekonstruierten Farbträgers mit dem Burst-Signal am Bildzeilenanfang kompensiert werden, wenn sie den Phasenregelbereich des Farbträgergenerators nicht überschreiten. Außerdem wurde das PAL-Verfahren zu einer Zeit entwickelt, in der die analoge Signalübertragung noch nicht so weit fortgeschritten war [Morgenstern, 1994].

Zur linearen Interpolation (s. Abb. 4.28) von Abtastzwischenwerten $s(\Delta\tau)$, die zwischen zwei aufeinander folgenden Abtastwerten $s(t)$ und $s(t + 1/\Delta t)$, mit $\Delta t = 1/f_s$, liegen sollen, wird zunächst eine Zeitdifferenz $\Delta\tau$ berechnet, um die ein durch lineare Interpolation zu berechnender Abtastwert $s(t + \Delta\tau)$ von den abgetasteten, im Ringpuffer gespeicherten, Abtastwerten $s(t)$ und $s(t + 1/\Delta t)$ abweicht (s. Abb. 4.28). Die Berechnung der Zeitdifferenz $\Delta\tau$ könnte bei einer Hardware-Implementierung ähnlich wie die Akkumulation der Phasendifferenz $\Delta\varphi$ bei der Direkten Digital Synthese (DDS, vgl. Abschnitt 2.3.4, S. 35, ff.) erfolgen. Bei der Software-Implementierung wurde die Zeitdifferenz $\Delta\tau$ auf Basis von Fließkommazahlendarstellung akkumuliert und $\Delta\tau := \Delta\tau - \Delta t$ wenn $\Delta\tau > \Delta t$.

Hardware-gerechte Vereinfachungen des PAL-Decoders

Die Implementierung eines Software-PAL-Decoders erfolgte vor dem Hintergrund einer möglichen Hardware-Implementierung. Das Ziel der Software-Implementierung war daher die Evaluation von Systemparametern und die Bestimmung von Entwurfsentscheidungen. Im Rahmen der vorliegenden Arbeit stellt die Software-Implementierung eines digitalen PAL-Decoders ein Anwendungsbeispiel systemnaher Validierung eines Entwurfs durch Beobachtung von Verhalten auf der Systemebene dar.

Der Verzicht auf einen FIR-Filter zur Extraktion des Chrominanz-Signals $c(t)$ durch Hochpaßfilterung bewirkt bei einer Software-Implementierung eine Reduzierung der Verarbeitungszeit, bei einer Hardware-Implementierung bewirkt dieser Verzicht eine Ersparnis von Schaltungskomponenten. Eine einzige (testweise mit Altera MAX+plus II V. 9.23) synthetisierte MAC-Einheit eines FIR-Filters benötigt etwa 4000 Gatteräquivalente eines FLEX10K10-FPGAs (Altera Typ EPF10K10LC84-3 [Altera Corporation, 1998b]). Im Vergleich dazu benötigt die Hardware-Implementierung des digitalen PAL-Encoders (vgl. *PalCo*, Abschnitt 4.3.1, S. 120) insgesamt etwa 9000 Gatteräquivalente eines FLEX10K10-FPGAs. Wird dem QAM-Modulator anstatt des Chrominanz-Signals $c(t)$ das Signal $c(t) + y(t)$ zugeführt, so berechnet dieser anstatt der Farbdifferenz-Signale $u(t)$ und $v(t)$ zunächst Signale $u(t) + y(t) \cdot \cos(2\pi \cdot f_{PAL} \cdot t)$ und $v(t) + y(t) \cdot \sin(2\pi \cdot f_{PAL} \cdot t)$, wie sich durch Einsetzen von $c(t) + y(t)$ in (2.33) im Abschnitt

2.3.2, S. 32 leicht zeigen läßt. Durch die anschließende Tiefpaßfilterung werden dann die Terme $y(t) \cdot \cos(2\pi \cdot f_{PAL} \cdot t)$ und $y(t) \cdot \sin(2\pi \cdot f_{PAL} \cdot t)$, unterdrückt. Höherfrequente Anteile des Spektrums von $y(t)$, die im Bereich der Frequenz des Farbträgers liegen, werden zwar nicht unterdrückt und stören die rekonstruierten Farbdifferenz-Signale $u(t)$ und $v(t)$. Die Verkämmung der Spektren von $y(t)$ und $c(t)$ ist eine Schwäche des PAL-Verfahrens, und die daraus resultierenden Störungen werden als *Cross-Color* bzw. *Cross-Luminance* bezeichnet [Mäusel, 1995]. Diese Störungen treten aber auch dann auf, wenn das Chrominanz-Signal $c(t)$ durch Hochpaßfilterung von $c(t) + y(t)$ dem QAM-Demodulator ohne das Luminanz-Signal $y(t)$ zugeführt wird, da höherfrequente Anteile des Spektrums von $y(t)$, die im Bereich der Frequenz des Farbträgers liegen, von einer solchen Hochpaßfilterung nicht unterdrückt werden.

Der Verzicht auf einen PAL-Laufzeit-Decoder erspart bei einer Software-Implementierung eines PAL-Decoders etwas Verarbeitungszeit und einige KBytes Speicher. Sowohl die Zeitersparnis als auch die Speicherersparnis fallen aber bei einer Software-Implementierung nicht ins Gewicht. Ganz anders ist das bei einer Hardware-Implementierung. Da die Verzögerung des Chrominanz-Signals um eine Bildzeile bei einer Hardware-Implementierung in Echtzeit erfolgen muß, macht es jedoch keinen Sinn, von einer Ersparnis an Verarbeitungszeit zu sprechen – für die Verarbeitung eines Videosignalabtastwerts steht immer genau ein Abtastzeitintervall zur Verfügung, mit oder ohne PAL-Laufzeit-Decoder. Was die Zwischenspeicherung einer Bildzeile in einem Ringpuffer von einigen KByte Speicher betrifft, so besteht zur Software-Implementierung ein wesentlicher Unterschied. Einige KBytes Speicher müssen entweder als externer Speicher realisiert werden oder aber als On-Chip-Speicher. Dadurch wird der Entwurf aufwendiger. Bei der Verwendung von externem Speicher wird der Entwurfsablauf und dessen Handhabung aufwendiger, da dann ein Simulationsmodell eines RAMs erforderlich wird. Bei Verwendung von On-Chip-RAM ist der Einsatz von RAM-Generatoren erforderlich, und bestimmte Zieltechnologien, wie etwa die Verwendung von FPGAs, werden ausgeschlossen oder lassen sich nicht effizient zur Implementierung einsetzen. Der Verzicht auf den PAL-Laufzeit-Decoder spart bei einer Hardware-Implementierung sowohl RAM und macht darüber hinaus die Hardware-Implementierung einer Komponente zur Interpolation von Zwischenabtastwerten überflüssig.

Der Verzicht auf den PAL-Laufzeit-Decoder, der ein wesentliches Merkmal des PAL-Verfahrens darstellt, läßt sich durch Verarbeitung von analogen PAL-Videosignalen von unterschiedlichen Signalquellen und die visuelle Bewertung daraus extrahierter Farbbilder begründen. So waren sogar bei Farbbildern, die aus digitalisierten PAL-Videosignalen extrahiert wurden, die nur mit einer sehr einfachen Antenne terrestrisch empfangen wurden, fast keine Unterschiede zwischen Bildern, die mit PAL-Laufzeit-Decoder und Bildern, die ohne PAL-Laufzeit-Decoder verarbeitet wurden, wahrnehmbar.

4.3.2 Bewertungshilfsmittel

Die Beurteilung von Simulationsergebnissen zur Validierung von Implementierungen auf der Systemebene wurde erst durch die bidirektionale Schnittstelle *VidTrans* zum Austausch von Videosignalen zwischen Simulation und realer Systemumgebung ermöglicht. So konnten mit Hilfe von *VidTrans* sowohl computergenerierte Bilder als auch reale Bilder in den Software-Simula-

tionen, aber auch in den Hardware-Simulationen verwendet werden und haben so die systemnahe Validierung der verschiedenen Implementierungen erst ermöglicht. Die Validierung der Hardware-Implementierung des digitalen PAL-Encoders wäre anhand von Impulsdiagrammen ungleich schwerer oder vielleicht sogar unmöglich gewesen.

Computergenerierte Bilder und Realbilder weisen für die Validierung von videosignalverarbeitenden Implementierungen unterschiedliche Eignungen auf. Computergenerierte Bilder – mit feinen Farbverläufen – eignen sich für die Validierung von PAL-Encodern unter technischen Aspekten. Das heißt, es lassen sich Phasendrehungen und Quantisierungsartefakte, wie sie als Folge numerischer Vereinfachungen oder auch als Folge von Implementierungsfehlern auftreten können, beobachten und gegebenenfalls korrigieren. Diese berechneten Bilder eignen sich daher auch für die qualitative Beurteilung von Bildkompressionsalgorithmen und Bildreduktionsalgorithmen wie etwa nach dem JPEG-Standard [ISO/IEC Standard 10918-1, JPEG]. Derartige computergenerierte Farbbilder sind aber auch gut für den qualitativen Vergleich von Farbdruckern (Laserdrucker, Tintenstrahldrucker, etc.) insbesondere unter wahrnehmungsphysiologischen Gesichtspunkten geeignet. So werden bei der Darstellung der generierten Farbtestbilder auf Computermonitoren sogar Quantisierungsartefakte von Graphik-Karten deutlich, und Qualitätsunterschiede von derzeit üblichen Farbdarstellungen (8-Farbpalette, 16-Bit-HiColor, 24-Bit-TrueColor) werden besonders deutlich.

Realbilder eignen sich im Gegensatz dazu für abschließende Beurteilungen eines Gesamtergebnisses einer vorliegenden Implementierung. So ist etwa Hautfarbe von dargestellten Personen ein wichtiges Kriterium für die Qualitätsbeurteilung eines Entwurfs. Dabei ist die Möglichkeit, aus Schaltungssimulationen gewonnenen Videosignale in betrachtbare Bilder umzuwandeln von essentieller Bedeutung – anhand von Impulsdiagrammen wären derartige Beurteilungen nicht möglich. Bei Entwicklungen im Bereich *Digital Video Broadcasting* [Reimers, 1997] besteht der Wunsch nach maschinellen Bildqualitätsmeßgeräten, wie im Rahmen von *The 1997 Digital Video Test Symposium* im Mai 1997 in München deutlich wurde [Hewlett-Packard, 1997]. Die Kombination analytischer Methoden der digitalen Signalverarbeitung mit neuronalen Netzwerken und geeigneten Methoden zur Beobachtung von Verhalten auf der Systemebene könnte in diesem Zusammenhang als Grundlage für die Entwicklung solcher physiologisch-visuellen Qualitätsmeßgeräte dienen. Solche Meßgeräte könnten dann Einsatz bei der Entwicklung neuer Bildkompressionsalgorithmen und Bildreduktionsalgorithmen finden.

Für die Validierung erster Entwürfe wurden zunächst farbige Kreise berechnet. Farbwerte im YUV-Farbraum wurden derart als Funktionen lokaler Kreiskoordinaten berechnet, daß Phasenverschiebungen im PAL-Videosignalgemisch dabei durch Drehung der Kreisdarstellungen direkt sichtbar werden. Dabei ist der Drehwinkel ein direktes Maß für eine Phasenverschiebung. Vorzeichenfehler von PAL-Videosignalkomponenten werden durch Spiegelung der Kreisdarstellung ebenfalls direkt sichtbar.

YUV-Farbtestbilder zur Validierung auf der Systemebene

Bei der Berechnung der – zur Validierung auf der Systemebene eingesetzten – UV-Farbtestbilder erfolgte die Zuordnung von Farbanteilen (r, g, b) über Luminanz y und Farbdifferenzen u und v gemäß der in Abschnitt 2.4.1, S. 39 beschriebenen linearen Transformation (2.43). Die Farbdif-

ferenzsignale u und v werden dabei aus lokalen Kreiskoordinaten errechnet.

Bei den Farbkreisen (s. Abb. 4.29) werden die Farbdifferenzwerte u und v aus lokalen Koordinaten in Einheitskreisen berechnet. Die Luminanz y wurde bei den Farbkreisen konstant auf *eins* gesetzt oder als Funktion der Farbdifferenzsignale u und v errechnet. Dieses Testbild wurde mit dem Ziel konstruiert, in nur einem Bild solche Farbverläufe zur Verfügung zu stellen, die im Zusammenhang mit der digitalen Verarbeitung von PAL-Videosignalen besonders zur Validierung auf der Systemebene geeignet sind, indem die Farbverläufe technischen Randbedingungen Rechnung tragen, die sich aus der Struktur des PAL-Videosignals ergeben.

Stark gesättigte Farben, also Farben mit großen Beträgen von u und v, bei geringer Luminanz ($y \approx 0$), können zu Untersteuerung des PAL-Videosignals führen, da das Chrominanz-Signal dann eine hohe Amplitude aufweist und dem Schwarzpegel des PAL-Videosignals additiv überlagert wird. Das Chrominanz-Signal kann dann sogar Signalwerte erreichen, die unterhalb des Synchronsignalpegels liegen. Eine wenig robuste, digitale Bildsynchronisationsschaltung kann dadurch so stark gestört werden, daß die Bildsynchronisation dabei verloren geht. Bei der Implementierung digitaler PAL-Encoder kann es dagegen zu Zahlenbereichsunterschreitungen kommen, die zu begrenzen sind, da sonst stark gesättigte Farben geringer Luminanz fälschlich mit sehr hoher Luminanz dargestellt würden.

Bei stark gesättigten Farben hoher Luminanz ($y \approx 1$) kann es dagegen zu Übersteuerung des PAL-Videosignals kommen, was bei analogen TV-Geräten im allgemeinen zwar nicht zu Problemen führt, aber bei dem Entwurf digitaler PAL-Encoder zu berücksichtigen ist. Wird ein möglicher interner Zahlenbereichsüberlauf bei der Erzeugung eines digitalen PAL-Videosignaldatenstroms nicht begrenzt, so würden stark gesättigte Farben hoher Luminanz zu Signalpegeln in der Nähe des Synchronsignalpegels führen, die auch die Synchronisation analoger TV-Geräte empfindlich stören können. Wird bei analogen Farbfernsehgeräten die zur Elektronenstrahlerzeugung erforderliche Hochspannung direkt von den Spulen zur Horizontalablenkung gewonnen [Limann, 1978] und wird die Horizontalablenkfrequenz nicht im Fernsehgerät bandbegrenzt, so können falsche Synchronisationssignale im Videosignalgemisch im Extremfall die Lebensdauer

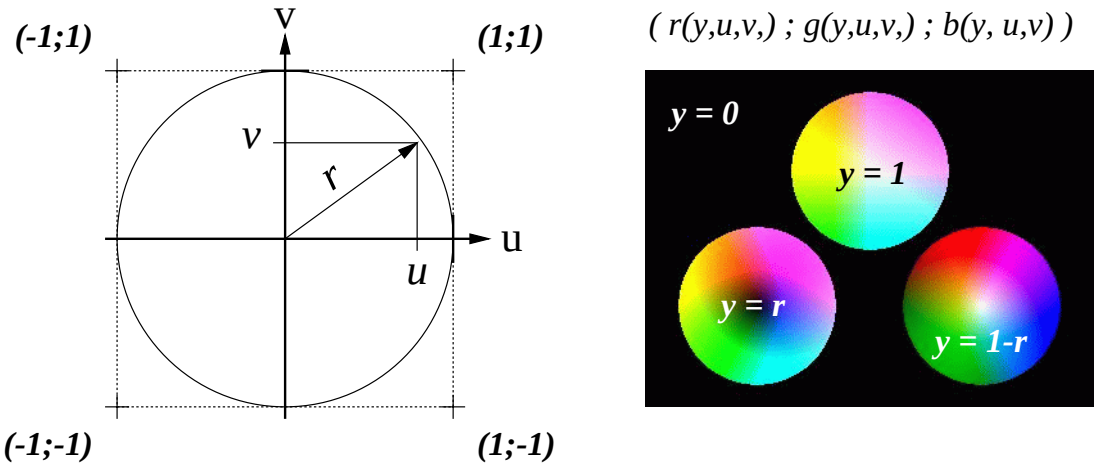


Abbildung 4.29: **Farbkreistestbild.** *Computergenerierte Farbkeise zur Validierung von Implementierungen digitaler PAL-Encoder und digitaler PAL-Decoder auf der Systemebene.*

des betreffenden Fernsehgeräts auf Minuten verkürzen.

Die Robustheit einer Implementierung in Bezug auf Untersteuerung und Übersteuerung eines PAL-Videosignals läßt sich mit geeigneten Farbtestbildern auf der Systemebene leicht evaluieren – anhand von Impulsdiagrammen digitaler Schaltungssimulationen, aber auch anhand von Oszillogrammen generierter PAL-Videosignale ist eine derartige Evaluation unter Umständen nicht zu erreichen. Dies setzt jedoch eine geeignete Schnittstelle zum Austausch von Signalen zwischen Simulation und realer Systemumgebung voraus – *VidTrans* [Larsson, 1996] ermöglicht eben gerade solchen Austausch.

Im Farbkreistestbild (Abb. 4.29) sind drei farbige Kreise in einem einzigen Bild zusammengefaßt. Die Motivation, mit möglichst nur *einem* Farbtestbild eine umfassende Beurteilung von Simulationsergebnissen zu ermöglichen, liegt in den langen Simulationszeiten beim Entwurf videosignalverarbeitender Komponenten begründet. Die für die funktionale Simulation der VHDL-Verhaltensberschreibung des digitalen PAL-Encoders *PalCo* erforderliche Rechenzeit liegt in der Größenordnung von einer halben Stunde zur Erzeugung eines PAL-Videosignals mit einer Realzeitdauer von $0,16$ s, das zur Darstellung mit *VidTrans* auf einem realen Farbfernseher geeignet ist. Ein PAL-Halbbild hat nämlich eine Dauer von $0,02s$, was $0,02$ s $\cdot\ 4,43361875$ MHz $= 88723,75$ Perioden des PAL-Farbträgersignals (vgl. Abschnitt 2.4.1, S. 39) entspricht. Acht PAL-Halbbilder mit einer Dauer von $0,16$ s entsprechen $0,16$ s$\cdot 4,43361875$ MHz $= 709790$ Perioden, was bedeutet, daß bei periodischer Darstellung von acht PAL-Halbbildern oder ganzzahligen Vielfachen von acht kein Phasensprung des PAL-Farbträgersignals auftritt.

Jedem Punkt (x, y) innerhalb eines der drei farbigen Kreise (Abb. 4.29) wird ein Wertepaar von Farbdifferenzen (u, v) zugeordnet. Dabei ist

$$\begin{aligned} u(x,y) &= x/R \\ v(x,y) &= y/R, \end{aligned} \tag{4.44}$$

und R ist der Radius des betreffenden Kreises. Der Ursprung $(0, 0)$ der lokalen Kreiskoordinaten liegt im Mittelpunkt des betreffenden Kreises. Bei dem oberen Kreis der Abbildung 4.29 ist die Luminanz

$$y(u, v) = 1 \tag{4.45}$$

konstant gleich eins. Dadurch ergeben sich große Bereiche mit Pastellfarben. Bei dem Kreis unten links ist

$$y(u, v) = \sqrt{u^2 + v^2} \tag{4.46}$$

eine Funktion der Farbdifferenzsignale u und v, wodurch sich ausgehend vom Kreismittelpunkt Farbverläufe mit Farben geringer Luminanz ergeben, bei denen Quantisierungsartefakte besonders deutlich hervortreten. Bei dem Kreis unten rechts ist

$$y(u, v) = 1 - \sqrt{u^2 + v^2}, \tag{4.47}$$

wodurch sich ein Farbkreis ergibt, der mit Darstellungen in der Literatur [Limann, 1978] vergleichbar ist und die Identifizierung von Phasendrehungen des Farbträgersignals erlaubt – eine

Phasenverschiebung des Farbträgersignals äußert sich als Drehung dieses Farbkreises um einen entsprechenden Winkel.

Für die Validierung der Hardware-Implementierung des PAL-Encoders (*PalCo* [Larsson, 1999b]) wurden ebenfalls Farbwerte im YUV-Farbraum als Funktion von lokalen Koordinaten zur Farbtestbildgenerierung verwendet. Dabei ist $u(i)$ eine Funktion der Bildschirmspalte i und $v(j)$ eine Funktion der Bildschirmzeile j, und die Luminanz $y(k)$ ist eine Funktion der Bildnummer k, so daß immer einer UV-Ebene bei konstantem y im YUV-Farbraum dargestellt wird.

Für die Hardware-Implementierung des digitalen PAL-Encoders wurde für das Luminanz-Signal y im Wertebereich $\{0, ..., 1\}$ eine vorzeichenlose Dualzahlendarstellung mit einer Genauigkeit von 8 Bit ($m = 8$) gewählt. Das Luminanz-Signal wird daher gemäß

$$J(x) = \frac{\lfloor 2^m \cdot x \rfloor}{2^m} \tag{4.48}$$

quantisiert. Für die Farbdifferenzsignale u und v, die im Wertebereich $\{-1, \ldots, 1\}$ liegen, wurde eine vorzeichenbehaftete Zahlendarstellung im 2er-Komplement mit einer Genauigkeit von 4 Bit ($n = 4$) gewählt. Die Beträge der Farbdifferenzsignale $|u|$ und $|v|$ werden also gemäß

$$K(x) = \frac{\lfloor (2^{n-1} - 1) \cdot x \rfloor}{2^{n-1} - 1} \tag{4.49}$$

quantisiert. Die Quantisierung der Luminanz y gemäß (4.48) ist weder beim Software-PAL-Encoder noch beim Hardware-PAL-Encoder wahrnehmbar. Bezüglich der Quantisierung der

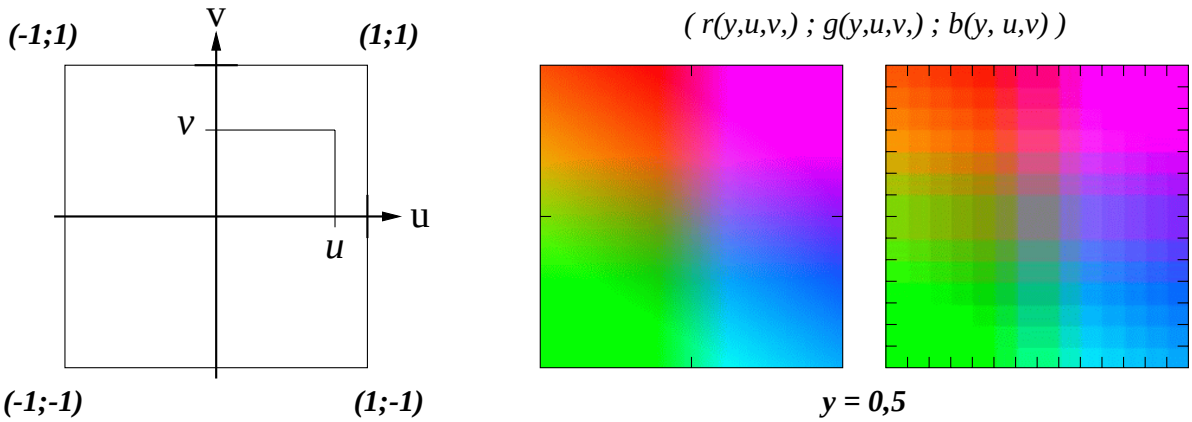


Abbildung 4.30: **Farbtestbilder.** *Beim Quadrat auf der rechten Seite der Abbildung wurden u und v entsprechend der für den FPGA-PAL-Encoder-Prototypen gewählten Zahlendarstellung (4 Bit 2er-Komplement) quantisiert. Die Hardware-Implementierung liefert eine PAL-Videosignal, welches auf einem PAL-Farbfernseher ein ähnliches, aber durch Quantisierungsrauschen des PAL-Farbträgers gestörtes, Bild liefert (s. Abb. 5.10, S. 155). Mit einem Software-PAL-Encoder, der mit genauerer interner Zahlendarstellung arbeitet, ist Quantisierungsrauschen des PAL-Farbträgers dagegen nicht wahrnehmbar (s. Abb. 4.25, S. 119).*

Farbdifferenzsignale u und v sind jedoch deutliche Unterschiede zwischen Software-Implementierung und Hardware-Implementierung digitaler PAL-Encoder bei Betrachtung der Farbtestbilder auf der Systemebene durch Darstellung auf einem Farbfernseher wahrnehmbar.

Der Software-PAL-Encoder liefert ein Bild, bei dem zwar die Quantisierung von u und v gemäß (4.49) deutlich in Form von Rechtecken mit konstantem u und v sichtbar ist. Diese Rechtecke weisen aber eine gleichmäßige Färbung auf. Dagegen liefert der Hardware-PAL-Encoder ein Bild, bei dem einige Rechtecke stark gestört sind. Das liegt zum einen am Quantisierungsrauschen des Farbträgersignals und zum anderen an internen Zahlenbereichsüberschreitungen im digitalen PAL-Videosignal, die sich jedoch, wie der Software-PAL-Encoder zeigt, durch Wahl hinreichender Zahlendarstellungsgenauigkeit vermeiden lassen.

Die Darstellung von Signalen im Zeitraum – als Funktion der Zeit – oder im Frequenzraum – in Form von mit Hilfe der Fourier-Transformation (vgl. Abschnitt 2.3, S. 25) berechneten Spektren – erlaubt zunächst nicht die direkte wahrnehmungsphysiologische Bewertung von Simulationsergebnissen, wie es durch den Austausch von Signalen mit realen Anwendungsumgebungen ermöglicht wird. Später kann jedoch unter Umständen eine Zuordnung zwischen numerischen Größen und qualitativen, wahrnehmungsphysiologischen Parametern gefunden werden, um zu qualitativen Aussagen zu kommen, die eine Zuordnung zwischen messbaren Größen und Qualitätsmaßstäben erlauben. Ein Beispiel dafür ist etwa die Aussage, daß ein mit 8 Bit, was einem Signalrauschabstand von etwa 48 dB entspricht, bei einer Abtastfrequenz von 15 MHz quantisiertes PAL-Farbvideosignal eine Darstellung ermöglicht, bei der Quantisierungsartefakte nicht oder fast nicht wahrnehmbar sind. Um zu solchen Aussagen gelangen zu können, ist jedoch der Austausch zwischen Computersimulationen und realen Anwendungsumgebungen erforderlich.

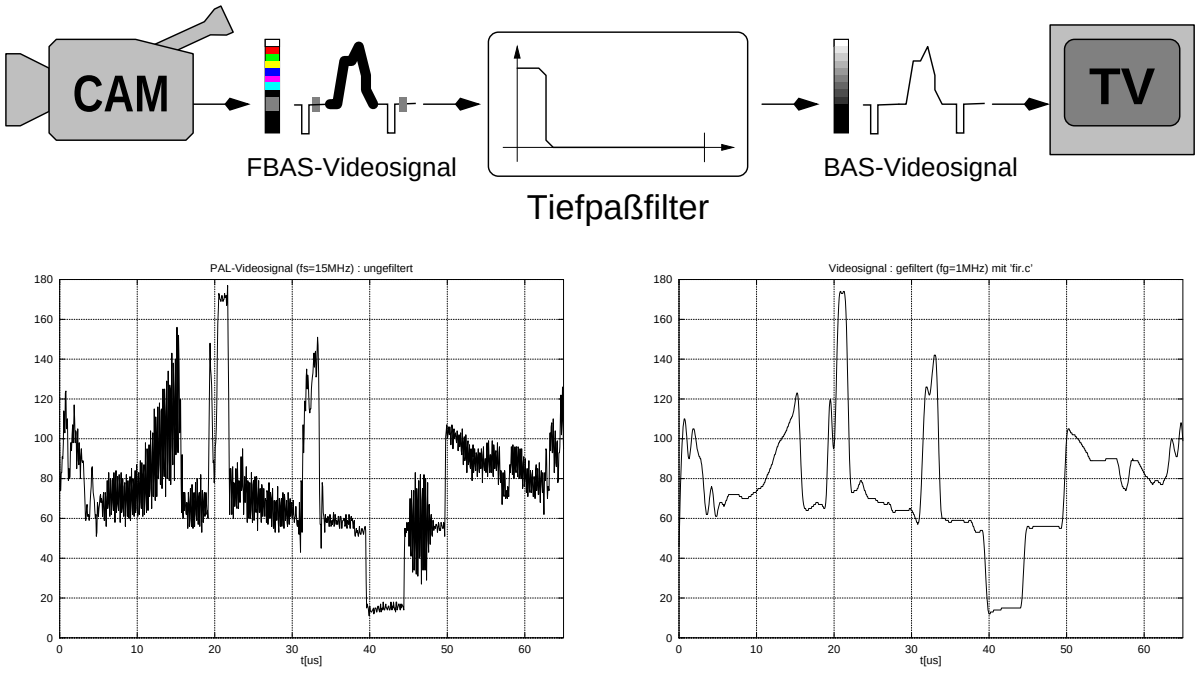


Abbildung 4.31: **FIR-Tiefpaßfilterung von PAL-Videosignalen.** *FBAS nach BAS.*

FIR-Filterung eines PAL-Videosignals

Die Tiefpaßfilterung zur Unterdrückung des PAL-Farbsignals mit einem FIR-Filter dient im folgenden als Anwendungsbeispiel, um Validierung eines Entwurfs durch Beobachtung von Verhalten auf der Systemebene zu demonstrieren und Vorteile aufzuzeigen. Dazu wird eine Software-Implementierung mit einer Hardware-Implementierung eines einfachen FIR-Filters verglichen. Dabei wird die Visualisierung von Simulationsergebnissen auf der Systemebene mit den etablierten Visualisierungsmethoden der graphischen Signaldarstellung und Spektrendarstellung verglichen. Die Testumgebung weist die in Abbildung 4.31 skizzierte Struktur auf.

Der implementierte FIR-Filter filtert das PAL-Farbvideosignal durch Berechnung einer diskreten Faltung (vgl. S. 30) gemäß (2.23) auf Basis von Fließkommazahlendarstellung. Für die Hardware-Implementierung des FIR-Filters wurde die in Abbildung 2.8 (unten) auf Seite 29 dargestellte Architektur gewählt. Dabei erfolgen die Berechnungen auf Basis von vorzeichenbehafteter Ganzzahldarstellung (signed). Die Berechnung der Filterkoeffizienten erfolgte gemäß (2.22) und (2.24) mit einem Fenster der Form (2.25).

Bei diesem Anwendungsbeispiel zeigt sich besonders deutlich, daß die Festlegung von Parametern auf Basis bestehender Verfahren zwar geeignet ist, einen Lösungsansatz für die vorliegende Aufgabe zu finden, daß aber erst die systemnahe Validierung die Eignung eines gewählten Parametersatzes im Systemkontext ermöglicht, da abstrakte Systemparameter das Verhalten der Implementierungen nur unzureichend wiedergeben. Darüber hinaus zeigt sich auch, daß die

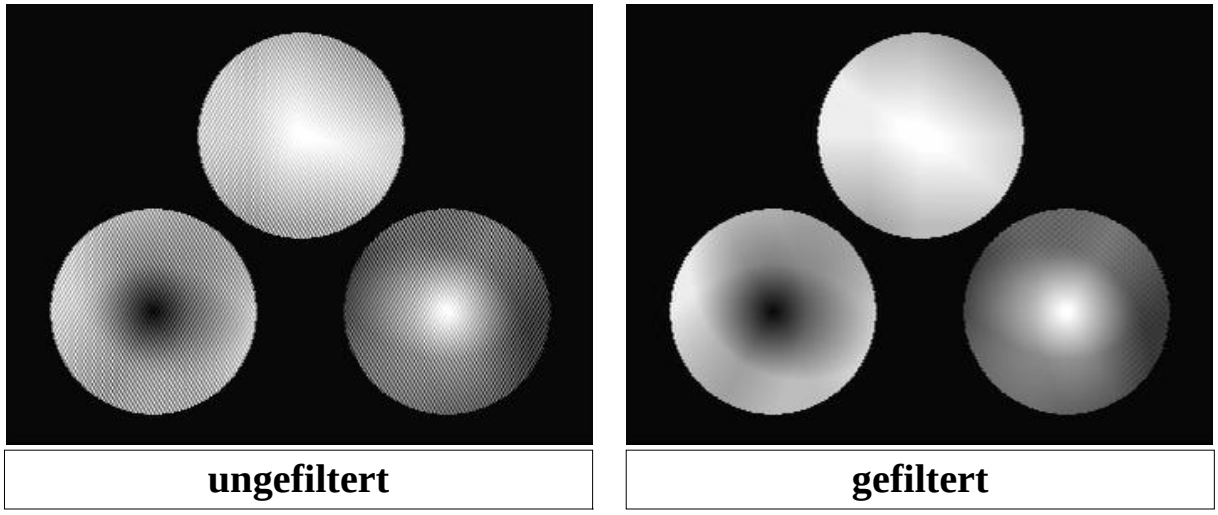
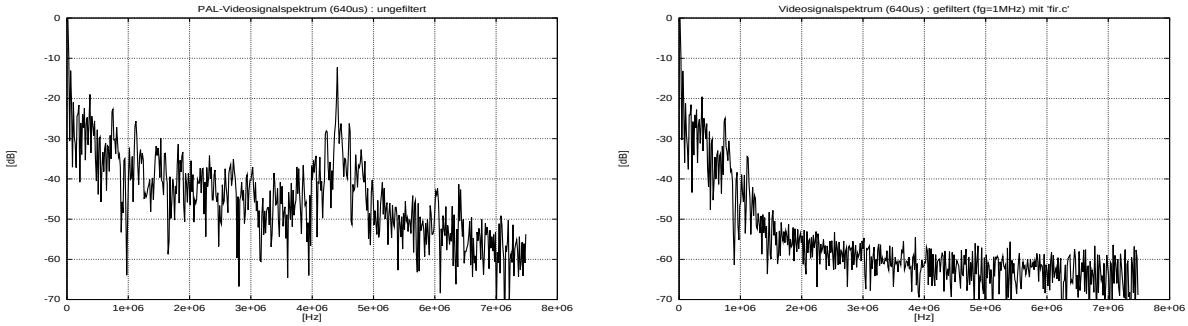


Abbildung 4.32: **Extrahierte Graubilder - ungefiltert vs. gefiltert.** *Beide Bilder wurden aus demselben, mit einem Software-PAL-Encoder generierten, PAL-Farbvideosignaldatenstrom extrahiert. Bei dem Bild auf der linken Seite wurde auf die Unterdrückung des PAL-Chrominanz-Signals durch Tiefpaßfilterung verzichtet. Das PAL-Chrominanz-Signal ist daher in dem Graubild als störendes Muster in den Kreisen wahrnehmbar. Bei dem Bild auf der rechten Seite wurde das PAL-Chrominanz-Signal mit einen FIR-Filter weitgehend unterdrückt. Die Unterschiede sind jedoch durch die für den Ausdruck notwendige Rasterung weniger deutlich wahrnehmbar, als bei der Bildschirmdarstellung mit echten Graustufen.*

systemnahe Validierung durch Beobachtung von Verhalten einer Implementierung auf der Systemebene geeignet ist, einen Lösungsansatz in Bezug auf Robustheit zu untersuchen, und darüber hinaus geeignet ist, mögliche Implementierungsfehler aufzuspüren. Die Verwendung eines FIR-Filters zur Unterdrückung des Chrominanz-Signals im PAL-Videosignal könnte außerdem dazu dienen, ein System zur Verarbeitung von Graubildern von Videosignalquellen dahingehend unabhängig zu machen, daß neben Videosignalquellen, die ein BAS-Signal, also ein Videosignal *ohne* Farbsignalanteil liefern, auch Videosignalquellen, die ein FBAS-Signal mit Farbsignalanteil liefern, gleichermaßen verwendet werden können. Diese Unabhängigkeit ist eher von praktischer Natur, da preiswerte handelsübliche Videosignalquellen meist ein PAL-Farbvideosignal liefern. Wird vor der Verarbeitung von Graubildern der Farbsignalanteil nicht unterdrückt, so tritt der Farbsignalanteil als störendes Muster in extrahierten Graubildern in Erscheinung, wie in Abbildung 4.32 dargestellt.

Beim Vergleich der Beobachtung des Verhaltens und der Wechselwirkung der Software-Implementierung und der Hardware-Implementierung eines FIR-Filters zur Tiefpaßfilterung eines PAL-Farbvideosignals mit realen Systemkomponenten wird besonders deutlich, daß zur Beurteilung der Robustheit von Implementierungen die Beobachtung auf der Systemebene erforderlich ist, wenn Systemkomponenten implementiert werden, die mit anderen realen Systemkomponenten interagieren, für die keine formalen Spezifikationen existieren. Außerdem wird anhand dieses Anwendungsbeispiels deutlich, daß die im Rahmen der vorliegenden Arbeit eingeführten systemnahen Validierungsmethoden Möglichkeiten eröffnen, auch solche Systeme zu entwerfen und vor der endgültigen Implementierung zu validieren, die sich formalen Validierungsmethoden sonst entziehen würden.



PAL-Videospektrum (ungefiltert) Videospektrum (tiefpaßgefiltert)

Abbildung 4.33: **Videosignalspektren** – *ungefiltert (links) und tiefpaßgefiltert (rechts). Den Spektren liegt ein Zeitintervall von* $640\ \mu s$ *zugrunde. Das Videosignal wurde mit* VidTrans *(8-Bit-ADC bei einer Abtastfrequenz von* $f_s = 15\ MHz$*) digitalisiert. Beim Spektrum des tiefpaßgefilterten PAL-Videosignals ist zu erkennen, daß das Farbsignal durch den Tiefpaßfilter bis unter die Schwelle des Quantisierungsrauschens von* 48 *dB*$(\approx 6 \cdot 8\ Bit)$ *unterdrückt ist.*

Parametrierungsansatz

Für die Implementierung des FIR-Filters zur Tiefpaßfilterung von PAL-Farbvideosignalen sind verschiedene Parameter festzulegen. Einige Parameter ergeben sich dabei aus vorab festgelegten, technischen Randbedingungen. Die Auflösung des Analog-Digital-Wandlers (ADC) bestimmt das Quantisierungsrauschen und die erforderliche Signalunterdrückung – und damit die Restwelligkeit im Sperrbereich – des Filters im Sperrbereich. Außerdem bestimmt die Auflösung des Analog-Digital-Wandlers direkt die Wortbreiten von MAC-Einheiten bei der Hardware-Implementierung eines FIR-Filters. Erfolgt die diskrete Faltung bei der Software-Implementierung eines FIR-Filters auf Basis von Fließkommazahlendarstellung, so ist der Dynamikumfang bei derzeit möglichen Auflösungen von Analog-Digital-Wandlern und Digital-Analog-Wandlern in jedem Fall hinreichend. Bei Fließkommadarstellung mit einfacher Genauigkeit (`float`) gemäß (4.30) ergibt sich ein Dynamikumfang von $\log_2(10^{38}) = \log(10^{38})/\log(2) = 38 \cdot \log(2) \approx 126$[dB], und bei doppelter Genauigkeit (`double`) ergibt sich sogar ein Dynamikumfang von $\log_2(10^{308}) = \log(10^{308})/\log(2) = 308 \cdot \log(2) \approx 1023$[dB] für die Zahlendarstellung.

Die Signalunterdrückung im Sperrbereich eines Filters sollte im Bereich des Quantisierungsrauschens liegen. Eine stärkere Signalunterdrückung ist nicht erforderlich und würde nur eine Erhöhung des Implementierungsaufwands zur Folge haben, wie aus den Aufwandsabschätzungen (2.28) und (2.29) zu entnehmen ist. Die relative Übergangsbreite geht ebenfalls in die Aufwandsabschätzung ein. Ein möglichst ausgedehnter Übergangsbereich führt zu einem geringeren Implementierungsaufwand. Die Zahl der Filterkoeffizienten ist dabei ein wichtiges Maß für den Implementierungsaufwand. Bei einer Software-Implementierung erhöht sich die Verarbeitungszeit mit der Zahl der Filterkoeffizienten, bei einer Hardware-Implementierung erhöht sich die Zahl der benötigten MAC-Einheiten.

Wesentlich für die Robustheit in Bezug auf die Parametrierung ist die Form *typischer PAL-Videosignalspektren*. In Abbildung 4.33 sind zwei PAL-Videosignalspektren dargestellt – ungefiltert und gefiltert. Die Spektren wurden aus einem digitalisierten PAL-Videosignaldatenstrom auf Basis der Fourier-Transformation (2.14) durch *diskrete Fourier-Transformation* [Press *et al.*, 1992] errechnet. Wie bei der Abbildung 4.33 dargestellt ist, überwiegen niederfrequente Signalanteile gegenüber höherfrequenten Signalanteilen. Diese Beobachtung gibt Anlaß zu der Annah-

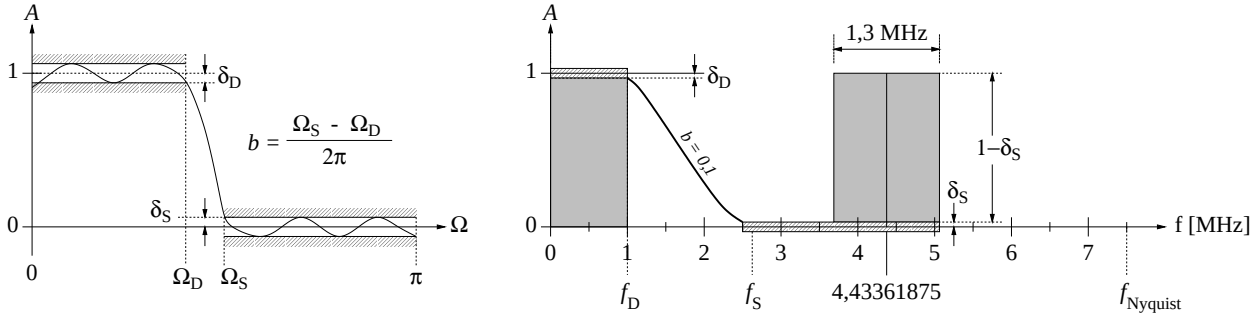


Abbildung 4.34: **FIR-Toleranz und PAL-Chrominanz-Spektrum** *(Schema links nach [Fliege, 1993])*

me, daß der Übergangsbereich des Tiefpaßfilters nur so breit sein muß, daß das PAL-Farbsignal hinreichend stark, d.h. in den Bereich des Quantisierungsrauschens, unterdrückt wird. In der Abbildung 4.34 sind Filtertoleranzschema und ein schematisches PAL-Videosignalspektrum gegenübergestellt.

Für die Zahl der Filterkoeffizienten wurden für dieses Beispiel empirisch 31 Koeffizienten angesetzt. Mit den Aufwandsabschätzungen nach *Bellanger* (2.26) und nach *Kaiser* (2.27) und geeignet umgeformten Ausdrücken (2.28) und (2.29) ergeben sich mit $\delta[\text{dB}] = 48$

$$\begin{aligned} b_{Bellanger} &\approx \frac{\delta[\text{dB}] + 10}{-15 \cdot N} \\ b_{Kaiser} &\approx \frac{\delta[\text{dB}] + 13}{-14,6 \cdot (N-1)} \end{aligned} \tag{4.50}$$

für die relative Übergangsbreite Werte von $b_{Bellanger} = 0,12$ und $b_{Kaiser} = 0,07$, also $b \approx 0,1$. Bei einer Abtastfrequenz $f_s = 15$ MHz ergibt sich bei einer relativen Übergangsbandbreite $b = 0,1$, also eine Übergangsbandbreite von $0,1 \cdot f_s = 1,5$ MHz. Diese festgelegten Randbedingungen für die Filterung sind in Abbildung 4.34 skizziert. Es sei an dieser Stelle noch mal darauf hingewiesen, daß hier nicht der Entwurf von FIR-Filtern im Vordergrund steht, sondern daß der Entwurf eines FIR-Filters zur Tiefpaßfilterung eines PAL-Videosignals hier als Anwendungsbeispiel angeführt wird, um die Problematik bei der Implementierung eines solchen Filters im Kontext einer realen Systemumgebung zu verdeutlichen, um die systemnahe Validierung eines solchen Filters durch Beobachtung seines Verhaltens im Wechselspiel mit realen Systemkomponenten zu demonstrieren und um die Unterschiede zwischen Software-Implementierung und Hardware-Implementierung anhand eines einfachen Beispiels praktisch aufzuzeigen.

Vergleich von Implementierungen

Der Vergleich der Bewertung von Simulationsergebnissen eines FIR-Filters anhand von Fourier-Spektren (s. Abschnitt 2.3, S. 25) mit der Bewertung von Simulationsergebnissen durch direkte Darstellung auf analogen Farbfernsehgeräten mit Hilfe von *VidTrans* zeigt die Vorteile der im Rahmen der vorliegenden Arbeit entwickelten Methoden zur Validierung durch Beobachtung von Verhalten auf der Systemebene. Im folgenden werden dazu funktional ähnliche Implementierungen des FIR-Filters zur Tiefpaßfilterung eines PAL-Farbvideosignals, die sich bezüglich gemachter Implementierungsvereinfachungen unterscheiden, anhand von Signalverläufen, Spektren, mit einem Software-PAL-Decoder extrahierten Bildern, sowie anhand von direkten Bilddarstellungen auf analogen Farbfernsehgeräten, verglichen.

In Abbildung 4.35 sind die Frequenzgänge des auf unterschiedlichen Plattformen implementierten FIR-Filters zur Tiefpaßfilterung des PAL-Farbvideosignals dargestellt. Die Spektren wurden durch inverse Fourier-Transformation (vgl. Abschnitt 2.3, S. 25) der Impulsantwort auf einen Delta-Impuls – genau ein Abtastwert mit maximal möglicher Amplitude ($2^8 - 1 = 255$) für die Dauer $t_s = 1/f_s$ genau einer Abtastung – gewonnen. Da sowohl für das Eingangssignal des FIR-Filters wie auch für das Ausgangssignal des FIR-Filters vorzeichenlose, ganzzahlige Darstellung gewählt wurde, werden mögliche positive Überläufe, aber auch mögliche negative Unterläufe

begrenzt. Diese Entwurfsentscheidung führt zu einer Veränderung des Frequenzgangs des FIR-Filters, obwohl filter-intern auf Basis vorzeichenbehafteter Zahlendarstellung gerechnet wird. Ein Delta-Impuls am Eingang des FIR-Filters sollte eigentlich die Filterkoeffizienten am Ausgang des FIR-Filters sequentiell erscheinen lassen – negative Filterkoeffizienten werden jedoch am Ausgang des FIR-Filters auf null limitiert.

Die Software-Implementierung des FIR-Filters, welche die Berechnungen mit Fließkommazahlendarstellung durchführt (`(float)`), zeigt einen Frequenzgang, der die gewünschte Unterdrückung des Farbsignalanteils des PAL-Videosignals vermuten läßt. Die Software-Implementierung des FIR-Filters (`fir.c`) und die funktional ähnliche Hardware-Implementierung des FIR-Filters (`fir.vhd`) zeigen bezüglich des Frequenzgangs auch ähnliches Verhalten. Die in Abschnitt 4.2.4, S. 99 beschriebene logarithmische Quantisierung (4.32) der Filterkoeffizienten ermöglicht zwar eine Multiplikation durch eine einfache Schiebeoperation zu realisieren, was aber zu einer erheblichen Veränderung des Frequenzgangs des FIR-Filters führt, wie in Abbildung 4.35 – `bslfir.vhd` – deutlich zu sehen ist. Insbesondere ist die Unterdrückung des PAL-Farbsignalanteils im Sperrbereich des Filters dabei mit ≈ 25 $[dB]$ weit über den angestrebten 48 $[dB]$. Die Frage, wie die unterschiedlichen Frequenzgänge der verschiedenen Implementierungsalternativen sich auf der Systemebene, d.h. bei der Darstellung des gefilterten PAL-Videosignals auswirken, kann durch Bewertung der Fourier-Spektren zunächst nicht abschließend geklärt werden.

In Abbildung 4.36 sind die gefilterten Signale und Spektren der Software-Implementierung

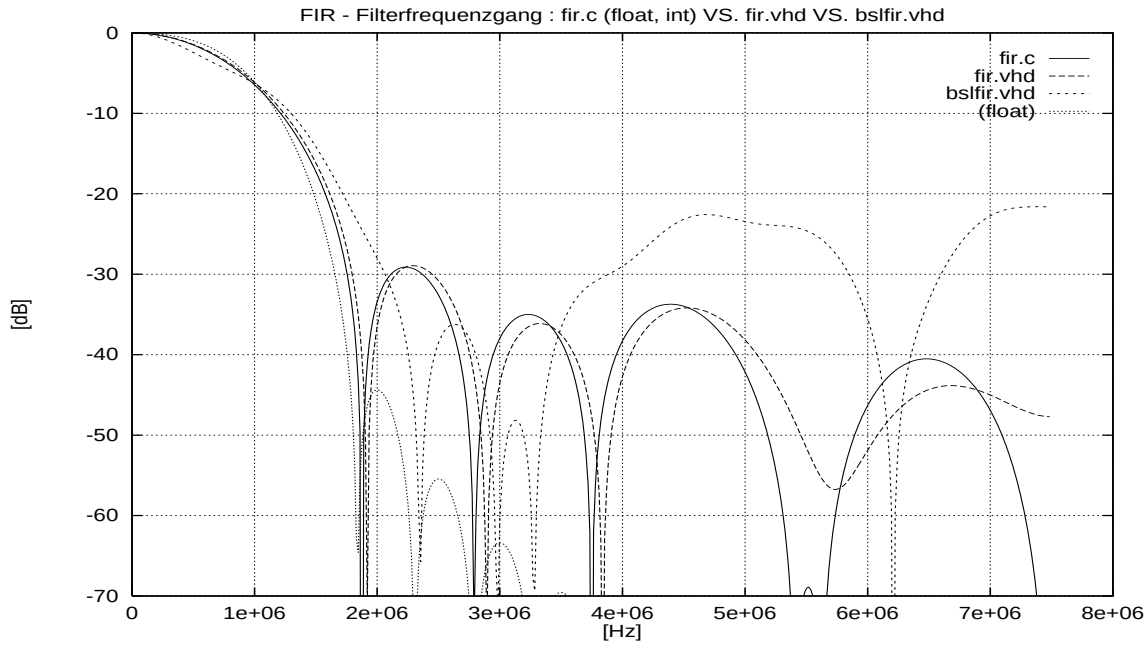


Abbildung 4.35: **FIR-Filterspektrum** *auf unterschiedlichen Implementierungsplattformen für eine festgelegte Grenzfrequenz* $f_g \approx 1\ MHz$.

des FIR-Filters (`fir.c`) und der Hardware-Implementierung des FIR-Filters (`fir.vhd`) dargestellt. Bei der Signaldarstellung und bei der Darstellung des Spektrums der Hardware-Implementierung des FIR-Filters (`fir.vhd`) ist zu erkennen, daß der Farbsignalanteil des tiefpaßgefilterten PAL-Videosignals nicht vollständig unterdrückt wird. Bei der Signaldarstellung ist das fast nicht wahrnehmbare Farbsignal bei genauem Hinsehen zu erkennen – besser ist der Farbsignalanteil im Fourier-Spektrum auszumachen. Aber auch hier stellt sich die Frage, wie sich dieser Signalanteil bei der Darstellung auf einem PAL-Farbfernseher auswirkt. Die Antwort auf diese Fragen liefert eben gerade die Darstellung auf realen PAL-Farbfernsehgeräten. Ist das Farbsignal vollständig unterdrückt, so sind bei der Extraktion mit dem Software-PAL-Encoder einige farbige Bildpunkte auszumachen, die aber fast nicht auffallen. Ganz anders ist das bei der Darstellung auf einem PAL-Farbfernsehgerät mit Hilfe von *VidTrans*. Dabei wird das Farbsignal im analogen Farbfernseher derart verstärkt, daß nach einigen Sekunden eine farbige Darstellung des tiefpaßgefilterten PAL-Videosignals zu sehen ist, die vermuten ließe, daß das PAL-Videosignal überhaupt nicht tiefpaßgefiltert wurde, wodurch die betreffenden Beispielentwürfe auf der Systemebene nicht-formal falsifiziert werden können.

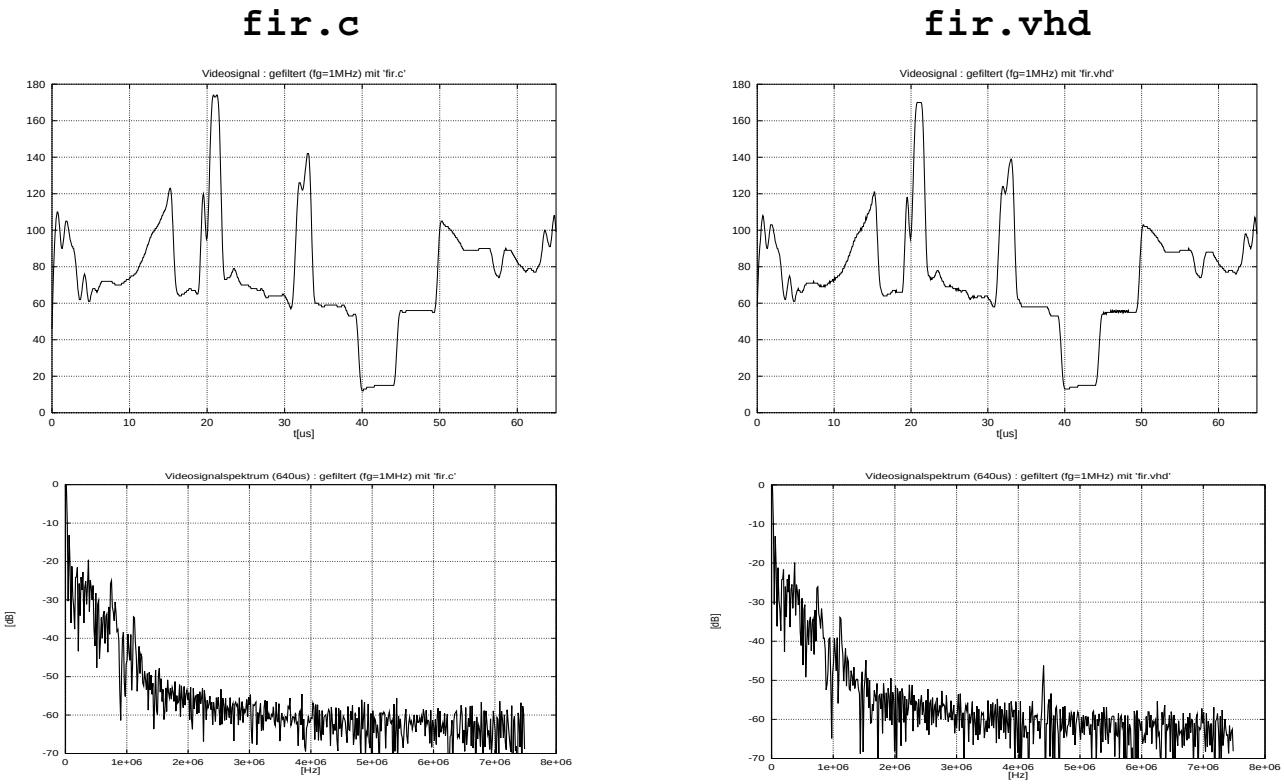


Abbildung 4.36: **Spektren – C vs. VHDL** *Software- vs. Hardware-Implementierung für* $f_g \approx 1\,MHz$.

4.3.3 Schlußfolgerungen

Der Vergleich von etablierten Methoden zur Bewertung von Simulationsergebnissen, als da sind Impulsdiagramme, Signalverläufe und Spektren, mit solchen zur Bewertung von Simulationsergebnissen auf der Systemebene des Zielsystems zeigt deutlich die Überlegenheit beim Entwurf von Systemkomponenten, die im Zielsystem mit realen Systemkomponenten wechselwirken, für die keine formalen Spezifikationen verfügbar sind. Der direkte Austausch von Simulationsergebnissen ist auch für die Unterstützung bei der Parametrierung von Komponenten geeignet, da sich Auswirkungen von der Festlegung von Parametern schon am Anfang eines Entwurfsprozesses evaluieren lassen. Dadurch kann mit Verhaltensmodellen höherer Abstraktionsebenen gearbeitet werden, die bezüglich der erforderlichen Ressourcen – Rechenzeit und Speicherplatz – deutlich günstiger sind als funktional ähnlich Detailmodelle – etwa Schaltungsmodelle auf Gatterebene – eines Entwurfs.

Etablierte Verfahren, die Zusammenhänge zwischen Systemparametern ausdrücken, sind nur dann von Nutzen, wenn die Semantik der betreffenden Systemparameter bekannt ist. Die Semantik von Systemparametern – wie beispielsweise Fehlerquadrat oder Signalrauschverhältnis – ist jedoch bei Systemen, die mit biologischen neuronalen Netzwerken in einer Weise interagieren,

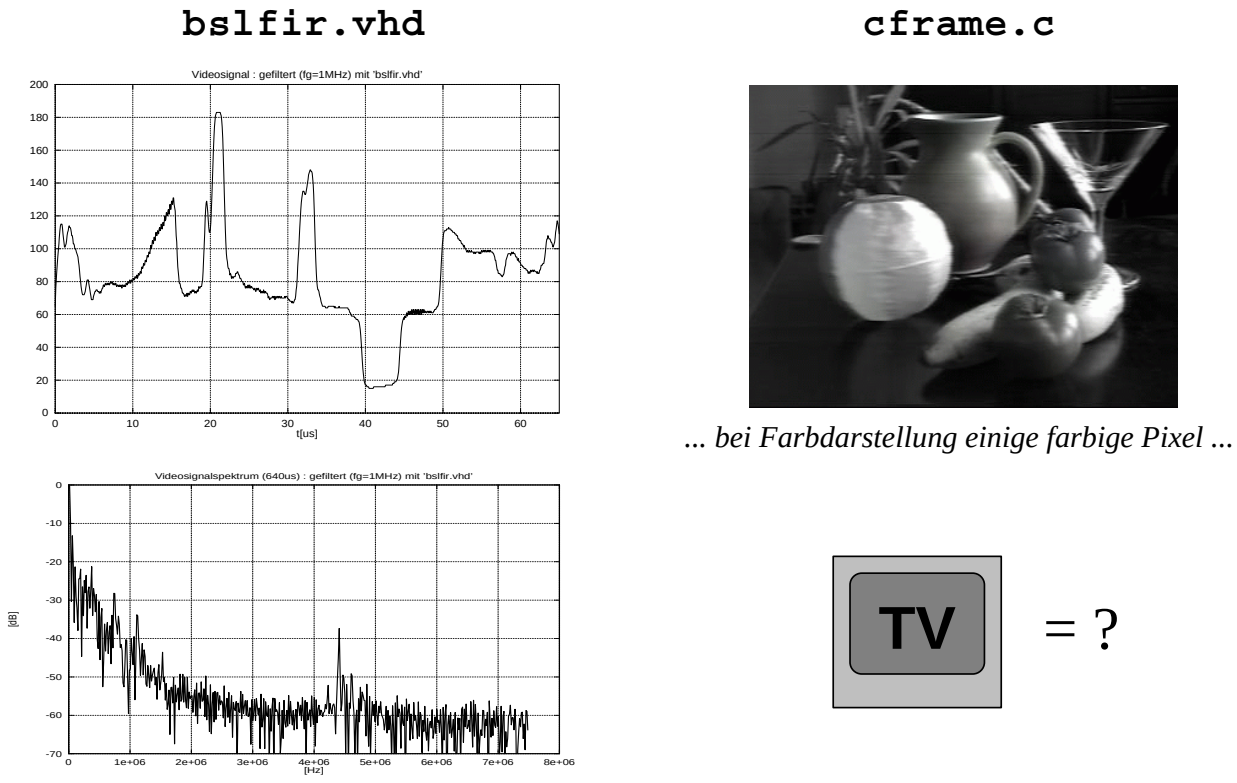


Abbildung 4.37: **Bewertung anhand von Signal, Spektrum, Bild und Realsystem** *Hardware-Implementierung mit (zu) starker numerischer Vereinfachung (Barrel-Shifter, statt Multiplizierer).*

bei der wahrnehmungsphysiologische Randbedingungen von Bedeutung sind, durch Austausch von Simulationsdaten bis hin zur Wahrnehmung zu evaluieren. Mit *VidTrans* konnten digitale PAL-Encoder und digitale PAL-Decoder realisiert und bewertet werden.

4.3.4 Ausblick

Der direkte Austausch von Signalen zwischen Simulation und realen Systemen durch eine geeignete Schnittstelle kann als Grundlage für weitergehende Entwicklungen dienen. Dabei ist auch vorstellbar, schrittweise generische Modelle von realen Systemkomponenten zu entwickeln und derart zu verfeinern, daß damit Simulationen in einer in sich geschlossenen Simulationsumgebung möglich werden.

4.4 Zusammenfassung

Mit der im Rahmen der vorliegenden Arbeit eingeführten Methode der *Lerntrajektorien* und der PLD-basierten bidirektionalen Videosignalschnittstelle *VidTrans* wurden neue Methoden für den Entwurf digitaler Systeme geschaffen, die in komplexen Wechselwirkungen mit realen Anwendungsumgebungen stehen, wie sie zunehmend als Ganzes auf einzelnen Chips integriert werden – System-on-a-Chip (SoC). Zusammen mit den im Rahmen der Arbeit implementierten Verhaltensmodellen des digitalen PAL-Farbvideosignal-Encoders und des digitalen PAL-Farbvideosignal-Decoders sind wissenschaftliche Grundlagen zur Entwicklung digitaler Systeme zur Echtzeitverarbeitung von Videobildern mit neuronalen Netzwerken gelegt. Darüber hinaus sind die gewonnenen wissenschaftlichen Erkenntnisse und Erfahrungen zur digitalen Verarbeitung analoger Videosignale möglicherweise auch für industrielle Entwicklungen im Bereich multimedialer technischer Systeme von praktischem Nutzen.

Kapitel 5

Implementierungen

5.1 Backpropagation-Netzwerk

Der Backpropagation-Algorithmus wurde zwecks Evaluation zunächst als Software-Implementierung in der Sprache C realisiert. Zur Implementierung des Backpropagation-Algorithmus wurden Datenstrukturen und Funktionen definiert, welche eine an die in Kapitel 2, Abschnitt 2.2, S. 13 dargestellte Matrixschreibweise angelehnte Formulierung des Backpropagation-Algorithmus erlaubt. Eingabevektoren, Ausgabevektoren und synaptische Gewichte werden in Gleitkommadarstellung in ASCII-Dateien gespeichert, die sich mit einem Texteditor bearbeiten lassen. Das Programm beinhaltet Module zum direkten Einlesen von Grauwertbildern in einem eigenen rudimentären Dateiformat und im PPM-Format (portable pixmap file format, Jef Poskanzer, 1989). Die Reaktion von Backpropagation-Netzwerken mit zwei Eingängen und einem Ausgang (2-n-1-Netzwerke) kann während des Lernens in einem Fenster beobachtet werden.

5.1.1 Lerntrajektorien

Die Erzeugung von Lerntrajektorien ist, wie in Kapitel 4, Abschnitt 4.2, S. 4.2 beschrieben, in das Programm integriert. Lerntrajektorien werden jedoch nicht direkt graphisch von dem Programm ausgegeben, sondern in eine Datei geschrieben, die mit Hilfe des Plot-Programms *GNU-Plot* [Williams *et al.*, 1993] dargestellt werden kann.

5.1.2 SIMD-Implementierung

Im Rahmen des *"MasPar Challenge Project'94"* wurde die bestehende C-Implementierung des Backpropagation-Algorithmus auf hochparallele (*Single Instruction Multiple Data* (SIMD) [Hennessy und Patterson, 1996]) MasPar-Rechner (MP-1216 und MP-2216) unter Verwendung der Sprache MPL (*MasPar Parallel Application Language*) [MasPar, 1993b,a, 1991] portiert [Larsson, 1994]. Diese Portierung hat aber – sogar unter Vernachlässigung des IO-Flaschenhalses – nur etwa die Performanz der NeNEB-Implementierung [Krol, 1996] erreicht. Für das Architekturkonzept von *NeNEB* (**Ne**uronales **N**etzwerk zur **E**chtzeitklassifizierung von **B**ildern) wurden

technische Randbedingungen in geeigneter Weise berücksichtigt, die in [Larsson *et al.*, 1996, 1997a,b] publiziert sind.

5.2 NeNEB

Die Möglichkeit, ein System zur Echtzeitverarbeitung von Videobildern als digitales System-on-a-Chip zu realisieren, ist aus mehreren Gründen interessant. Die Implementierung digitaler synchroner Schaltungen bietet den Vorteil der hohen Reproduzierbarkeit, da die Informationsverarbeitung digitaler synchroner Schaltungen von elektrotechnischen Parametern unabhängig ist. Darüber hinaus gibt es sehr viele verschiedene Möglichkeiten, digitale Schaltungen in Form von integrierten Schaltungen zu realisieren, insbesondere stehen programmierbare Logikschaltungen grundsätzlich zur Verfügung, die derzeit in den Bereich von 10^6 Gatteräquivalenten vordringen [Altera Corporation, 1998a; Xilinx, Inc., 1999] und so in Zukunft sogar die Realisierung auf einem einzigen FPGA ermöglichen sollten.

Flaschenhalsvermeidung

Die Implementierung eines bildverarbeitenden Systems als System-on-a-Chip erlaubt, den Flaschenhals von Bilddatenübertragung und Bilddatenspeicherung zu vermeiden oder mindestens stark zu reduzieren. So liefert ein 8-Bit-ADC bei einer Abtastfrequenz von $15\ MHz$ – wie im Rahmen der vorliegenden Arbeit gewählt – innerhalb von einer Minute fast ein Gigabyte Daten, die zu übertragen und unter Umständen zu speichern sind. Können jedoch Daten direkt an der Quelle ihres Entstehens verarbeitet werden, so kann dieser Flaschenhals vermieden werden. Das

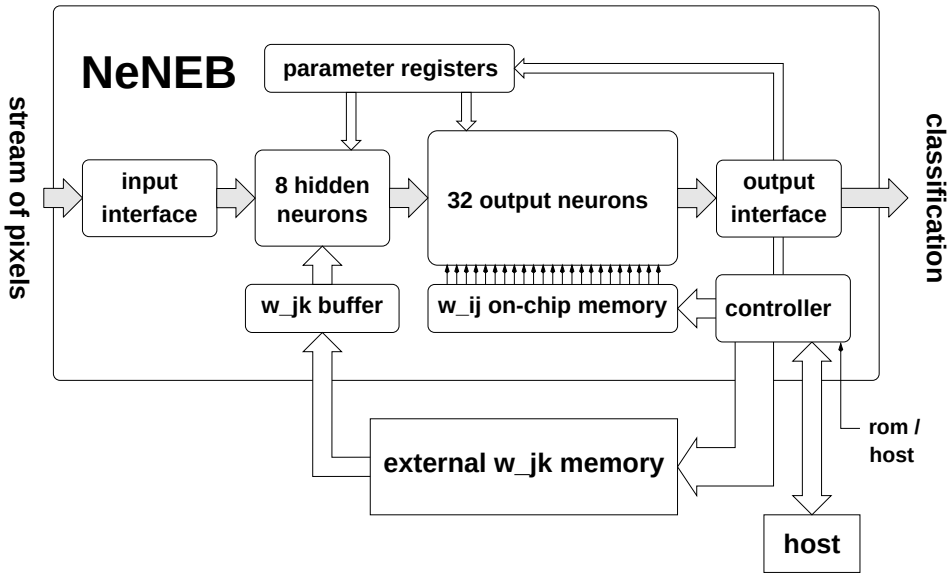


Abbildung 5.1: **NeNEB-Chip-Blockdiagramm** *[Larsson* et al.*, 1997a]*

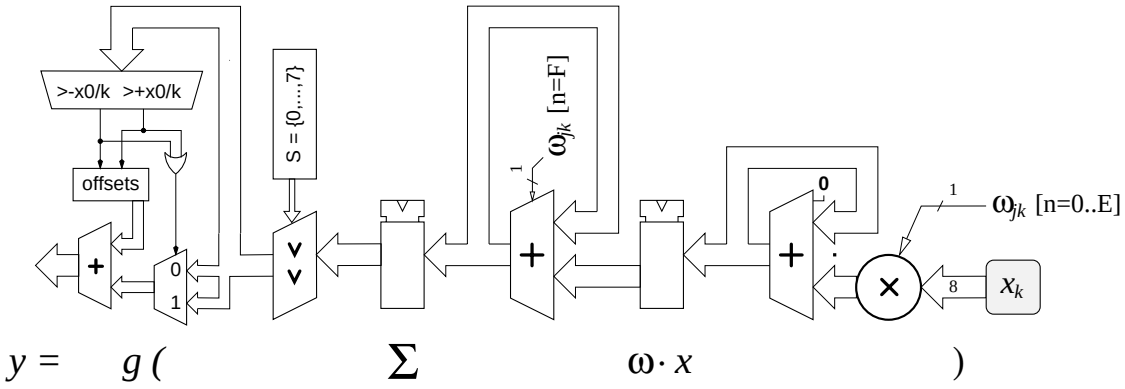


Abbildung 5.2: **Implementierungsschema des Neurons** *[Larsson* et al., *1997a]*

setzt jedoch unter Umständen sehr viel Rechenkapazität voraus, die dann nur mit anwendungsspezifischer Hardware-Implementierung erreichbar ist.

Die externen Anschlüsse einer integrierten Schaltung, über die die Verbindung mit anderen Systemkomponenten hergestellt wird, stellen auch einen möglichen Flaschenhals dar. Daher wurde für die Speicherung der synaptischen Gewichte der verdeckten Neuronen-Schicht das in Abbildung 5.3 skizzierte Speicherschema gewählt.

Serielles Verarbeitungsschema für Videobilder

Viele neuronale Netzwerke sind sehr gut zur Implementierung auf parallelen Architekturen geeignet, da sich künstliche neuronale Netzwerke in Anlehnung an ihre biologischen Vorbilder aus

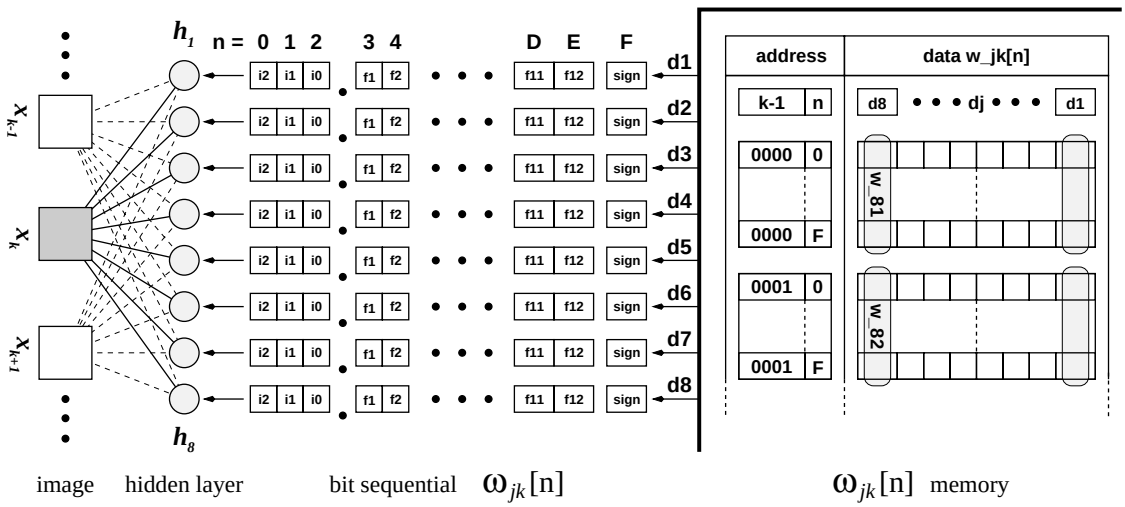


Abbildung 5.3: **Speicherschema der synaptischen Gewichte** *der Hidden-Neuronen [Larsson* et al., *1997a].*

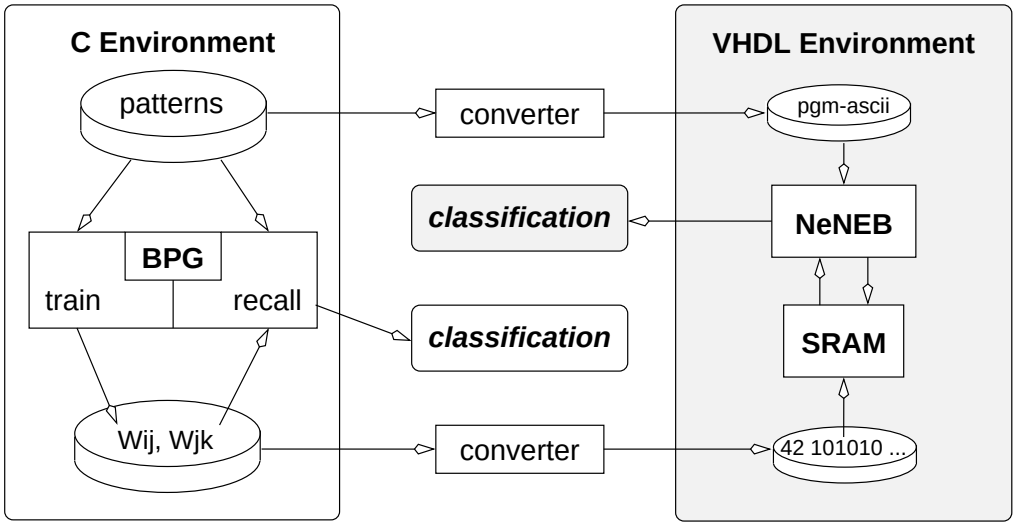


Abbildung 5.4: **Validierung von NeNEB** *[Larsson* et al.*, 1997a].*

vielen parallel arbeitenden Neuronen zusammensetzen. Das Backpropagation-Netzwerk und der Backpropagation-Algorithmus sind Beispiele dafür. Wenn zu verarbeitende Daten jedoch in sequentieller Form vorliegen oder der Zugriff auf gespeicherte Daten nur sequentiell erfolgen kann, dann bietet eine Parallelimplementierung allenfalls geringe Vorteile. Bei biologischen Systemen, wie dem menschlichen Auge, erfolgen die Bildübertragung und auch die Bildverarbeitung hochparallel. Im Gegensatz dazu liefern handelsübliche preiswerte Videokameras einen sequentiellen Video-Datenstrom, da es technisch derzeit sehr aufwendig oder gar unmöglich ist, parallel bildverarbeitenden Einheiten hunderttausende oder gar Millionen von Bildpunkten zuzuführen; es sei denn, eine Parallelverarbeitung erfolgt direkt auf einem Bildsensor selbst.

Bildvorverarbeitung

Zur Schaltungsimplementierung von Bildvorverarbeitungsalgorithmen sind durch den digitalen Schaltungsentwurf implizierte Randbedingungen zu berücksichtigen. Besonders geeignet sind Verfahren, die bezüglich der zu verarbeitenden Bildpunkte einen lokalen Charakter aufweisen, da dann unter Umständen auf die Zwischenspeicherung eines oder mehrerer Bilder verzichtet werden kann. Erfolgt die Vorverarbeitung direkt auf einem bildpunktsequentiellen Videodatenstrom, so ergibt sich aus der vorgegebenen Bildpunktreihenfolge eine weitere wichtige Randbedingung. Darüber hinaus sind solche Vorverarbeitungsoperationen besonders gut geeignet, die durch Veränderung von nur wenigen Parametern die Adaption der notwendigen Verarbeitungsgeschwindigkeit in einem weiten Bereich ermöglichen. Diese Flexibilität ermöglicht unter Umständen auch den Einsatz solcher Chips, die nicht die von entsprechenden Simulationen abgeschätzte Verarbeitungsgeschwindigkeit erreichen. Die Gefahr, daß ein Chip nicht die von der Simulation abgeschätzte Verarbeitungsgeschwindigkeit erreicht, besteht insbesondere bei anwendungsspezifischen Standardzellentwürfen und ganz besonders bei Prototypen.

Modularität

Das System zur Verarbeitung von Videobildern in Echtzeit mit neuronalen Netzwerken wurde derart modular konzipiert, daß Systemkomponenten zu einem hohen Grad unabhängig voneinander entwickelt und untersucht werden konnten und auf bestehenden Komponenten aufgebaut werden konnten, die für eine Systemintegration unter Umständen teilweise anwendungsspezifische Entwicklungen erfordern. Eine wichtige Voraussetzung dafür war die frühzeitige Definition geeigneter Schnittstellen, die eine hinreichend hohe Flexibilität für den Austausch einzelner Systemkomponenten aufweisen.

So wurde die Extraktion von Bildern aus einem digitalisierten Videosignaldatenstrom getrennt von der Klassifizierung von Bildern mit einem Backpropagation-Netzwerk untersucht, da die Realisierung geeigneter digitaler Komponenten in Bezug auf die besondere Entwurfsmethodik, die für einen erfolgreichen Entwurf derartiger rein digitaler Systemkomponenten erforderlich ist, von wissenschaftlichem Interesse ist. Die wählbare Reduzierung der Bildauflösung wurde als eigenständiges Modul realisiert. Dazu wurde ein Verarbeitungsschema gewählt, welches mit einfachen Rechenoperationen realisierbar ist und darüber hinaus einen lokalen Charakter aufweist, so daß nur die Zwischenspeicherung vom Umfang einer Bildzeile für eine Reduktionsstufe erforderlich ist [Krol, 1996]. Die Schnittstellendefinition für das neuronale Netz zur Klassifizierung von Bildern wurde zwar primär in Hinblick auf die Klassifizierung von Bildern, die bildpunktsequentiell als Bildpunktvektoren verarbeitet werden, gewählt, jedoch in einer Weise, die für andere Anwendungen offen ist.

Validierung

Bei *NeNEB* wurde zunächst nur die Erkennungsphase (vgl. Abbildung 2.4, S. 18) implementiert. Der Vergleich zwischen Software-Implementierung (s. Abschnitt 5.1) und der Hardware-Implementierung erfolgte nach dem in Abbildung 5.4 skizzierten Schema.

5.3 VidTrans

Als erster Test wurde das vom ADC digitalisierte Videosignal zunächst direkt zum DAC durchgeschleift und auf einem Farbfernseher dargestellt, um zu evaluieren, wie die mit 8-Bit-Quantisierung zu erreichende Bildqualität ist und ob der Farbsignalanteil dabei noch hinreichend gut übertragen wird. Damit wurde festgestellt, daß bei 8-Bit-Quantisierung eine Bildqualität zu erreichen ist, die von der direkten Videosignalübertragung – durch ein Koaxialkabel – kaum zu unterscheiden ist.

Ein PAL-Videosignal ist auf eine Bandbreite von 5 MHz festgelegt [Kirsch, 1993; Mäusel, 1995; Jack, 1996]. Nach dem Abtasttheorem muß die Abtastfrequenz daher also mindestens 10 MHz betragen. Bei einer Abtastfrequenz von 15 MHz werden innerhalb des sichtbaren Bereichs einer Zeile so viele Bildpunkte abgetastet, daß das Verhältnis von abgetasteten Bildpunkten zur Zeilenzahl eines Bildes etwa dem festgelegten Seitenverhältnis von 8:3 entspricht, so daß eine Verzerrung von dargestellten Objekten bezüglich ihres Höhe-Breite-Verhältnises vermieden wird, wenn zwei aufeinander folgende Bildpunkte zusammengefaßt werden.

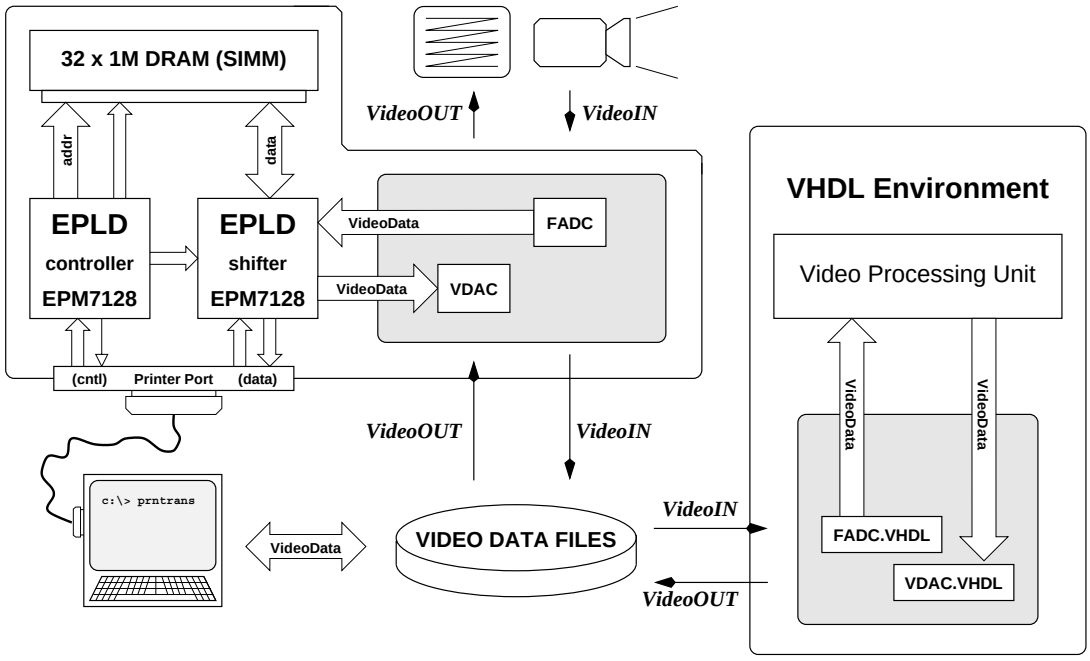


Abbildung 5.5: **VidTrans - Schema** *[Larsson, 1996].*

Das Bildseitenverhältnis ist beim PAL-Fernsehstandard auf 4:3 festgelegt und ein Vollbild (= zwei Halbbilder) setzt sich aus 625 ($= 2 \cdot 312,5$) Zeilen zusammen. Die Zeilenfrequenz beträgt 15.625 Hz. Um einen Phasensprung des Farbträgersignals bei zyklischer Wiedergabe einer Bildsequenz zu vermeiden, sind genau 8 Halbbilder als Sequenz zu speichern. Eine Sequenz von 8 aufeinander folgenden PAL-Halbbildern wird in der Literatur auch als PAL-8er-Sequenz bezeichnet [Morgenstern, 1994]. Die Frequenz des PAL-Farbträgersignals ist $f_{PAL} = 4,43361875$ MHz und die Halbbildfrequenz beträgt 50 Hz. Ein Halbbild dauert demnach $0,02$ s. Um einen Phasensprung des Farbträgers zu vermeiden, muß die Periodenlänge zyklisch angezeigter Halbbilder derart gewählt werden, daß diese einer ganzzahligen Periodenzahl des Farbträgersignals entspricht. Dies ist gerade bei ganzzahligen Vielfachen von 8 Halbbildern = $0,16$ s $\Leftrightarrow 709379$ Perioden des $4,43361875$ MHz Farbträgersignals der Fall. Ein kleiner Phasensprung des Farbträgers führt beim periodischen Abspielen auf einem Farbfernseher zu deutlich sichtbaren – mit der Abspielperiode auftretenden – Farbstörungen. Das ist insofern bemerkenswert, wenn man bedenkt, daß bei einer Abtastfrequenz von 15 MHz während einer Periode des Farbträgersignals etwa 3 Werte abgetastet werden, wogegen innerhalb von $0,16$ s insgesamt 2.400.000 Werte abgetastet werden. Nur ein Abtastwert (± 66 ns) mehr oder weniger ($\approx \pm\frac{2\pi}{3}$ Phasensprung) führt zu einer deutlich sichtbaren Störung auf einem Farbfernseher, was mit *VidTrans* gesehen werden konnte. Das Auffinden eines derartigen Fehlers scheint nur durch derartige systemnahe Simulationsmethoden überhaupt erst möglich, auch wenn das Problem prinzipiell einfach zu durchschauen ist - nur der Umfang der Störung auf der Systemebene ist überhaupt nicht abschätzbar.

Komponenten

VidTrans ist mit zwei EPLDs (Altera EPM7128LC84 [Altera Corporation, 1998c]) realisiert, wie in Abbildung 5.6 dargestellt. Das Controller-EPLD (Abbildung 5.7) hat – zusätzlich zu weiteren Aufgaben – die Funktion eines Steuerwerks, und das Shifter-EPLD (Abbildung 5.8) hat die Funktion eines Operationswerks. Teil des Controller-EPLDs ist ein DRAM-Controller, der die Schnittstelle zum dynamischen Speicher bildet. Die Steuerung sowie das Speichern und das Auslesen von digitalisierten Videosignalen erfolgt über einen externen Rechner, der über die Druckerschnittstelle mit *VidTrans* kommuniziert. Die Kommunikation von *VidTrans* mit dem externen Rechner wird vom Controller-EPLD gesteuert.

Der Video Shifter speichert vier aufeinander folgende 8 Bit breite Abtastwerte des digitalisierten Videosignals und stellt diese als 32 Bit breites Datenwort für Speicherzugriffe zur Verfügung. Dadurch muß nur mit jedem vierten Abtastwert ein Speicherzugriff auf den relativ langsamen DRAM-Videosignalspeicher erfolgen. DRAM-Speicher wurden für die Realisierung von *VidTrans* gewählt, um den Verdrahtungsaufwand etwas zu reduzieren, insbesondere im Hinblick auf die beschränkte Zahl von Anschlüssen der verwendeten EPLDs. *VidTrans* arbeitet mit einer Taktfrequenz von 30 MHz, die Abtastfrequenz für das Videosignal beträgt 15 MHz.

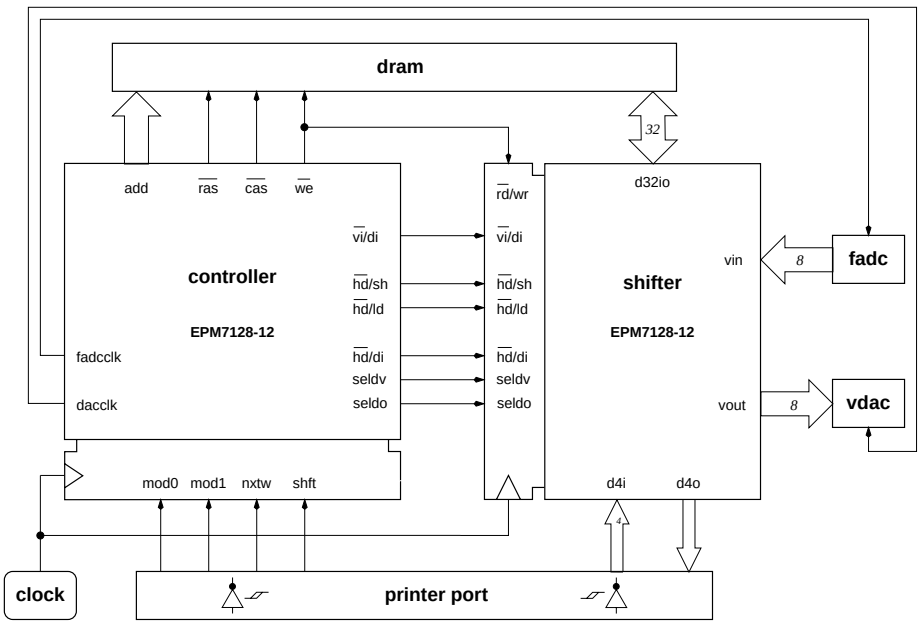


Abbildung 5.6: **VidTrans - Controller und Shifter** *[Larsson, 1996].*

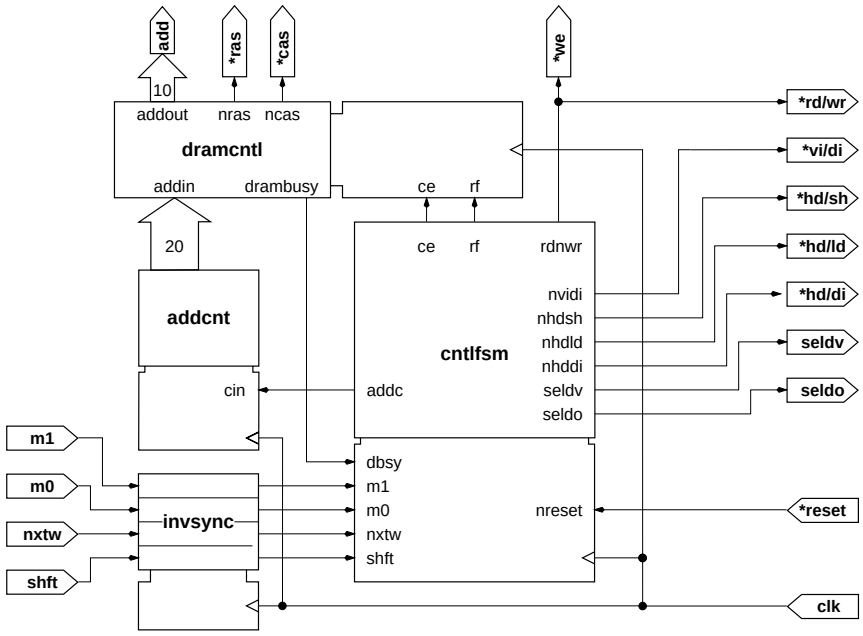


Abbildung 5.7: **VidTrans - Controller** *[Larsson, 1996]. Der Controller steuert den Video Shifter.*

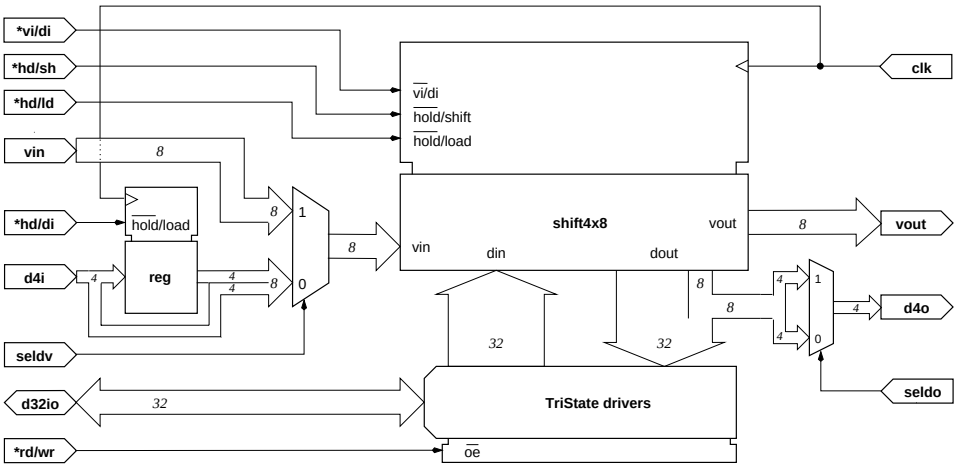


Abbildung 5.8: **VidTrans - Video Shifter** *[Larsson, 1996].*

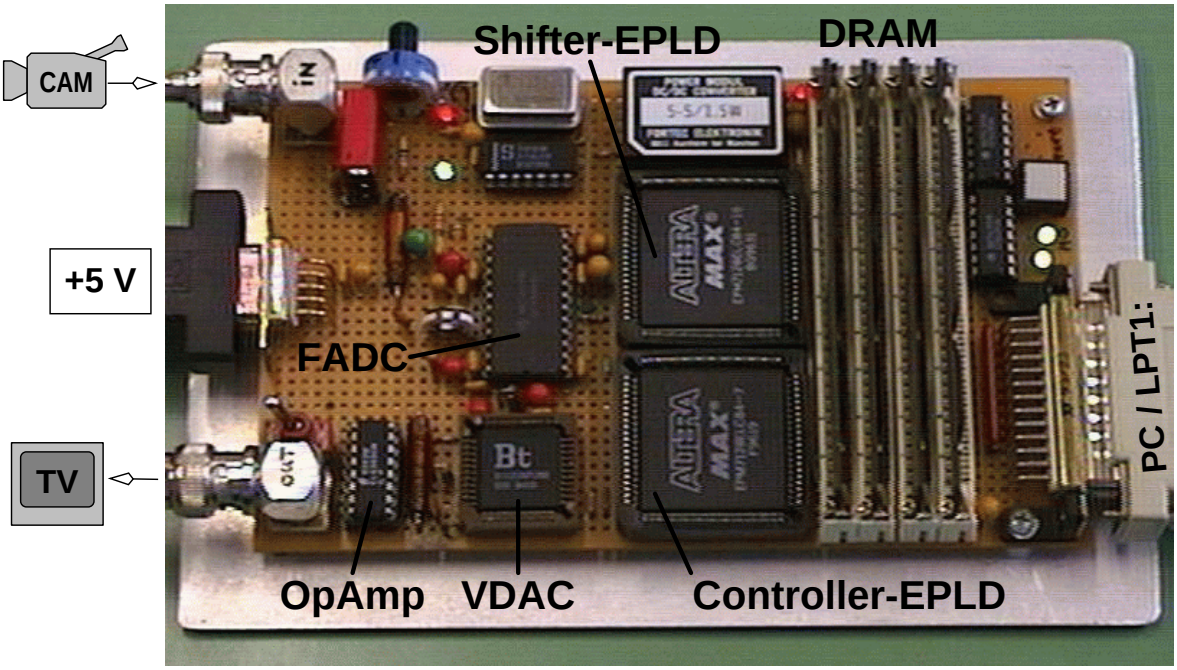


Abbildung 5.9: **VidTrans - Prototypplatine**

5.4 Digitaler PAL-Encoder

Die Aufgabe eines PAL-Encoders ist die Konvertierung von Farbbildinformation in ein Signalgemisch, das zur Darstellung dieser Farbbildinformation auf analogen PAL-Farbfernsehern geeignet ist. Ein PAL-Encoder konvertiert einen Bilddatenstrom

$$(r(t), g(t), b(t)) \longrightarrow (y(t), u(t), v(t)) \longrightarrow \text{fbas}(t). \tag{5.1}$$

Die in vorliegender Arbeit beschriebene Implementierung eines digitalen PAL-Encoders konvertiert einen YUV-Bilddatenstrom in einen digitalen FBAS-Bilddatenstrom mit einer Datenwortbreite von $8\ Bit$. Ein PAL-Encoder ist immer dann erforderlich, wenn Bildinformationen auf *analogen* PAL-Videogeräten dargestellt oder weiter verarbeitet werden sollen.

Komponenten

In Abbildung 4.26, S. 121 ist die als FPGA-Prototyp implementierte Architektur skizziert. Der PAL-Encoder-Kern setzt sich aus vier Komponenten zusammen, die jeweils in einer VHDL-Entity beschrieben sind. Der PAL-Encoder-FPGA-Prototyp verfügt über drei Betriebsmodi, die durch einen Schalter (SW) selektiert werden können. Im ersten Betriebsmodus kann eine Farbe aus dem YUV-Farbraum durch DIP-Schalter ausgewählt werden (s. Abbildung 5.10). Dabei kann der gewünschte Punkt im YUV-Farbraum mit einer Genauigkeit von insgesamt $16\ Bit$ spezifiziert werden. Durch die endliche Zahlendarstellungsgenauigkeit ist nur eine Teilmenge der Farben des YUV-Farbraums darstellbar. Das gilt insbesondere für Farben mit großen Beträgen von

u und v bei großem und kleinem y. Mögliche interne Zahlenbereichsüberschreitungen werden jedoch begrenzt. Durch einen zusammengesetzten Bit-Vektor der Form `y(7 downto 0) & u(3 downto 0) & v(3 downto 0)` erfolgt die Farbauswahl. Dabei ist `y` vom Typ `unsigned`, `u` und `v` sind vom Typ `signed` (ieee.std_logic_1164, ieee.std_logic_signed). Im zweiten Betriebsmodus wird y von einem Bildzähler geliefert, u und v werden aus höherwertigen Bits von Zeilen- und Spaltenadressen abgeleitet. So wird der gesamte adressierbare YUV-Farbraum durchlaufen. Im dritten Betriebsmodus wird der Bildzähler angehalten, so daß eine Ebene im YUV-Farbraum permanent dargestellt wird.

Timer

Der TIMER erzeugt alle für die Erzeugung eines FBAS-Videosignals notwendigen Signalkomponenten sowie interne Steuersignale für den PAL-Encoder. Außerdem werden Adressen zur Ansteuerung von externen SRAMs erzeugt, die aber bei dem FPGA-Prototypen nicht herausgeführt sind.

Farbträger-Generator

Die Komponente DDS ($\approx$ **D**irekte **D**igital **S**ynthese) erzeugt das PAL-Farbträgersignal (2.46) durch fortschreitende Akkumulation einer Phasenkonstante $\Delta\varphi$ mit jedem Abtastschritt (vgl. *Quadrature Subcarrier Generation* mit *Ratio Counter* [Jack, 1996]), wie in Abschnitt 2.3.4, ab Seite 35 beschrieben. Beim vorliegenden Entwurf wird jedoch anstelle einer in einem ROM abgelegten Sinus-Tabelle diese durch ein Schaltnetz (synthetisierbare VHDL-Beschreibung) repräsentiert. Dadurch kann bei der Schaltungssynthese auf (ROM-)Generatoren verzichtet werden, wodurch die Wiederverwendbarkeit etwas vereinfacht werden kann [Keating und Bricaud, 1998]. Ein Farbträgersignal (2.46) läßt sich allgemein durch den Ausdruck (2.38) beschreiben. Die Phasenwinkeldifferenz zwischen aufeinander folgenden Abtastwerten beträgt

$$\Delta\varphi = 2\pi \cdot \frac{f_{PAL}}{f_s}. \tag{5.2}$$

Dabei ist die Frequenz des PAL-Farbträgers $f_{PAL} = 4,43361875$ MHz und die Abtastfrequenz ist $f_s = 15$ MHz. Die Wortbreite des Phasenakkumulators wurde auf $\mathtt{a} = 24$ Bit festgelegt. Damit ergibt sich für die Phasenwinkelkonstante aus Gleichung (2.41)

$$\begin{aligned} \mathtt{C} &= 2^{\mathtt{a}} \cdot \frac{f_{PAL}}{f_s} \\ &= 2^{24} \cdot 4,43361875 \text{ MHz} / 15 \text{ MHz} \\ &= 4958918,628693. \end{aligned} \tag{5.3}$$

Aufgerundet auf die nächstgrößere ganze Zahl ist im Dezimalsystem $\mathtt{C} = (4958919)_{10}$ und im Dualsystem ist $\mathtt{C} = (010010111010101011000111)_2$. Für den Prototypen wird eine Tabelle mit $\mathtt{N} = 32$ Sinus-Funktionswerten verwendet. Für $\mathtt{C} = (4958919)_{10}$ beträgt die Frequenz f_{DDS} des durch DDS erzeugten Sinus-Signals

$$f_{DDS} \quad = \quad \mathtt{C} \cdot 2^{-\mathtt{a}} \cdot f_s. \tag{5.4}$$

Die Abweichung $\Delta f = |f_{PAL} - f_{DDS}|$ zwischen Sollfrequenz f_{PAL} und f_{DDS} beträgt damit

$$\begin{array}{rcl} \Delta f & \approx & |4,433618750 \text{ MHz} - 4,433619082 \text{ MHz}| \\ & \approx & 0,332 \text{ Hz}. \end{array} \tag{5.5}$$

Die Validierung des PAL-Encoders in der Anwendungsumgebung zeigte, daß dieser Wert für die Erzeugung der PAL-Farbträgersignals mit hinreichender Frequenzgenauigkeit geeignet ist.

QAM-Modulator

Der QAM-Modulator moduliert die Farbdifferenzsignale $u(t)$ und $v(t)$ gemäß (2.44) und erzeugt so das Chrominanz-Signal $c(t)$. Dabei wird die Berechnung (2.44) bei jedem zweiten Takt ausgewertet. Daher wird der digitale PAL-Encoder mit der doppelten Frequenz der für das FBAS-Signal gewünschten Frequenz des FBAS-Signals betrieben. Der QAM-Modulator ist als MAC-Einheit (**M***ultiply-and-***AC***cumulate*) implementiert. Dadurch wird zur Berechnung von (2.44) nur *ein* Multiplizierer benötigt, der wegen der hohen erforderlichen Verarbeitungsgeschwindigkeit als Parallel-Multiplizierer (Schaltnetz) realisiert werden muß. Bei der ersten (τ_0) von zwei (. . ., τ_0, τ_1, . . .) aufeinander folgenden Taktflanken wird die MAC-Einheit mit $u(t) \cdot cos(\omega t)$ initialisiert. Bei der zweiten Taktflanke (τ_1) der beiden aufeinander folgenden Taktflanken (. . ., τ_0, τ_1, . . .) wird $v(t) \cdot sin(\omega t)$ zu dem im Akkumulator-Register A gespeicherten Wert hinzu addiert. Die Weitergabe des im Akkumulator gespeicherten Wertes an den Videomischer erfolgt mit jedem zweiten Takt, so daß gilt

$$\begin{array}{rcl} A_{\tau_0} & := & u(t) \cdot \cos(\omega \cdot \tau_0) \\ A_{\tau_1} & := & A_{\tau_0} \pm v(t) \cdot \sin(\omega \cdot \tau_0). \end{array} \tag{5.6}$$

Dabei sind a_{τ_0} und a_{τ_1} die Werte des Akkumulator-Registers zweier aufeinander folgender Takte. Da die Weitergabe des Akkumulator-Inhalts nur mit jeder zweiten Taktflanke erfolgt, ist $c_{\tau_0} = c_{\tau_1} = A_{\tau_1}$.

Videomischer

Aufgabe des Videomischers MIXER ist es, ein vorzeichenloses FBAS-Signal durch Addition von konstantem Schwarzpegel, Luminanz-Signal y und Chrominanz-Signal c zu berechnen. Darüber hinaus kommt dem MIXER die Aufgabe der Selektion der Burst-Signale zu. Dazu werden geeignete Steuersignale vom TIMER generiert. Außerdem überwacht der MIXER Zahlenbereichsüberschreitungen und begrenzt diese gegebenenfalls auf den gültigen FBAS-Zahlenbereich ($0 \ldots 255$). Dabei ist entscheidend, daß die Addition des Chrominanz-Signals zum Luminanz-Signal *vorzeichenbehaftet* erfolgt, *bevor* das FBAS-Signal in ein vorzeichenloses Signal konvertiert wird. Wird das Chrominanz-Signal vor der Addition in ein vorzeichenloses Signal überführt, so hat die Addition nicht die erforderliche Semantik. Vom TIMER wird die Auswahl der Signalkomponenten durch Steuersignale csync, black, burst bestimmt, von denen immer

nur ein Signal aktiv ist, mit Ausnahme von black. Die Signale csync und burst haben jedoch gegenüber dem Signal black Priorität.

$$\text{fbas}(t) \quad = \quad \begin{cases} \text{SYNC-Pegel} & \text{wenn} \quad \text{csync} = aktiv \\ \text{Schwarz-Pegel} & \text{wenn} \quad \text{black} = aktiv \\ \text{Schwarz-Pegel} + \frac{1}{8} \cdot f(t, \Phi) & \text{wenn} \quad \text{burst} = aktiv \\ \text{Schwarz-Pegel} + y(t) + c(t) & \text{sonst.} \end{cases} \tag{5.7}$$

Für den Fall, daß fbas(t) kleiner als null wird, wird fbas(t) der Wert 0 zugewiesen. Ergibt sich für fbas(t) ein Wert größer als $2^8 - 1$, so wird fbas(t) auf $2^8 - 1$ gesetzt. Dazu erfolgen die Berechnungen (5.7) mit hinreichend großen Wortbreiten zur Vermeidung von Zahlenbereichsüberschreitungen, da für die VHDL-Verhaltensbeschreibung der Operator + direkt verwendet wird, um die Lesbarkeit der VHDL-Beschreibung zu erhalten. Die Handhabung von Carry-Signalen zur Überlaufbehandlung wurde in diesem Zusammenhang gar nicht erst in Betracht gezogen. Das digitale FBAS-Signal wird durch einen 8-Bit-Video-DAC vom Typ Philips TDA8702 [Philips Semiconductors, 1996b] in ein analoges FBAS-Videosignal konvertiert.

Näherungen

Zur Vermeidung bzw. Reduzierung von Über- und Untersteuerung werden die Farbdifferenzsignale zuvor gemäß $u := u \cdot 0,49$ und $v := v \cdot 0,88$ reduziert. Diese Reduzierung kann durch $u := u \cdot 0,50$ und $v := v \cdot (1-0,125)$ erfolgen, da $0,49 \approx 0,5 = 2^{-1}$ und $0,88 \approx 0,85 = 1-2^{-3}$.

Design-Ablauf

Grundlage der vorliegenden VHDL-Implementierung des digitalen PAL-Encoders ist eine in der Sprache C geschriebene ausführbare Spezifikation eines digitalen PAL-Encoders. Dieses Programm wurde vom Autor des vorliegenden Beitrags im Rahmen betreuter Studien- und Diplomarbeiten [Jürgens, 1996, 1999] geschrieben. Die ausführbare Spezifikation des digitalen PAL-Encoders diente primär der Validierung des Signalverarbeitungsschemas eines digitalen PAL-Encoders. Das C-Programm wurde mit der bidirektionalen Videosignalschnittstelle *VidTrans* [Larsson, 1996] in realer Systemumgebung unter Echtzeitbedingung validiert. Die Validierung der ausführbaren Spezifikation des digitalen PAL-Encoders durch Darstellung digital erzeugter PAL-Videosignale mit der Videosignalschnittstelle *VidTrans* hat sich bei der vorliegenden Signalverarbeitungsaufgabe als unverzichtbares Hilfsmittel erwiesen. Darüber hinaus konnte so auch die hohe, durch einen digitalen PAL-Encoder grundsätzlich erreichbare, Bildqualität *vor* der Hardware-Implementierung evaluiert werden. In Abbildung 4.25 sind synthetische Farbbilder, mit der C-Implementierung des digitalen PAL-Encoders erzeugte Videosignalausschnitte und die mit einer C-Implementierung eines digitalen PAL-Decoders rück-extrahierten Farbbilder dargestellt. Die Unterschiede zwischen den Farbbildern sind derart gering, daß sie bei Ausgabe über handelsübliche Farbtintendrucker oder Farblaserdrucker nicht wahrnehmbar sind. Quantisierungsartefakte lassen sich erst bei Darstellung auf analogen Farbfernsehgeräten beobachten.

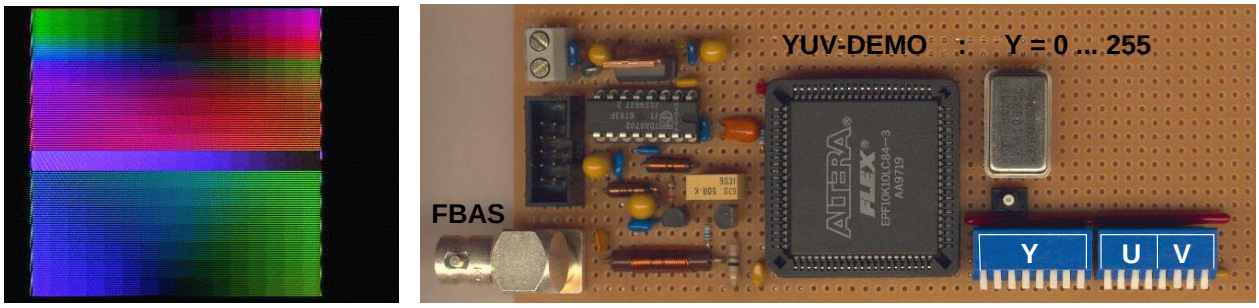


Abbildung 5.10: **PAL-Video-Bild und PAL-Encoder-FPGA-Prototyp-Board.** *Auf der linken Seite der Abbildung ist ein PAL-Videobild als Ergebnis einer funktionalen VHDL-Simulation dargestellt, wie es der Prototyp des FPGA-PAL-Encoders liefert. Das PAL-Videobild wurde mit einem Software-PAL-Decoder aus dem Videosignaldatenstrom der VHDL-Simulation extrahiert. [Larsson, 1999b]*

Top-Down-Architektur-Konzept – Bottom-Up-Implementierung

Die Konzipierung der Architektur (Abbildung 4.26, S. 121) des digitalen PAL-Encoders erfolgte Top-Down. Der Entwurf der Komponenten erfolgte dagegen Bottom-Up in der Reihenfolge: TIMER, DDS, QAM, MIXER. Abschließend wurde die Komponente YUV-DEMO entworfen. Für den vorliegenden PAL-Encoder wurde die benötigte Sinus-Tabelle mit einem C-Programm als VHDL-Beschreibung (ein Prozeß mit `case` Statement in einer VHDL-Entity) generiert, die das Verhalten eines Schaltnetzes impliziert [Keating und Bricaud, 1998; Bhasker, 1996] und von der Logiksynthese nach Möglichkeit als einstufiges Schaltnetz realisiert wird oder zumindest in ein Schaltnetz mit wenig Stufen umgesetzt wird.

Für jede Komponente wurde nach der Fertigstellung eine eigene Testumgebung implementiert, um die Komponenten vor der Instantiierung intensiv durch Verhaltenssimulation zu validieren. Jede Komponente wurde nach der Validierung synthetisiert, um die Synthetisierbarkeit der Komponenten und später des gesamten PAL-Encoders zu gewährleisten. Abschließend wurde die VHDL-Beschreibung des PAL-Encoders zusammen mit einer geeigneten Testumgebung erstellt. Für die Testumgebung des PAL-Encoders wurde ein vereinfachtes Verhaltensmodell des verwendeten 8-Bit-Video-DACs (TDA8702 [Philips Semiconductors, 1996b]) erstellt, welches die Abtastwerte des digitalen FBAS-Datenstroms in eine Textdatei schreibt. Diese Textdatei dient als Basis für die graphische Signaldarstellung (wie in Abbildung 4.25) oder Videobildextraktion (wie in Abbildung 5.10) des erzeugten PAL-Videosignals und zur Darstellung des erzeugten PAL-Videosignals auf einem analogen Farbfernsehgerät mit Hilfe von *VidTrans* [Larsson, 1996].

FPGA-Prototyp-Board

Zur Validierung des VHDL-Modells in der Anwendungsumgebung wurde eine Prototyp-Platine (s. Abbildung 5.10) innerhalb eines Arbeitstages in Fädeltechnik aufgebaut. Der zügige Aufbau des Prototyp-Boards wurde durch die Verwendung nur eines einzigen FPGAs, welches hinreichend viele Gatter-Äquivalente zur Verfügung stellt, ermöglicht. Neben dem FPGA (Altera EPF10K10LC84-3, 10 K Gatteräquivalente [Altera Corporation, 1998b]) befinden sich auf dem Prototyp-Board noch ein 30-MHz-Quarzoszillator, einige Schalter und Buchsen, sowie der 8-Bit-

Video-DAC (Philips TDA8702 [Philips Semiconductors, 1996b]). Die Beschaltung des Video-DACs mit einigen analogen Komponenten wurde im wesentlichen dem Video-DAC-Datenblatt [Philips Semiconductors, 1996b] entnommen.

Schlußfolgerungen

Eine synthetisierbare VHDL-Verhaltensbeschreibung eines digitalen PAL-Encoders, die sogar für die Implementierung als FPGA-Prototyp geeignet ist, ist dann für digitale System-on-a-Chip-Entwürfe (SoC) geeignet, wenn beim Entwurf Leitlinien für SoC-Designs [Keating und Bricaud, 1998] berücksichtigt werden. So wurde beim vorliegenden Entwurf auf die Ausnutzung von herstellerspezifischen Merkmalen des zum Prototyping verwendeten FPGAs verzichtet. Es wurde ein rein synchroner Entwurf mit Einflanken-Triggerung realisiert, und es wurden nur Moore-Automaten verwendet. Dadurch steht das Verhalten der funktionalen Simulation mit dem Verhalten des synthetisierten FPGAs, das im System unter Echtzeitbedingungen validiert werden konnte, im Einklang.

Standardzellen- und Full-Custom-Entwürfe [Weste und Eshraghian, 1993] bieten Möglichkeiten, die beim Prototyping mit FPGAs nicht zwingend zur Verfügung stehen. So ist etwa die Verwendung von größeren On-Chip-Speichern bei FPGAs im allgemeinen nicht effizient möglich. Zwar unterstützen FPGAs der Flex10K-Familie [Altera Corporation, 1998b] bis zu einem gewissen Grade die Verwendung von On-Chip-RAM, jedoch nur als herstellerspezifische Makroblocks.

Kapitel 6

Zusammenfassung der Ergebnisse

Ziel dieser Arbeit war, Verfahren zur Validierung der Spezifikationen von videobildverarbeitenden Komponenten unter Berücksichtigung realer Anwendungsbedingungen zu entwickeln. Der neue Ansatz der vorliegenden Arbeit ist die *systemnahe Validierung originärer Spezifikationen* – Verhaltensbeschreibungen von Komponenten – durch Beobachtung des Verhaltens implementierter Komponenten auf der Systemebene. Damit lassen sich auch Implementierungen auf unterschiedlichen Plattformen vergleichen, die im allgemeinen zueinander kompatibles, aber nicht äquivalentes Verhalten zeigen.

Ausgehend von einem Architekturkonzept zur Echtzeitverarbeitung von Videobildern mit neuronalen Netzwerken [Larsson *et al.*, 1996, 1997a,b] wurde eine neue Methode zur Visualisierung des Lernvorgangs von Backpropagation-Netzwerken [Rumelhart *et al.*, 1986a,b] unter dem Namen *Lerntrajektorien* eingeführt [Larsson, 1997, 1999a]. Diese Methode projiziert den Verlauf der Veränderung aller synaptischen Gewichte eines neuronalen Netzwerks während des Lernens in eine zweidimensionale Darstellung. Damit ist es möglich, Störungen des Lernvorgangs, wie sie durch numerische Vereinfachungen – Festkommadarstellung statt Gleitkommadarstellung – verursacht werden, direkt zu beobachten. Die Visualisierung hochdimensionaler Zustandsänderungen durch Lerntrajektorien ist nicht auf die Visualisierung des Lernens mit dem Backpropagation-Algorithmus beschränkt, sondern scheint für die Parametrierung und Weiterentwicklung von Algorithmen, die dem Gradientenverfahren verwandt sind, geeignet zu sein.

Zur Entwicklung und Validierung digitaler videosignalverarbeitender Komponenten wurde eine bidirektionale Schnittstelle als Brücke zwischen Simulationen und Realsystemen geschaffen [Larsson, 1996], die später unter dem Namen *VidTrans* auf internationalen Fachmessen ausgestellt sowie bei regionalen Informationsveranstaltungen praktisch demonstriert wurde. Mit Hilfe von *VidTrans* konnten PAL-Encoder und PAL-Decoder als Software-Implementierung realisiert werden, die eine Bildqualität liefern, die mit handelsüblichen Videogeräten vergleichbar ist. Der Erfolg *systemnaher Validierung* wurde abschließend mit einem Fallbeispiel durch Hardware-Implementierung eines digitalen PAL-Farbvideosignal-Encoders und Vergleich mit der funktional ähnlichen Software-Implementierung praktisch demonstriert [Larsson, 1999b]. Die digitale Verarbeitung analoger Farbvideosignale ist direkt für die Multimediatechnologie von Bedeutung.

Je eine Software-Implementierung und Hardware-Implementierung eines digitalen PAL-Videosignal-Encoders [Larsson, 1999a], der später als *PalCo* bezeichnet wurde, dienen im Rahmen der vorliegenden Arbeit als Fallbeispiel für den direkten Vergleich zweier *funktional ähnlicher* Implementierungen auf unterschiedlichen Plattformen. Die Hardware-Implementierung *PalCo* steht nun grundsätzlich als synthetisierbare validierte VHDL-Verhaltensbeschreibung zur Verfügung. Darüber hinaus ist eine Software-Implementierung eines digitalen PAL-Videosignal-Decoders namens *CFrame* entstanden, der Farbbilder in hoher Bildqualität aus einem digitalisierten PAL-Farbvideosignaldatenstrom extrahiert. Die Software-Implementierungen von PAL-Encoder und PAL-Decoder sind begleitend zu einer vom Autor betreuten Studienarbeit [Jürgens, 1996] und Diplomarbeit [Jürgens, 1999] entstanden.

6.1 Ausblick

Für Lerntrajektorien sind, über die Visualisierung des Backpropagation-Lernens hinaus gehende, weitere Anwendungen denkbar. So scheinen Lerntrajektorien auch zur Steuerung von den Backpropagation-Lernalgorithmus bestimmenden Parametern während des Lernens geeignet zu sein, unter der Annahme, durch lokale Minima stärker gekrümmte Bereiche der Gesamtfehlerfunktion des Backpropagation-Netzwerks von Plateaus der Gesamtfehlerfunktion unterscheiden zu können. Lerntrajektorien könnten in allgemeinerem Sinne auch zur Verhaltensbeobachtung von anderen Vielparametersystemen, hochdimensionalen Prozessen und anderen Algorithmen, die einen Satz von vielen Parametern quasi gleichzeitig verändern, geeignet sein und dadurch Parametrierungen oder auch Weiterentwicklungen von entsprechenden Algorithmen unterstützen. Lerntrajektorien könnten aber auch zur direkten Vorverarbeitung für neuronale Netzwerke dienen. Wird ein Bild als ein Punkt in einem hochdimensionalen Raum verstanden, so stellt sich eine Sequenz von aufeinander folgenden Bildern als eine Folge von Punkten in diesem hochdimensionalen Raum dar, die möglicherweise für Bildsequenzen charakteristische Lerntrajektorien in zweidimensionaler Darstellung liefern. Eine in eine Lerntrajektorie transformierte Bildsequenz könnte dann einem neuronalen Netzwerk als Eingabe dienen und so zur Identifikation von Bildsequenzen eingesetzt werden.

Die mit Hilfe von *VidTrans* implementierten digitalen PAL-Encoder und PAL-Decoder können als Ansatz für die Entwicklung eines generischen Farbfernseherverhaltensmodells dienen, welches dann für Simulationen in einer geschlossenen Simulationsumgebung zur Verfügung stehen würde. Die Erweiterung auf den amerikanischen NTSC-Videostandard oder auf den französischen SECAM-Videostandard ist denkbar und für industrielle Entwicklung von Geräten erforderlich, die weltweit eingesetzt werden sollen. Darüber hinaus ist durch Kombination von digitaler Videosignalverarbeitung und neuronalen Netzwerken die Entwicklung von Meßgeräten zur Klassifizierung der Qualität von Bildern denkbar, indem ein geeignetes neuronales Netzwerk lernt, das Empfinden von Bildqualität verschiedener Testpersonen nachzubilden.

6.2 Schlußfolgerungen

Anhand der im Rahmen der vorliegenden Arbeit entstandenen Implementierungen wird die Notwendigkeit der neu eingeführten Methoden zur Validierung originärer Spezifikationen unter Berücksichtigung realer Anwendungsumgebungen gezeigt. Die eingeführten Methoden leisten einen direkten Beitrag im Bereich der Entwurfsmethodik. Darüber hinaus sind *Lerntrajektorien* zur Visualisierung hochdimensionaler Prozesse geeignet und scheinen daher etwa für die Weiterentwicklung bestimmter Optimierungsalgorithmen geeignet zu sein. Die Realisierung von *VidTrans* hat gezeigt, daß ein solches Hilfsmittel bei der Entwicklung bestimmter digitaler videosignalverarbeitender Komponenten unverzichtbar ist.

Anhang A

Abkürzungen

ASCII	:	**A**merican **S**andard **C**ode for **I**nformation **I**nterchange
ADC	:	**A**nalog **D**igital **C**onverter
ALVINN	:	**A**utonomous **L**and **V**ehicle **i**n a **N**eural **N**etwork
ASIC	:	**A**pplication **S**pecific **I**ntegrated **C**ircuit
BAS - Signal	:	**B**ild-**A**ustast-**S**ynchron - Signal
BPG - Algorithmus	:	**B**ack**p**ropa**g**ation - Algorithmus
CCIR	:	**C**omité **C**onsultativ **I**nternational des **R**adiocommunications
CMOS	:	**C**omplimentary **M**etal **O**xide **S**emiconductor
CPS	:	**C**onnections **P**er **S**econd
CUPS	:	**C**onnection **U**pdates **P**er **S**econd
DAC	:	**D**igital **A**nalog **C**onverter
DDS	:	**D**irect **D**igital **S**ynthesis
DES	:	**D**ata **E**ncryption **S**tandard
DMA	:	**D**irect **M**emory **A**ccess
DRAM	:	**D**ynamic **R**andom **A**ccess **M**emory
EPLD	:	**E**lectrically **P**rogrammable **L**ogic **D**evice
FADC	:	**F**lash **A**nalog **D**igital **C**onverter
FBAS - Signal	:	**F**arb-**B**ild-**A**ustast-**S**ynchron - Signal
FFT	:	**F**ast **F**ourier **T**ransformation
FIR - Filter	:	**F**inite **I**mpulse **R**esponse - Filter
FPGA	:	**F**ield **P**rogrammable **G**ate **A**rray
FPU	:	**F**loating **P**oint **U**nit
IIR - Filter	:	**I**ninite **I**mpulse **R**esponse - Filter
IO	:	**I**nput **O**utput
IP	:	**I**ntellectual **P**roperty
LUT	:	**L**ook **U**p **T**able

MAC - Einheit	:	**M**ultiply and **AC**cumulate - Einheit
MOPS	:	**M**illion of **O**perations **P**er **S**econd
MSB	:	**M**ost **S**ignificant **B**it *oder* **M**ost **S**ignificant **B**yte
MTS	:	**M**ultiple **T**ime **S**cale
MVL	:	**M**ultiple **V**alued **L**ogic
NCO	:	**N**umerically **C**ontrolled **O**scillator
NTSC - Videostandard	:	**N**ational **T**elevision **S**ystem **C**ommittee - Videostandard
NeNEB	:	**Ne**uronales **N**etzwerk zur **E**chtzeitklassifizierung von **B**ildern
OpAmp	:	**O**perational **A**mplifier
PA	:	**P**rocessor **A**rray
PAL - Chip	:	**P**rogrammable **A**rray **L**ogic - Chip
PAL - Videostandard	:	**P**hase **A**lternation **L**ine - Videostandard
PalCo - FPGA	:	**P**AL-En**Co**der - FPGA
PLD	:	**P**rogrammable **L**ogic **D**evice
PLL	:	**P**hase **L**ocked **L**oop
QAM	:	**Q**uadratur **A**mplituden **M**odulation
RMS	:	**R**oot **M**ean **S**quare
ROM	:	**R**ead **O**nly **M**emory
RT - Ebene	:	**R**egister **T**ransfer - Ebene
SIMD	:	**S**ingle **I**nstruction **M**ultiple **D**ata
SNR	:	**S**ignal **N**oise **R**atio
SoC	:	**S**ystem-**o**n-a-**C**hip
SOM	:	**S**elf-**O**rganizing **M**aps
SRAM	:	**S**tatic **R**andom **A**ccess **M**emory
TP	:	**T**ief**p**aß
VDAC	:	**V**ideo **D**igital **A**nalog **C**onverter
VCO	:	**V**oltage **C**ontrolled **O**scillator
VHDL (**V**HSIC **HDL**)	:	**V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage
VidTrans	:	**Vid**eosignal-**Trans**ienten-Recorder
VLSI	:	**V**ery **L**arge **S**cale **I**ntegration

Anhang B

Umformungen

B.1 Sigmoidfunktion

Ableitung der Sigmoidfunktion

$$
\begin{aligned}
g(x) &= \frac{1}{1+e^{-x/\tau}} \\
&= (1+e^{-x/\tau})^{-1} \\
\\
g'(x) &= (-1)\cdot(1+e^{-x/tau})\cdot e^{-x/\tau}\cdot\tfrac{-1}{\tau} \\
&= \frac{1}{\tau}\cdot\frac{e^{-x/\tau}}{(1+e^{-x/\tau})\cdot(1+e^{-x/\tau})} \qquad \left|\cdot\frac{e^{x/\tau}}{e^{x/\tau}}\right. \\
&= \frac{1}{\tau}\cdot\frac{1}{(1+e^{x/\tau})}\cdot\frac{1}{(1+e^{-x/\tau})} \\
&= \frac{1}{\tau}\cdot g(x)\cdot g(-x)
\end{aligned}
\tag{B.1}
$$

B.2 Gebrochenrationale Funktion

$$
\begin{aligned}
f(r) &= \frac{1}{1+r^2} \\
f(0) &= 1 \\
\lim_{|r|\to\infty} f(r) &= 0 \\
f'(r) &= \frac{\mathrm{d}}{\mathrm{d}r}(1+r^2)^{-1} \\
&= -2\cdot(1+r^2)^{-2}\cdot 2r \\
&= \frac{-4r}{(1+r^2)^2} \\
&= \frac{-4r}{1+2r^2+r^4}
\end{aligned}
\tag{B.2}
$$

$$
\frac{1}{1+2r^2+r^4} > 0 \quad \forall \quad r \qquad \Rightarrow \qquad f'(r) = 0 \quad \text{für} \quad r = 0.
$$

B.3 Quadraturamplitudenmodulation

$$c(t) = u(t) \cdot \cos(\omega t) + v(t) \cdot \sin(\omega t)$$

$$\begin{aligned} c_{cos}(t) &= c(t) \cdot \cos(\omega t) \\ &= u(t) \cdot \cos(\omega t) \cdot \cos(\omega t) + v(t) \cdot \sin(\omega t) \cdot \cos(\omega t) \\ &= u(t) \cdot \cos^2(\omega t) + v(t) \cdot \sin(\omega t) \cdot \cos(\omega t) \\ &= \tfrac{1}{2}u(t) + \tfrac{1}{2}u(t) \cdot \cos(2\omega t) + \tfrac{1}{2}v(t) \cdot \sin(2\omega t) \end{aligned} \quad \text{(B.3)}$$

$$\begin{aligned} c_{sin}(t) &= c(t) \cdot \sin(\omega t) \\ &= u(t) \cdot \sin(\omega t) \cdot \cos(\omega t) + v(t) \cdot \sin(\omega t) \cdot \sin(\omega t) \\ &= u(t) \cdot \sin(\omega t) \cdot \cos(\omega t) + v(t) \cdot \sin^2(\omega t) \\ &= \tfrac{1}{2}u(t) \cdot \sin(2\omega t) + \tfrac{1}{2}v(t) - \tfrac{1}{2}v(t) \cdot \cos(2\omega t) \end{aligned}$$

Produkte Trigonometischer Funktionen [Bronstein und Semendjajew, 1981]

$$\begin{aligned} \sin(x) \cdot \sin(y) &= \tfrac{1}{2} \cdot (\cos(x-y) - \cos(x+y)) \\ \sin^2(x) &= \tfrac{1}{2} \cdot (\cos(0) - \cos(2x)) \\ &= \tfrac{1}{2} - \tfrac{1}{2} \cdot \cos(2x) \end{aligned}$$

$$\begin{aligned} \cos(x) \cdot \cos(y) &= \tfrac{1}{2} \cdot (\cos(x-y) + \cos(x+y)) \\ \cos^2(x) &= \tfrac{1}{2} \cdot (\cos(0) + \cos(2x)) \\ &= \tfrac{1}{2} + \tfrac{1}{2} \cdot \cos(2x) \end{aligned} \quad \text{(B.4)}$$

$$\begin{aligned} \sin(x) \cdot \cos(y) &= \tfrac{1}{2} \cdot (\sin(x-y) + \sin(x+y)) \\ \sin(x) \cdot \cos(x) &= \tfrac{1}{2} \cdot (\sin(0) + \sin(2x)) \\ &= \tfrac{1}{2} \cdot \sin(2x) \end{aligned}$$

B.4 Filteraufwandsabschätzungen

Filteraufwandsabschätzung nach Bellanger [Fliege, 1993]

$$
\begin{aligned}
N &\approx \frac{2}{3} \cdot \log_{10}\left(\frac{1}{10 \cdot \delta_D \cdot \delta_S}\right) \cdot \frac{1}{b} \\
N &\approx \frac{2}{3} \cdot \log_{10}\left(\frac{1}{10 \cdot \delta^2}\right) \cdot \frac{1}{b} \qquad \text{mit} \qquad \delta := \delta_D = \delta_S \\
\frac{3}{2} \cdot N \cdot b &\approx \log_{10}\left(\frac{1}{10 \cdot \delta^2}\right) \\
10^{-\frac{3}{2} \cdot N \cdot b} &\approx 10 \cdot \delta^2 \\
\delta &\approx \sqrt{10^{-\frac{3}{2} \cdot N \cdot b - 1}} \\
\delta\text{[dB]} &= 20 \cdot \log_{10}(\delta) \\
\delta\text{[dB]} &\approx 20 \cdot \log_{10}\left(\sqrt{10^{-\frac{3}{2} \cdot N \cdot b - 1}}\right) \\
&= \tfrac{1}{2} \cdot 20 \cdot \log_{10}\left(10^{-\frac{3}{2} \cdot N \cdot b - 1}\right) \\
&= 10 \cdot \left(-\tfrac{3}{2} \cdot N \cdot b - 1\right) \\
\hookrightarrow \qquad \delta\text{[dB]} &\approx -15 \cdot N \cdot b - 10
\end{aligned}
\tag{B.5}
$$

Filteraufwandsabschätzung nach Kaiser [Fliege, 1993]

$$
\begin{aligned}
N &\approx \frac{-20 \cdot \log_{10}(\sqrt{\delta_D \cdot \delta_S}) - 13}{14,6 \cdot b} + 1 \\
N &\approx \frac{-20 \cdot \log_{10}(\sqrt{\delta^2}) - 13}{14,6 \cdot b} + 1 \quad \text{mit} \quad \delta := \delta_D = \delta_S \\
N - 1 &\approx \frac{-20 \cdot \log_{10}(\delta) - 13}{14,6 \cdot b} \\
14,6 \cdot (N-1) \cdot b &\approx -20 \cdot \log_{10}(\delta) - 13 \\
14,6 \cdot (N-1) \cdot b + 13 &\approx -20 \cdot \log_{10}(\delta) \\
\delta\text{[dB]} &= 20 \cdot \log_{10}(\delta) \\
\delta &= 10^{\delta\text{[dB]}/20} \\
14,6 \cdot (N-1) \cdot b + 13 &\approx -20 \cdot \log_{10}(10^{\delta\text{[dB]}/20}) \\
&= -20 \cdot \delta\text{[dB]}/20 \\
&= -1 \cdot \delta\text{[dB]} \\
\hookrightarrow \qquad \delta\text{[dB]} &\approx -14,6 \cdot (N-1) \cdot b - 13
\end{aligned}
\tag{B.6}
$$

Anhang C

Tabellen

Abtastfrequenz	15	MHz
Datenwortbreite	8	Bit (*unsigned*)
Filterkoeffizientenwortbreite	1+7	Bit (*signed*)
Koeffizientenzahl (Ordnung-1)	31	(30)
Chip-Fläche (ecpd07, 0.7μ)	$\approx$ 15	mm^2
1 MAC-Unit (Altera FLEX10K100)	$\approx$ 4000	Gatteräquivalente

Tabelle C.1: **FIR-Filter – Implementierungsparameter.**

		Filterperformanz	
Filtermodell	Verarbeitungszeit	absolut	relativ
`FIR` (Hardware, Echtzeit)	0,16 s	15 Msps	$\approx$ **1** (465 M MAC/s)
`fir.c` (Software)	13 s	190 Ksps	$\approx$ 1 / 100
`fir.vhd` (Verhaltensmodell)	30 m	1,3 Ksps	$\approx$ 1 / 10.000
`fir_gl.vhd` (Gate Level)	40 h	17,5 sps	$\approx$ 1 / 1.000.000

Tabelle C.2: **FIR-Filter – Simulationszeiten.** *Die angegebene Filterperformanz bezieht sich auf 31 Filterkoeffizienten – für jeden Abtastwert sind 31 MAC-Operationen zu berechnen.*

Literaturverzeichnis

Altera Corporation (1998a). *Altera – APEX 20K Programmable Logic Family – Advance Information Brief, Ver. 1*. San Jose, California, USA.

Altera Corporation (1998b). *Altera – FLEX 10K – Embedded Programmable Logic Family – Data Sheet Ver. 3.13*. San Jose, California, USA.

Altera Corporation (1998c). *Altera – MAX7000 – Programmable Logic Device – Data Sheet Ver. 5.03*. San Jose, California, USA.

Altera Corporation (1998d). *MAX+PLUS II – Programmable Logic Development System & Software – Data Sheet*. San Jose, California, USA.

G. An (1996). The Effect of Adding Noise During Backpropagation Training on Generalization Performance. *Neural Computation*, Band 8(3).

J. A. Anderson (1972). A Simple Neural Network Generating an Interactive Memory. *Mathemetical Biosciences*, Band 14:S. 197–220.

J. A. Anderson, E. Rosenfeld, Eds. (1988). *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, USA.

A. G. Andreou, K. A. Boahen, P. O. Pouliquen, A. Pavasovic, R. E. Jenkins, K. Strohbehn (1991). Current-Mode Subthreshold MOS Circuits for Analog VLSI Neural Systems. *IEEE Transactions on Neural Networks*, Band 2(1):S. 205–213.

D. Babcock, C. Lee, B. Gupta, J. Kim, R. Goodman (1996). Active Drag Reduction Using Neural Networks. In *Proc. of the 1996 IEEE Int. Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing, NICROSP'96*, S. 279–287. IEEE Computer Society Press, Los Alamitos, California.

P. Bakker, S. Phillips, J. Wiles (1993). The N-2-N Encoder: A Matter of Representation. In *Proc. of the Int. Conf. on Artificial Neural Networks, ICANN'93, Amsterdam, The Netherlands, 13-16 September 1993*, S. 554–557. Springer-Verlag, Heidelberg.

M. A. Bayoumi, Ed. (1994). *VLSI Design Methodologies for Digital Signal Processing Architectures*. Kluwer Academic Publishers, Boston.

M. Bellanger (1984). *Digital Processing of Signals*. John Wiley & Sons, New York.

R. E. Best (1996). *Phase Locked Loops: Design, Simulation, and Applications*. McGraw-Hill, Berkeley.

J. Bhasker (1996). *A VHDL Synthesis Primer*. Star Galaxy Publishing, Allentown, Pennsylvania.

J. Bhasker (1997). *A Verilog HDL Primer*. Star Galaxy Publishing, Allentown, Pennsylvania.

P. K. Bondyopadhyay (1998). Moore's Law Governs the Silicon Revolution. *Proceedings of the IEEE – Special Issue: 50th Anniversary of the Transistor*, S. 78–81.

B. A. Bowen, W. R. Brown (1982). *VLSI Systems Design for Digital Signal Processing*. Prentice-Hall International, London.

I. N. Bronstein, K. A. Semendjajew (1981). *Taschenbuch der Mathematik*. Verlag Harry Deutsch, 21. Auflage.

Y. Chauvin, D. E. Rumelhart, Eds. (1995). *Backpropagation – Theory, Architectures, and Applications (developments in connectionist theory)*. Erlbaum, Hillsdale, NJ.

J. Choi, B. J. Sheu (1995). *Neural Information Processing and VLSI*. Kluwer Academic Publishers, Boston.

J. Cloutier, P. Y. Simrad (1994). Hardware Implementation of Backpropagation without Multiplication. In *Proc. of Fourth Int. Conf. on Microelectronics for Neural Networks and Fuzzy Systems, Turin, Italy, 26-28 September 1994*, S. 46–55.

M. Costa, F. Palma, D. Palmisano, E. Pasero (1997). Applying Backpropagation through Time to a Real Inverted Pendulum Problem. In *Proc. of 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Berlin, Germany, 24-29 August 1997*, A. Sydow, Ed., Band 4, S. 161–166. Wissenschaft und Technik Verlag, Berlin.

R. M. Debenham, S. C. Grath (1989). Investigations into the effect of numerical resolution on the performance of back propagation. In *Proceedings of the NEuro'88 Conference – Neural Networks from Models to Applications, Paris, France, 6-9 June 1988*, L. Personaz, G. Dreyfus, Eds., S. 752–755.

Digital Semiconductor (1997). *Digital Semiconductor – Alpha 21164PC Microprocessor – Hardware Reference Manual*. Maynard, Massachusetts, USA.

X. Ding, S. Canu, T. Denoeux (1996). Neural Network Based Models for Forecasting. In *Neural Networks and their Applications*, J. G. Taylor, Ed., S. 153–167. John Wiley & Sons, New York.

D. A. Durfee, F. S. Shoucair (1992). Comparison of Floating Gate Neural Network Memory Cells in Standard VLSI CMOS Technology. *IEEE Transactions on Neural Networks*, Band 3(1):S. 347–353.

Fairchild Semiconductor (1998). *TMC22091/TMC22191 Digital Video Encoders/Layering Engine – Data Sheet*.

W. A. Fisher, R. J. Fujimoto, R. C. Smithson (1991). A Programmable Analog Neural Network Processor. *IEEE Transactions on Neural Networks*, Band 2(1):S. 222–229.

N. Fliege (1993). *Multiraten-Signalverarbeitung*. Teubner-Verlag, Stuttgart.

E. F. Foundation (1998). *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*. O'Reilly & Associates, Inc., Sebastopol, California.

R. C. Frye, E. A. Rietmann, C. C. Wong (1991). Back-Propagation Learning and Non-Idealities in Analog Neural Network Hardware. *IEEE Transactions on Neural Networks*, Band 2(1):S. 110–117.

M. Fukumi, S. Omatu (1991). A New Back-Propagation Algorithm with Coupled Neuron. *IEEE Transactions on Neural Networks*, Band 2(1):S. 535.

E. D. Gajski (1988). *Silicon Compilation*. Addison-Wesley.

S. I. Gallant (1993). Perceptron-Based Learning Algorithms. *IEEE Transactions on Neural Networks*, Band 3(1):S. 179–191.

P. Gerdsen (1996). *Digitale Nachrichtenübertragung*. Teubner-Verlag, Stuttgart.

B. Girod, R. Rabenstein, A. Stenger (1997). *Einführung in die Systemtheorie*. Teubner-Verlag, Stuttgart.

M. Glesner, W. Pöchmüller (1994). *Neurocomputers – An Overview of Neural Networks in VLSI*. Chapman & Hall, London.

V. Grupe, R. Rauscher (1991). An A/D-Chip for Accurate Power Measurement. In *Proceedings of EURO ASIC '91, Paris, France*, S. 352–355. IEEE Computer Society Press, Los Alamitos, California.

L. H. Hahn (1995). Realisierung eines Echtzeit-Video-Digitalisierers unter Verwendung einer Hardware - Beschreibungssprache (AHDL, VHDL) als FPGA Prototyp. Studienarbeit, Universität Hamburg, Fachbereich Informatik.

A. Hamilton, A. F. Murray, D. J. Baxter, S. Churcher, H. M. Reekie, L. Tarassenko (1992). Integrated Pulse Stream Neural Networks: Results, Issues, and Pointers. *IEEE Transactions on Neural Networks*, Band 3(1):S. 385–393.

R. W. Hamming (1983). *Digital Filters*. Prentice-Hall International, London, 2. Auflage.

Harris Semiconductor (1998). *HMP8170, HMP8171, HMP8172, HMP8173 – NTSC/PAL Video Encoder – Data Sheet*.

D. O. Hebb (1949). *The Organization of Behavior*. Wiley, New York.

R. Hecht-Nielsen (1990). *Neurocomputing*. Addison-Wesley, Reading, Massachusetts.

N. Hendrich (1996a). *Mustererkennung mit neuronalen Assoziativspeichernetzen*. Dissertation, Universität Hamburg, Fachbereich Informatik.

N. Hendrich (1996b). A Scalable Architecture for Binary Couplings Attractor Neural Networks. In *Proc. of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, MicroNeuro'96*, S. 213–220.

J. L. Hennessy, D. A. Patterson (1996). *Computer Architecture – A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., San Francisco, second edition Auflage.

J. Hertz, A. Krogh, R. G. Palmer (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, Reading, Massachusetts.

W. Hess (1993). *Digitale Filter*. Teubner-Verlag, Stuttgart, 2. Auflage.

Hewlett-Packard (1997). Testing Digital Video – The 1997 Digital Video Test Symposium, München. Hewlett-Packard.

M. Hoehfeld, S. E. Fahlman (1992). Learning with Limited Numerical Precision the Cascade-Correlation Algorithm. *IEEE Transactions on Neural Networks*, Band 3(1):S. 602–611.

M. Holler, S. Tam, H. Castro, R. Benson (1989). An Electrically Trainable Artificial Neural Network (ETANN) with 10240 Floating Gate Synapses. In *Proceedings of IEEE/INNS International Joint Conference on Neural Networks, IJCNN'89*, Band 2, S. 191–196.

P. W. Hollis, J. S. Harper, J. J. Paulos (1990). The Effect of Precision Constraints in a Backpropagation Learning Network. *Neural Computing*, Band 2(3):S. 363–373.

J. J. Hopfield (1982). Neural Networks and Physical Systems with Emergent Collective Computational Abilities. In *Proceedings of the National Academy of Sciences*, Band 79, S. 2554–2558.

J. J. Hopfield (1984). Neurons with Garded Response Have Collective Computational Properties Like those of Two-State Neurons. In *Proceedings of the National Academy of Sciences*, Band 81, S. 3088–3092.

IEEE (1994). IEEE Standard VHDL Language Reference Manual – VHDL Language Reference Manual, IEEE Standard 1076-1993. New York.

Intel Corporation (1990). *i486 Microprocessor – Programmer's Reference Manual*. Santa Clara, CA 95051, USA.

Intel Corporation (1995). *Pentium Processor Family Developer's Manual – Architecture and Programming Manual*. Santa Clara, CA 95052-8119, USA.

Intel Corporation (1997). *Intel Architecture Software Developer's Manual - Basic Architecture*. Santa Clara, CA 95052-8119, USA.

ISO/IEC Standard 10918-1 (JPEG). Digital Compression and Coding of Continuous-Tone Still Images.

K. Jack (1996). *Video Demystified: A Handbook for the Digital Engineer*. HighText, San Diego, CA, 2. Auflage.

B. Jähne (1993). *Digitale Bildverarbeitung*. Springer-Verlag, Heidelberg, 2. Auflage.

S. Jürgens (1996). Simulation rein digitaler Farbseparation (YUV, RGB) aus einem Videosignal im PAL-Standard zur Abschätzung des Implementationsaufwands. Studienarbeit, Universität Hamburg, Fachbereich Informatik.

S. Jürgens (1999). Systemnahe Simulation rein digitaler Verarbeitung analoger Farbvideosignaldaten (PAL). Diplomarbeit, Universität Hamburg, Fachbereich Informatik.

K. D. Kammeyer, K. Kroschel (1992). *Digitale Signalverarbeitung – Filterung und Spektralanalyse*. Teubner-Verlag, Stuttgart, 2. Auflage.

E. D. Karnin (1990). A Simple Procedure for Pruning Back-Propagation Trained Neural Networks. *IEEE Transactions on Neural Networks*, Band 1(1):S. 239–242.

M. Keating, P. Bricaud (1998). *Reuse Methodology Manual for System-On-A-Chip Designs*. Kluwer Academic Publishers.

L. Kirsch (1993). *Fernsehtechnik*. Vieweg, Braunschweig / Wiesbaden.

G. Kock, T. Becher (1997). MiND: An Environment for the Development, Integration, and Acceleration of Connectionist Systems. In *Proc. of 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Berlin, Germany, 24-29 August 1997*, A. Sydow, Ed., Band 6, S. 499–504. Wissenschaft und Technik Verlag, Berlin.

Kodak Photo CD^{TM} (1992). *Photo CD Access Software & Photo Sampler – Version 1.0 for Windows*. Rochester, New York, USA.

T. Kohonen (1972). Correlation Matrix Memories. *IEEE Transactions on Computers*, Band C-21:S. 353–359.

T. Kohonen (1996). New Developments and Applications of Self-Organizing Maps. In *Proc. of the 1996 IEEE International Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing, NICROSP'96, Venice, Italy, 21-23 August 1996*, S. 164–172. IEEE Computer Society Press, Los Alamitos, California.

R. Kolla, P. Molitor, G. Osthof (1989). *Einführung in den VLSI-Entwurf*. Teubner-Verlag, Stuttgart.

K. Köllmann, K.-R. Riemschneider, H. C. Zeidler (1996). On-Chip Backpropagation Training Using Parallel Stochastic Bit Streams. In *Proc. of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, MicroNeuro'96*, S. 149–156.

Y. Kondo, Y. Sawada (1992). Functional Abilities of a Stochastic Logic Neural Network. *IEEE Transactions on Neural Networks*, Band 3(1):S. 434–443.

A. König (1995). Survey and Current Status of Neural Network Hardware. In *Proc. of the Int. Conf. on Artificial Neural Networks, ICANN'95*, Band 1, S. 391–410. Paris.

B. Kosko (1992a). *Neural Networks and Fuzzy Systems – A Dynamical Systems Approach to Machine Intelligence*. Prentice-Hall International, London.

B. Kosko (1992b). *Neural Networks for Signal Processing*. Prentice-Hall International, London.

A. V. Krishnamoorthy, G. Yayla, S. C. Esener (1992). A Scalable Optoelectronic Neural System Using Free-Space Optical Interconnects. *IEEE Transactions on Neural Networks*, Band 3(1):S. 404–413.

S. Krol (1996). *Hardware-Realisierung eines neuronalen Netzwerkes zur Echtzeit – Bildklassifikation – NeNEB-Chip*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik.

T. Kropf (1998). *Introduction to Formal Hardware Verification*. Springer-Verlag.

H. T. Kung (1981). Use of VLSI in Algebraic Computation: Some Suggestions. In *Proc. 1981 ACM Symp. Symbolic and Algebraic Computation, ACM Sigsam*, S. 418–222.

P. Kurup, T. Abbasi (1995). *Logic Synthesis Using SYNOPSYS*. Kluwer Academic Publishers, Boston.

K. Lagemann (1987). *Rechnerstrukturen*. Springer-Verlag, Heidelberg.

J. A. Lansner, T. Lehmann (1993). An Analog CMOS Chip Set for Neural Networks with Arbitrary Topologies. *IEEE Transactions on Neural Networks*, Band 4(3):S. 441–444.

L. Larsson (1991). *Simulation neuronaler Netzwerke und Test des Z-Kammer-Triggers am H1-Detektor*. Interner Bericht DESY FH1T 92-03, Universität Hamburg, Fachbereich Physik. Diplomarbeit.

L. Larsson (1994). *Implementing the BPG Neural Network Algorithm on MP-1216 / MP-2216 Massive Parallel SIMD Machines, MasPar Challenge Project '94*. MasPar, Sunnyvale, CA, USA.

L. Larsson (1996). An EPLD Based Transient Recorder for Simulation of Video Signal Processing Devices in a VHDL Environment Close to System Level Conditions. In *Proc. of Sixth Int. Workshop on Field Programmable Logic and Applications, FPL'96, Darmstadt, Germany, 23-25 September 1996*, R. W. Hartenstein, M. Glesner, Eds., Band 1142 von *Lecture Notes in Computer Science*, S. 371–375. Springer-Verlag, Heidelberg.

L. Larsson (1997). Visualization of Backpropagation by Learn Trajectories to Explore Approximations for Hardware Implementations. In *Proc. of 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Berlin, Germany, 24-29 August 1997*, A. Sydow, Ed., Band 6, S. 517–522. Wissenschaft und Technik Verlag, Berlin.

L. Larsson (1999a). Learn Trajectories - A Systems Point of View Visualizing Method. In *Proc. of the Third World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and the Fifth International Conference on Information Systems Analysis and Synthesis (ISAS'99), Orlando, Florida, USA, July 31 - August 4, 1999*, Band 6, S. 97–104. International Institute of Informatics and Systemics (IIIS), Skokie, Illinois, USA.

L. Larsson (1999b). VHDL-Implementierung und FPGA-basierte Validierung eines digitalen PAL-Videosignal-Encoders. In *Tagungsband der DSP'99 Deutschland, München, 21.-23. September 1999*, H. Rogge, R. Ester, Eds., S. 315–324. Design & Elektronik, WEKA Fachzeitschriftenverlag, Poing, Deutschland.

L. Larsson, A. Klindworth, K. Lagemann, B. Schütz (1994). Teaching System Integration Using FPGAs. In *Proceedings of the Fifth Eurochip Workshop on VLSI Design Training, Dresden, Germany, 17-18-19 October 1994*, S. 194–199.

L. Larsson, S. Krol, K. Lagemann (1996). NeNEB – An Application Adjustable Single Chip Neural Network Processor for Mobile Real Time Image Processing. In *Proc. of the 1996 IEEE International Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing, NICROSP'96, Venice, Italy, 21-23 August 1996*, S. 154–162. IEEE Computer Society Press, Los Alamitos, California.

L. Larsson, S. Krol, K. Lagemann (1997a). NeNEB – Implementation of a Single Chip Neural Network Processor with Respect to System Level Constraints of Real Time Image Processing. *Journal of Microelectronic Systems Integration*, Band 5(1):S. 19–29.

L. Larsson, S. Krol, K. Lagemann (1997b). NeNEB – Neuronales Netzwerk zur Echtzeit - Klassifizierung von Bildern. In *Tagungsband 8. E.I.S.-Workshop, Entwurf Integrierter Schaltungen, Hamburg, Germany, 8-9 April 1997*, L. Peters, K. Lagemann, Eds., S. 57–66. GMD, St. Augustin.

L. Larsson, B. Mertsching, S. Schmalz (1997c). Interaktive Schaltungssimulation – Ein Beitrag zur zukunftsorientierten Grundstudiumsausbildung in der Technischen Informatik. In *Tagungsband 8. E.I.S.-Workshop, Entwurf Integrierter Schaltungen, Hamburg, Germany, 8-9 April 1997*, L. Peters, K. Lagemann, Eds., S. 239–248. GMD, St. Augustin.

Y. Le Cun (1985). Une Procédure d'Apprentissage pour Réseau à Seuil Assymétrique. In *Cognitiva 85: A la Frontière de I'Intelligence Artificelle des Sciences de la Connaissance des Neurosciences*, S. 599–604. Paris.

C. Lehmann, M. Viredaz, F. Blayo (1993). A Generic Systolic Array Building Block for Neural Network with On-Chip Learning. *IEEE Transactions on Neural Networks*, Band 4(3):S. 400–407.

O. Limann (1978). *Fernsehtechnik ohne Ballast*. Franzis, München, 12. Auflage.

C.-T. Lin, C. S. G. Lee (1996). *Neural Fuzzy Systems – A Neuro-Fuzzy Synergism to Intelligent Systems*. Prentice-Hall International, London.

J. B. Lont, W. Guggenbühl (1992). Analog CMOS Implementation of a Multilayer Perceptron with Nonlinear Synapses. *IEEE Transactions on Neural Networks*, Band 3(1):S. 457–465.

H. D. Lüke (1995). *Signalübertragung – Grundlagen der digitalen und analogen Nachrichtenübertragungssysteme*. Springer-Verlag, Heidelberg.

A. C. Luther (1997). *Principles of Digital Audio and Video*. ARTECH House, Inc., Norwood, Massachusetts, USA.

M. Maris, H. Fürth (1997). NNetView: A Real-World Neural Network Programming Environment. In *Proc. of 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Berlin, Germany, 24-29 August 1997*, A. Sydow, Ed., Band 4, S. 233–238. Wissenschaft und Technik Verlag, Berlin.

P. Marwedel (1993). *Synthese und Simulation von VLSI-Systemen*. Hansa Verlag, München.

P. Masa (1995). *NeuroClassifier : analog VLSI neural network for very high speed pattern classification*. Dissertation, University of Twente, Enschede, The Netherlands.

P. Masa, K. Hoen, H. Wallinga, L. Larsson, H.-J. Behrend, W. Zimmermann (1993). 20 Million Pattern per Second VLSI Neural Network Pattern Classifier. In *Proc. of the Int. Conf. on Artificial Neural Networks, ICANN'93*, S. 1058–1061. Heidelberg.

MasPar (1991). *MasPar Computer Corporation Data-Parallel Programming Guide*. Sunnyvale, CA.

MasPar (1993a). *MasPar Parallel Application Language (MPL) – Reference Manual*. Sunnyvale, CA, Software Version 3.2, Document Part # 9302-0101, Revision A4.

MasPar (1993b). *MasPar Parallel Application Language (MPL) – User Guide*. Sunnyvale, CA, Software Version 3.2, Document Part # 9302-0101, Revision A5.

L. W. Massengill, D. B. Mundie (1992). An Analog Neural Hardware Implementation Using Charge-Injection Multipliers and Neuron-Specific Gain Control. *IEEE Transactions on Neural Networks*, Band 3:S. 354–362.

R. Mäusel (1995). *Fernsehtechnik*. Hüthing Buch Verlag, Heidelberg, 2. Auflage.

J. L. McClelland, D. E. Rumelhart (1986). *Parallel Distributed Processing: Psychological and Biological Models*, Band 2. MIT Press, Cambridge, USA.

W. McCulloch, W. Pitts (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathathematical Biophysics*, Band 5:S. 115–133.

C. Mead (1989). *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, Massachusetts.

C. Mead, M. Ismail (1989). *Analog VLSI Implementation of Neural Systems*. Kluwer Academic Publishers, Boston.

C. A. Mead, X. Arreguit, J. Lazzaro (1991). Analog VLSI Model of Binaural Hearing. *IEEE Transactions on Neural Networks*, Band 2(1):S. 230–236.

J. L. Meador, A. Wu, C. Cole, N. Nintunze, P. Chintrakulchai (1991). Programmable Impulse Neural Circuits. *IEEE Transactions on Neural Networks*, Band 2:S. 101–109.

MicroNeuro'96 (1996). *MicroNeuro'96 – Proc. of the Fifth Int. Conf. on Microelectronics for Neural Networks and Fuzzy Systems, Lusanne, Switzerland, February 12-14, 1996*. IEEE Computer Society Press.

O. Mildenberger (1995). *System- und Signaltheorie*. Vieweg, Braunschweig / Wiesbaden, 3. Auflage.

M. Minsky, S. Papert (1969). *Perceptrons*. MIT Press. Third printing 1988.

P. D. Moerland, E. Fiesler (1996). Hardware-Friendly Learning Algorithms for Neural Networks: An Overview. In *Proc. of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, MicroNeuro'96*, S. 117–124. IEEE Computer Society Press, Los Alamitos, California.

G. Moon, M. E. Zaghloul, R. W. Newcomb (1992). VLSI Implementation of Synaptic Weighting and Summing in Pulse Coded Neural-Type Cells. *IEEE Transactions on Neural Networks*, Band 3(1):S. 394–403.

A. Moopenn, T. Duong, A. P. Thakoor (1989). Digital-Analog Hybrid Synapse Chip for Electronic Neural Networks. In *Advances in Neural Informations Processing Systems 2*.

G. E. Moore (1998). Cramming More Components onto Integrated Chips. *Proceedings of the IEEE – Special Issue: 50th Anniversary of the Transistor*, S. 82–85.

B. Morgenstern (1994). *Farbfernsehtechnik*. Teubner Studienskripte. Teubner-Verlag, Stuttgart, 4. Auflage.

U. A. Muller, B. Baumle, P. Kohler, A. Gunzinger, W. Guggenbuhl (1992). Achieving Supercomputer Performance for Neural Net Simulation with an Array of Digital Signal Processors. *IEEE Micro*.

A. F. Murray, D. D. Corso, L. Tarassenko (1991). Pulse-Stream VLSI Neural Networks Mixing Analog and Digital Techniques. *IEEE Transactions on Neural Networks*, Band 2(1):S. 193–204.

F. H. Netter (1987). *Nervensystem I: Neuroanatomie und Physiologie*. Georg Thieme Verlag.

D. Nguyen, B. Widrow (1989). The truck backer-upper: An example of self-learning in neural networks. In *Proc. of the Int. Joint Conf. on Neural Networks, IJCNN'89*, Band II, S. 357–363. IEEE Computer Society Press, Los Alamitos, California.

OKI Semiconductor (1998). *NTSC/PAL Digital Encoder – Data Sheet*.

A. R. Omondi (1994). *Computer Arithmetic Systems – Algorithms, Architectures and Implementations*. Prentice-Hall International, London.

D. B. Parker (1985). Learning Logic. Technical Report TR-47, Center of Computional Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA.

S. D. Pasquale, G. Maifredi, M. Temporin (1996). ISTRIA: An On-Line Neural Network System for the Analysis of Financial Markets. In *Neural Networks and their Applications*, J. G. Taylor, Ed., S. 133–142. John Wiley & Sons, New York.

D. W. Patterson (1996). *Artificial Neural Networks – Theory and Applications*. Prentice-Hall International, London.

R. Paul (1995). *Elektrotechnik und Elektronik für Informatiker - Band 2 Grundgebiete der Elektronik*. Teubner-Verlag.

D. Pellerin, D. Taylor (1997). *VHDL Made Easy!*. Prentice-Hall International, London.

Philips Semiconductors (1996a). *Digital Video Encoder (DENC2) – Data Sheet*.

Philips Semiconductors (1996b). *TDA8702 8-Bit Video Digital-to-Analog Converter – Data Sheet*.

Philips Semiconductors (1998). *SAA6750H – Encoder for MPEG2 image recording (EMPIRE) – Preliminary specification*. Eindhoven, The Netherlands.

P. J. Plauger (1992). *The Standard C Library*. Prentice-Hall International, London.

D. A. Pomerleau (1989). ALVINN: An Autonomous Land Vehicle in a Neural Network. Technischer Bericht, Carnegie-Mellon University.

D. A. Pomerleau (1993). *Neural Network Perception for Mobile Robot Guidance*. Kluwer Academic Publishers, Boston.

L. Pratt, S. Nicodemus (1994). Case Studies in the Use of a Hyperplane Animator for Neural Network Research. In *Proc. of IEEE Int. Conf. on Neural Networks, ICNN'94*, Band 1, S. 78–83. IEEE Computer Society Press, Los Alamitos, California.

W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling (1986). *Numerical Recipes – The Art of Scientific Computing*. Cambridge University Press.

W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling (1992). *Numerical Recipes in C – The Art of Scientific Computing*. Cambridge University Press, 2. Auflage.

L. R. Rabiner, T. P. J.H. McCellan (1975). FIR Digital Filter Design TEchniques Using Weighted Chebyshev Approximation. *IEEE Proc.*, Band 63:S. 595–610.

U. Ramacher, W. Raab, J. Anlauf, J. Beichter, U. Hachmann, N. Brüls, M. Weßeling, E. Sichender (1993). Multiprocessor and Memory Architecture of the Neurocomputer SYNAPSE-1. In *Proc. of the Int. Conf. on Artificial Neural Networks, ICANN'93*, S. 1034–1039. Springer-Verlag, Heidelberg.

U. Ramacher, W. Raab, J. Anlauf, U. Hachmann, M. Weßeling (1994). *SYNAPSE-1 – ein General-Purpose Neurocomputer*. Siemens-Nixdorf, Zentralabteilung Forschung und Entwicklung, München.

N. Ramakrishna, M. A. Bayoumi (1994). Sphinx: A High Level Synthesis System for ASIC Design. In *VLSI Design Methodologies for Digital Signal Processing Architectures*, The Kluwer International Series in Engineering and Computer Science, S. 1–34. Kluwer Academic Publishers, Boston.

F. J. Rammig (1989). *Systematischer Entwurf digitaler Systeme*. Teubner-Verlag, Stuttgart.

R. Rauscher (1996). *Entwurfsmethodik hochintegrierter anwendungsspezifischer digitaler Systeme*. Wissenschaftliche Schriften – Informatik. Pro Universitate Verlag, Sinzheim, 1. Auflage.

U. Reimers (1997). *Digitale Fernsehtechnik – Datenkompression und Übertragung für DVB*. Springer-Verlag, Heidelberg, 2. Auflage.

P. Rojas (1993). *Theorie der neuronalen Netzwerke – Eine systematische Einführung*. Springer-Verlag, Heidelberg.

P. Rojas (1994). The Fractal Geometry of Backpropagation. In *Proc. of IEEE Int. Conf. on Neural Networks, ICNN'94*, Band 1, S. 233–238. IEEE Computer Society Press, Los Alamitos, California.

F. Rosenblatt (1958). The Perceptron: A Probabilistic Model for Information Storage and Organisation in the Brain. *Psychological Review*, Band 65:S. 386–408.

A. Rost (1983). *Grundlagen der Elektronik*. Springer-Verlag.

D. E. Rumelhart, G. E. Hinton, R. J. Williams (1986a). Learning Internal Representations by Error Propagation. In *Neurocomputing: Foundations of Research*, J. A. Anderson, E. Rosenfeld, Eds., S. 673. MIT Press.

D. E. Rumelhart, G. E. Hinton, R. J. Williams (1986b). Learning representations by back propagating errors. In *Neurocomputing: Foundations of Research*, J. A. Anderson, E. Rosenfeld, Eds., S. 696. MIT Press, Cambridge, USA.

D. E. Rumelhart, G. E. Hinton, R. J. Williams (1986c). Learning representations by back propagating errors. *Nature*, Band 323:S. 533–536.

D. E. Rumelhart, J. L. McClelland (1986). *Parallel Distributed Processing: Foundations*, Band 1. MIT Press, Cambridge, USA.

S. Satyanarayana, Y. Tsividis, H. P. Graf (1989). A Reconfigurable Analog VLSI Neural Network Chip. In *Advances in Neural Informations Processing Systems 2*, S. 758–768.

S. Schiek, G. Schmidt (1995). Application of a High Speed Analog Neural Network Chip for First Level Triggering at the H1-Experiment at HERA. In *Proc. of the Int. Conf. on Artificial Neural Networks, ICANN'95*, Band 1, S. 363–368. Paris.

E. Schöneburg, Ed. (1993). *Industrielle Anwendungen Neuronaler Netzwerke – Fallbeispiele und Anwendungskonzepte*. Addison-Wesley, Reading, Massachusetts.

E. Schöneburg, N. Hansen, A. Gawelczyk (1990). *Neuronale Netzwerke*. Markt & Technik.

A. Singer (1990). Exploiting the Inherent Parallelism of Artificial Neural Networks to Achieve 1300 Million Interconnects per Second. *Parallel Computing*, Band 14:S. 305–315.

D. J. Smith (1998). *HDL Chip Design - A practical guide for designing, synthesizing and simulating ASICs and FPGAs using VHDL or Verilog*. Doone Publications.

Sony Semiconductor (1998). *CXD1910AQ – Digital Video Encoder – Data Sheet*.

Sony Semiconductor Company of America (1999). *Sony CXD1922Q – MPEG2 Video Encoder – Advance Information*. San Jose, California, USA.

M. Stevenson, R. Winter, B. Widrow (1990). Sensitivity of Feedforward Neural Networks to Weight Errors. *IEEE Transactions on Neural Networks*, Band 1(1):S. 71–80.

R. C. Stogdill (1999). Dealing with Obsolete Parts. *IEEE Design & Test of Computers*, S. 17–25.

A. S. Tannenbaum, J. Goodman, Eds. (1999). *Computerarchitektur – Strukturen, Konzepte, Grundlagen*. Prentice-Hall International, London, 4. Auflage.

J. G. Taylor, Ed. (1996). *Neural Networks and their Applications*. John Wiley & Sons, New York.

Texas Instruments (1998). *TVP6000C Data Manual NTSC/PAL Digital Video Encoder – Data Sheet*. Houston, Texas, USA.

Texas Instruments Inc. (1997). *TMS320C80 – Digital Signal Processor – Data Sheet*. Houston, Texas, USA.

U. Tietze, C. Schenk (1993). *Halbleiter-Schaltungstechnik*. Springer-Verlag, Heidelberg, 10. Auflage.

V. Tryba (1996). Neuro-ASIC for Low Cost Supervision of Water Pollution. In *Proc. of the 1996 IEEE Int. Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing, NICROSP'96*, S. 111–116. IEEE Computer Society Press, Los Alamitos, California.

M. Valle, D. D. Caviglia, Donzellini (1992). Design of an Analog CMOS Neural Chip. In *The Proceedings of the Third Eurochip Workshop on VLSI Design Training*, S. 173–178.

A. D. Vella, B. Hudson, P. E. Irving (1996). Estimating Helicopter Strain Using a Neural Network Approach. In *Neural Networks and their Applications*, J. G. Taylor, Ed., S. 35–49. John Wiley & Sons, New York.

D. C. von Grünigen (1993). *Digitale Signalverarbeitung: Grundlagen und Anwendungen – Beispiele und Übungen mit MATLAB*. AT-Verlag, Arau, Schweiz.

T. Watanabe, K. Kimura, M. Aoki, T. Sakata, K. Ito (1993). A Single 1.5-V Digital Chip for a 10^6 Synapse Neural Network. *IEEE Transactions on Neural Networks*, Band 4(3):S. 387–393.

T. Watanabe, Y. Sugiyama, T. Kondo, Y. Kitamura (1989). Neural Network Simulation on a Massive Parallel Cellular Array Processor: AAP-2. In *Proc. of the Int. Joint Conf. on Neural Networks*, Band 2, S. 155.

J. Wawrzynek, K. Asanovic', N. Morgan (1993). The Design of a Neuro-Microprocessor. *IEEE Transactions on Neural Networks*, Band 4(3):S. 394–399.

P. J. Webros (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Dissertation, Harvard University.

N. H. E. Weste, K. Eshraghian (1993). *Principles of CMOS VLSI Design – A Systems Perspective*. Addison-Wesley, 2. Auflage.

B. Widrow, M. E. Hoff (1960). Adaptive Switching Circuits. In *1960 IRE WESCON Convention Road*, Band Part 4, S. 96–104. New York.

T. Williams, C. Kelley, *et al.* (1993). *GNUPLOT-Program, Version 3.5, 1986-1993*.

T. W. Williams, Ed. (1986). *VLSI Testing*, Band 5 von *Advances in CAD for VLSI*. Elsevier Science Publishers B.V., Amsterdam, The Netherlands.

C. Wolff, G. Hartmann, H. Rahne (1999). Parallele Simulation großer pulscodierter neuronaler Netzwerke auf DSPs. In *Tagungsband der DSP'99 Deutschland, München, 21.-23. September 1999*, H. Rogge, R. Ester, Eds., S. 267–273. Design & Elektronik, WEKA Fachzeitschriftenverlag, Poing, Deutschland.

S. Wolfram (1997). *Das Mathematica-Buch: Mathematica Version 3 – Die offizielle Dokumentation*. Addison-Wesley, Reading, Massachusetts.

H. Wüst, K. Kasper, H. Reininger (1998). Hybrid Number Representation for the FPGA-Realization of a Versatile Neuro-Processor. In *Proc. of the Euromicro'98 Workshop for Computational Inteligence, Västeras, Sweden, 25-27 August 1998*. IEEE Computer Society Press, Los Alamitos, California.

Xilinx, Inc. (1999). *Xilinx – Virtex^{TM} Field Programmable Gate Arrays – Advance Product Specification, Version 1.6*. San Jose, California, USA.

A. Zell (1994). *Simulation Neuronaler Netzwerke*. Addison-Wesley, Reading, Massachusetts.

Y. Zhang, G. E. Hearn, P. Sen (1993). A Modified Learning Algorithm for Backpropagation Network. In *Proc. of the Int. Conf. on Artificial Neural Networks, ICANN'93, Amsterdam, The Netherlands, 13-16 September 1993*, S. 478.

Zilog (1981). *Microcomputer Components Data Book*. Cupertino, California, USA.

Danksagung

Herrn Prof. Dr.-Ing. Klaus Lagemann danke ich für die Betreuung der Arbeit und die besondere Aufgeschlossenheit für neue Ideen und die Förderung wissenschaftlicher Entfaltung durch Gewähren von Freiräumen, Ideen selbständig zu verfolgen und wissenschaftliche Publikationen auf nationalen und internationalen Konferenzen auch eigenverantwortlich vorzustellen.

Herrn Prof. Dr.-Ing. D. P. F. Möller danke ich für die Übernahme des Korreferats.

Ich danke Dipl.-Ing. Manfred Grove, Herrn Prof. Dr. Klaus von der Heide, Dr. Norman Hendrich, Dipl.-Inform. Georg Heßmann, Dipl.-Inform. Andre Klindworth, Dipl.-Inform. Andreas Mäder, Dipl.-Inform. Bernd Schütz und Heike Tewes für Hilfsbereitschaft, Unterstützung, wissenschaftliche Anregungen, fachliche Diskussionen und für die angenehme Arbeitsatmosphäre während der gemeinsamen Zeit am Arbeitsbereich ”Technische Grundlagen der Informatik”. Besonderer Dank gilt Dr. habil. Reinhard Rauscher.

Desweiteren möchte ich Dipl.-Inform. Lars Harald Hahn, Dipl.-Inform. Sven Jürgens und Dipl.-Inform. Stephan Krol danken, deren Arbeiten ich betreut habe und die so zum Erfolg dieser Arbeit beigetragen haben.

Ich möchte an dieser Stelle auch allen Fachbereichseinrichtungen danken, die zum Gelingen der vorliegenden Arbeit beigetragen haben. Besonderer Dank gilt dabei den Mitarbeiterinnen und Mitarbeitern der Rechenzentrums und der Bibliothek.

Meinen Freunden Dipl.-Inform. Christian Markus und Dr. Dirk Michelsen danke ich für die Hilfe, die zum Gelingen der Arbeit beigetragen hat.

Ich danke allen Freundinnen und Freunden für das Verständnis, daß ich in den letzten Monaten vor dem Abschluß der Arbeit nur wenig Zeit für die Pflege alter Freundschaften gehabt habe. Bettina Basler und Wiebke Weick danke ich für die Begleitung verschiedener Abschnitte.

Besonderer Dank gilt meiner Mutter für das Korrekturlesen der fast fertigen Dissertation unter großem Zeitdruck kurz vor Abgabe der Arbeit. Ich danke allen, die mir durch zügiges Korrekturlesen ermöglicht haben, diese Arbeit noch im vergangenen Millennium abzuschließen.