Diplomarbeit

zum Thema:

Integration von Signalprozessor-Hardware in ein Echtzeit-Betriebssystem

an der

Technischen Universität Dresden
Fakultät Informatik
Institut für Betriebssysteme, Datenbanken und Rechnernetze
Lehrstuhl Betriebssysteme

Eingereicht von: Robert Baumgartl
Geboren am: 06.09.1970
Geboren in: Mittweida/Sa.
Matrikel-Nr.: 1188091
Eingereicht am: 01.11.1995

Betreuender Hochschullehrer:
Prof. Dr. H.Härtig (TU Dresden)

Inhaltsverzeichnis

Verzeichnis der Abbildungen

Verzeichnis der Tabellen

Verzeichnis der verwendeten Abkürzungen

ALU	Arithmetic Logic Unit
BPU	Branch Prediction Unit (Einheit zur Sprungzielvorhersage)
CISC	Complex Instruction Set Computer
Codec	Coder/Decoder
CPU	Central Processing Unit
DCT	Discrete Cosine Transform (Diskrete Cosinustransformation)
DMA	Direct Memory Transfer
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
DSV	Digitale Signalverarbeitung
FAQ	Frequently Asked Questions
FDC	Floppy Disk Controller
FFT	Fast Fourier Transform (schnelle Fouriertransformation)
FIR	Finite Impulse Response (endliche Impulsantwort)
FPGA	Field Programmable Gate-Array
HI	Host Interface
IDCT	Inverse Discrete Cosine Transform
IEEE	Institute of Electrical and Electronic Engineers
JFIF	JPEG File Interchange Format
JPEG	Joint Photographic Expert Group
MIMD	Multiple Instruction Multiple Data
MIPS	Million Instructions Per Second
MOPS	Million Operations Per Second
MPEG	Motion Picture Expert Group
NSP	Native Signal Processing
PC	Personal Computer
PCI	Peripheral Connect Interface
PLD	Programmable Logic Device
PVRG	Portable Video Research Group
RAM	Random Access Memory
ROM	Read-Only Memory
SRAM	Static Random Access Memory
VDC	Video Display Controller
VLIW	Very Long Instruction Word

1 Einführung

Die Entwicklung der Schnittstelle zwischen Mensch und Computer hat in den letzten Jahren einen tiefgreifenden Wandel erfahren. Multimediale Darstellungen gestatten die Vermittlung selbst komplexer Sachzusammenhänge und prädestinieren damit den Rechner als idealen Vermittler von Lehrinhalten und Informationen. Die Integration von Texten, Stand- und Bewegtbildern sowie Audioinformationen in computerbasierte Dokumente stellt höchste Anforderungen an die Leistungsfähigkeit von Rechnersystemen und Betriebssystemsoftware. Besonders die Verarbeitung digitalisierter Video- und Audioinformationen erfolgt in steigendem Maße unter Echtzeitbedingungen, so daß echtzeitfähige Dienste in Betriebssystemen in Zukunft an Bedeutung gewinnen werden.

Eine der vielversprechendsten Entwicklungen der letzten Jahre in der Mikroprozessortechnik sind zweifelsohne die Digitalen Signalprozessoren. Es handelt sich hierbei um äußerst leistungsfähige Prozessoren, die mit einer Vielzahl innovativer Merkmale aufwarten können. Infolge der ausgeprägten Spezialisierung dieser Prozessoren auf die Algorithmen der Digitalen Signalverarbeitung fanden sie bisher jedoch nur in geringem Maße Zugang zu aktuellen Computerarchitekturen.

Das Ziel dieser Arbeit ist daher die Untersuchung der Frage, inwiefern Digitale Signalprozessoren in der Lage sind, signifikant zur Entlastung des Zentralprozessors eines Computers von rechenintensiven Aufgaben bei der Bearbeitung multimedialer Datenströme beizutragen. Zu diesem Zweck werden ausgewählte elementare Algorithmen sowie ein komplexes Applikationsbeispiel sowohl für eine Universal- als auch eine Signalprozessor-Architektur implementiert und verglichen. Weiterhin soll im Rahmen der komplexen Applikation die DSP-Hardware vollständig in das Betriebssystem VSTa integriert und Aussagen zur Leistungsfähigkeit der erarbeiteten Lösung getroffen werden. Darüber hinaus ist es wünschenswert, beliebige Algorithmen hinsichtlich ihrer Eignung für eine Implementierung auf Signalprozessoren beurteilen zu können. Die Erarbeitung entsprechender Bewertungskriterien ist daher ein weiteres Ziel der Untersuchungen.

Die vorliegende Arbeit gliedert sich in 6 Kapitel. Den einleitenden Worten im Kapitel 1 folgen eine Zusammenfassung wichtiger Merkmale und ein Überblick über die Entwicklung und das erreichte Technologieniveau der Digitalen Signalprozessoren in Kapitel 2 sowie eine kurze Diskussion zu deren Perspektiven. Kapitel 3 beinhaltet Erläuterungen zum logischen Konzept der Untersuchungen, diskutiert verschiedene Lösungsansätze und begründet die getroffenen Entscheidungen zur praktischen Realisierung des komplexen Applikationsbeispiels. Die Beschreibung der gesamten Implementation erfolgt in Kapitel 4. Das anschließende Kapitel 5 ist den Leistungsmessungen und der Diskussion der ermittelten Ergebnisse vorbehalten. Kapitel 6

faßt wesentliche Aussagen der Arbeit zusammen und gibt einen Ausblick auf mögliche Fortsetzungen des bearbeiteten Themenkreises.

An dieser Stelle möchte ich mich bei meinen Betreuern Herrn Prof.Dr.rer.nat. Hermann Härtig und Herrn Dipl.-Inf. Jörg Wittenberger für die gewährte Unterstützung und Hilfe herzlich bedanken.

2 Signalprozessoren

2.1 Entwicklung der Digitalen Signalverarbeitung

Angestoßen durch Erfordernisse der Radar- und Sonardatenauswertung in der Militärtechnik begann man etwa um 1950 herum, erste Konzepte der Digitalen Signalverarbeitung zu formulieren. Ziel war dabei, die der bis dahin ausschließlich eingesetzten Analogen Signalverarbeitung inherenten Nachteile zu umgehen:

- Toleranz analoger Bauelemente bedingt Abgleich der Schaltung vor Inbetriebnahme.
- Gleitende Verschlechterung von Parametern infolge Alterung der Bauelemente erfordert periodische Neukalibrierung.
- Empfindlichkeit gegen Störungen, z.B. Temperaturschwankungen
- verhältnismäßig geringe Ausfallsicherheit
- mangelnde Flexibilität

Erst später erkannte man, daß bestimmte Klassen von Signalverarbeitungssystemen ausschließlich digital realisierbar sind, z.B. Filter mit exakt linearem Phasenverlauf.

Die rasante Entwicklung der Mikroelektronik ab den 70er Jahren resultierte in einem ungeheuren Aufschwung der Digitalen Signalverarbeitung ([Heue90]). Mit der Schaffung leistungsfähiger Prozessoren war es nun möglich, die theoretisch bereits erarbeiteten Algorithmen in der Praxis einzusetzen. Umgekehrt förderte die Bereitstellung der entsprechenden Hardware auch die Entwicklung neuer Algorithmen. Gleichzeitig hielt die neue Technologie Einzug in die unterschiedlichsten Anwendungssphären.

Die Algorithmen der Signalverarbeitung erfordern für ihre optimale Umsetzung Prozessorarchitekturen, die von denen der Universalprozessoren beträchtlich differieren. Im Laufe der Zeit entstand daher eine Klasse von leistungsfähigen Mikroprozessoren, die speziell für die Verarbeitung von großen Datenmengen unter Echtzeitanforderungen entwickelt wurden, trotzdem aber frei programmierbar sind und damit prinzipiell für jede Datenverarbeitungsaufgabe genutzt werden könnten.

Heute sind Digitale Signalprozessoren ein fester Bestandteil der Technikausstattung in Industrie und Forschung genauso wie im privaten Haushalt. Die aktuelle Situation ist weiterhin gekennzeichnet durch ein Eindringen der DSPs in traditionelle Domänen der Universalprozes-

soren. Zugleich wird von deren Herstellern versucht, durch beschleunigte Innovationen diesem Trend entgegenzuwirken.

Zeitpunkt	Ereignis
Ende der 40er Jahre	Abtasttheorem und die z-Transformation ermöglichen die Analyse von Abtastsystemen.
50er Jahre	Diskret aufgebaute digitale Systeme zur Radardatenverarbeitung entstehen.
1958	erste Veröffentlichungen über Abtastsysteme
1960	Entwicklung eines Algorithmus zur schnellen Fouriertransformation durch Cooley und Tukey
1963	Arbeiten zum Entwurf digitaler Filter
1969	erstes Lehrbuch zur Digitalen Signalverarbeitung
1979	erster programmierbarer Signalprozessor (Intel i2920)
1986	erster DSP mit 16-Bit-Hardware-Multiplizierer (NEC µPD7720)

Tabelle 1: Entwicklung der Digitalen Signalverarbeitung

Tabelle 1 faßt einige Meilensteine der Entwicklung der Digitalen Signalverarbeitung zusammen.

Zu den Algorithmen der Digitalen Signalverarbeitung zählen u.a. die Digitalen Filter, die diskrete und die schnelle Fouriertransformation, sog. Compander (Verschlüsselung digitaler Signale zum Zwecke der optimalen Ausnutzung von Übertragungsmedien) und die Signalgeneratoren.

2.2 Architektur

2.2.1 Differenzen zu Universalprozessoren

Wie bereits gesagt, wurden DSPs mit dem Ziel entwickelt, Algorithmen der Digitalen Signalverarbeitung so effektiv wie möglich abzuarbeiten. Abbildung 1 zeigt das Blockdiagramm eines derartigen Systems ([Meye82]). Ein am Eingang anliegendes kontinuierliches Signal wird durch einen Analog/Digital-Wandler zyklisch abgetastet, quantisiert, in ein proportionales Digitalwort endlicher Länge überführt und dem Signalprozessor übermittelt. Dieser transformiert das Eingangswort mittels des gewählten Algorithmus in ein Ausgangswort und überträgt es an den folgenden Digital/Analog-Wandler, der das kontinuierliche Ausgangssignal restauriert. Gewöhnlich stimmen die Wandlungsfrequenzen überein, und die Arbeit des DSP erfolgt in Echtzeit.

Kennzeichend für diese Art der Datenverarbeitung sind die anfallenden großen Datenmengen und der im Vergleich dazu viel kleinere notwendige Speicherplatz für die Codierung des Ver-

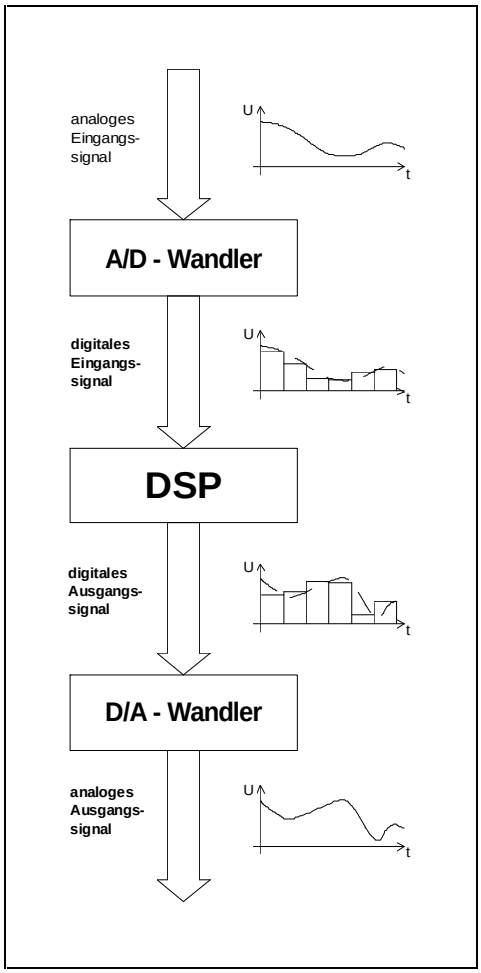

Abbildung 1: System zur Digitalen Signalverarbeitung

arbeitungsalgorithmus (Programm). Die einzelnen Datenworte besitzen stets einheitliche Länge, die von der erwünschten Quantisierungsgenauigkeit determiniert wird. Die maximale Zeitkomplexität des Algorithmus hängt vom technologischen Entwicklungsstand des DSP (Zykluszeit, mögliche Elementaroperationen) sowie der Abtastfrequenz der Wandler ab: Je schneller der DSP arbeitet und je niedriger die Abtastfrequenz ist, desto komplizierter darf der Verarbeitungsalgorithmus ausfallen.

Im Gegensatz zur Von-Neumann-Struktur der Universalprozessoren weisen DSPs im allgemeinen eine Harvard-Architektur auf (d.h. es existieren getrennte Speicherräume für Verarbeitungs- und Programmdaten), die teils nur innerhalb des Prozessors, teils auch für externen Speicher realisiert ist. Dabei erfolgt nicht nur eine Trennung von Programm- und Datenspeicher, sondern es sind bis zu 3 separate Datenräume verwirklicht. Dieses Merkmal erlaubt eine vollständige Parallelisierung der Zugriffe auf Programmcode und Daten.

Eine Eigenart der Algorithmen der Digitalen Signalverarbeitung ist das ständige Auftreten von kombinierten Additions- und Multiplikationsoperationen, die durch folgende Rechenvorschriften bedingt sind:

$$f(x) = \sum_{k=1}^{N} x_k m_k$$

Um diese Operationen so effektiv wie möglich zu implementieren, besitzen alle modernen DSPs eine dedizierte Baugruppe, die sog. MAC-Einheit, die eine Multiplikation zweier Werte, deren Addition zu einem weiteren Wert und das Zurückschreiben der Summe in einem Befehlszyklus ausführt (vgl. Abbildung 2). Die Operanden befinden sich dabei meist in Registern.

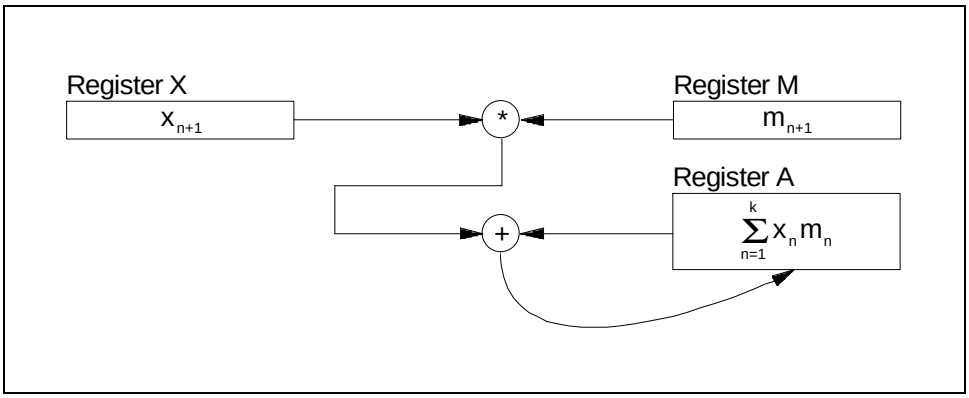

Abbildung 2:Arbeitsweise der MAC-Baugruppe

Die bei der Digitalisierung generierten Abtastwerte interpretiert man in der Signalverarbeitung gewöhnlich als gebrochene Werte im Intervall {-1, +1}. Festkomma-DSPs stellen daher ihre Daten im sog. „gebrochenen“ Zahlenformat (fractional data format) dar. Abbildung 3 verdeutlicht die Unterschiede zum gewöhnlich genutzten Festkomma-Zahlenformat bei gegebener Wortlänge *n*. Vorteil dieser Zahlendarstellung ist die Unmöglichkeit von Überläufen bei Multiplikationsoperationen, was entsprechende Tests unnötig macht. Sollen Festkomma-Werte in herkömmlicher Darstellung auf DSPs verarbeitet werden, so sind u.U. besondere Vorkehrungen nötig.

Zusätzlich zu den Festkomma-DSPs existieren ebenfalls Gleitkomma-Varianten, wobei sowohl reine Gleitkomma-Arithmetik als auch zur Integer-ALU parallel arbeitende Gleitkomma-Einheiten (analog zu numerischen Coprozessoren) vorkommen.

Weiterhin sind DSV-Algorithmen durch eine hohe Zahl zyklischer Befehlsausführungen mit fester Anzahl von Wiederholungen gekennzeichnet. Folgerichtig verfügen DSPs über Spezialregister und -befehle zur Organisation von Schleifen mit fester Iterationszahl, die (abzüglich eines initialen Verwaltungsaufwands) ein zu linearem Code identisches Laufzeitverhalten von

Softwareschleifen garantieren. Dem gegenüber weisen DSV-Algorithmen einen sehr geringen Anteil bedingter Sprünge auf, so daß diese in Architektur und Befehlssatz verhältnismäßig schlecht unterstützt sind.

Echtzeit-Applikationen erfordern meist eine kleine Interruptlatenz. Generelles Merkmal aller Maschinenbefehle eines DSP ist daher eine kurze Zykluszeit. Komplexe Maschinenoperationen wie die Division werden dabei entweder partitioniert, indem z.B. ein DIV-Befehl pro Befehls-

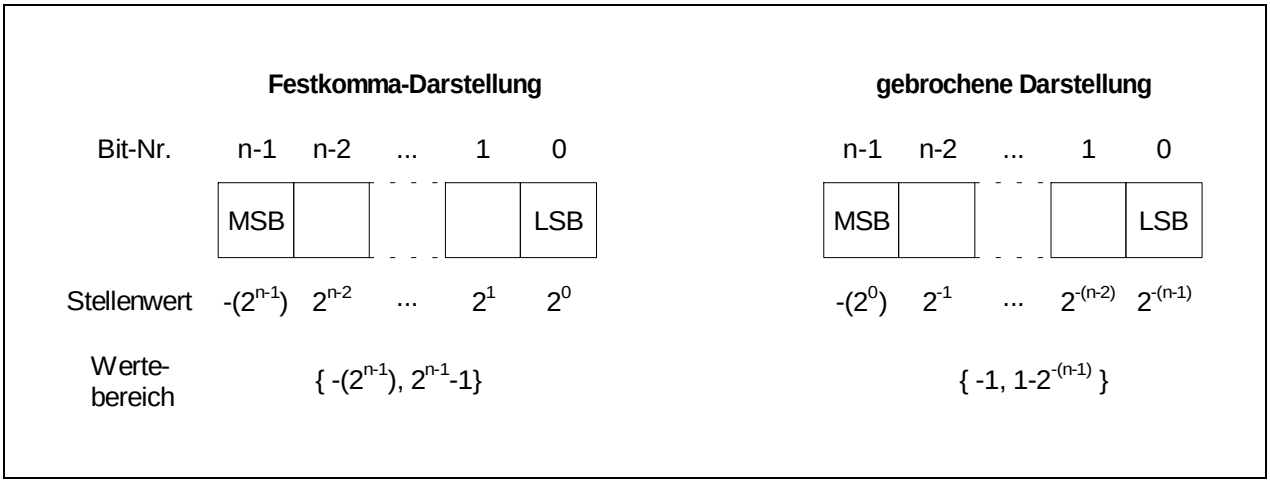

Abbildung 3: Vergleich von Festkomma- und gebrochenem Zahlenformat

zyklus nur 1 Quotienten-Bitstelle errechnet, also zyklisch über Wortoperanden ausgeführt werden muß, oder die Operation wird durch spezielle Hardware unterstützt (vgl. MAC-Baugruppe).

In der Vergangenheit wurden DSPs fast ausschließlich im Standalone-Betrieb innerhalb von industriellen Steuerungs- und Meßapplikationen genutzt. Auch in der Gegenwart ist diese Einsatzform vorherrschend, wenn auch zunehmend andersgeartete Konzepte für die Verwendung dieser Prozessoren entstehen. DSPs besitzen daher ausgeprägte Mikrocontroller-Fähigkeiten. Sie sind mit minimaler Beschaltung einsatzfähig, können z.B. aufgrund integrierten Speichers vielfach ohne externen Speicher betrieben werden. Eine Vielzahl flexibler Schnittstellen ermöglichen den Datenaustausch mit Sensoren oder weiteren Prozessoren, was gleichzeitig den kostengünstigen Aufbau von Multiprozessorsystemen ermöglicht. Vorreiter bei dieser Entwicklung ist der DSP TI320C40 der Firma Texas Instruments, der über 6 bidirektionale serielle Schnittstellen verfügt, die direkt dem Transputerkonzept entlehnt sind. Die in Signalverarbeitungssystemen obligatorischen D/A- und A/D-Wandler können meist ohne zusätzliche Bausteine direkt angeschlossen werden bzw. sind ebenfalls auf dem Chip integriert. Ein oder mehrere frei programmierbare Zeitgeberbausteine gestatten z.B. die Generierung von Abtastraten oder die periodische Abfrage von peripheren Einheiten. Moderne DSPs verfügen darüber

hinaus über Schnittstellen und Funktionseinheiten zum Debugging, so daß teure In-Circuit-Emulatoren überflüssig werden.

Um die Arbeitsgeschwindigkeit der DSPs voll nutzen zu können, werden diese im allgemeinen mit statischem Speicher betrieben. Der Vorteil der entfallenden Waitstates wird durch einen hohen Preis und eine beschränkte Kapazität des Systemspeichers erkauft. Letzteres stellt keinen Widerspruch zur Aussage, der DSP verarbeite große Datenmengen, dar, da vom gesamten Datenaufkommen einer DSV-Applikationen zu einem festen Zeitpunkt stets nur ein verhältnismäßig kleiner Teil benötigt wird. So sind je 64 KWorte für Programm- und Datenspeicher für den allergrößten Teil der Applikationen mehr als ausreichend, was den Adreßbus auf 16 Bit Breite begrenzt. Es sind ebenfalls DSPs mit großen Adreßräumen und Möglichkeiten zum Anschluß dynamischen Speichers verfügbar.

In vielen Fällen erzeugen Algorithmen der Signalverarbeitung zu einem festen Zeitpunkt aus den *n* letzten bis dahin empfangenen Eingangsdatenworten ein Ausgangsdatenwort. Für die effiziente Speicherung der Eingangsdaten bieten sich Ringpuffer an, für deren Verwaltung der DSP entsprechende Adressierungsarten zur Verfügung stellt. Jeder moderne DSP verfügt des weiteren über eine spezielle Adressierungsart für die Operanden der schnellen Fouriertransformation (FFT) sowie über mehrere weitere Adressierungsmöglichkeiten zum flexiblen Zugriff auf Datenworte.

Die elementare Einheit der Informationsverarbeitung ist bei DSPs stets das Wort mit einer Länge von mindestens 16 Bit. Dieser Umstand in Verbindung mit der Vielzahl der in einem Befehlszyklus parallel möglichen Aktionen bedingt die Kodierung der Maschinenbefehle nach einem VLIW-ähnlichen (Very Long Instruction Word) Prinzip.

Die exakte Bestimmung des Zeitbedarfs von Algorithmen ist unter Echtzeitbedingungen von enormer Wichtigkeit. Funktionseinheiten, die durch ihre Komplexität diese Bestimmung erschweren, finden nur schleppend den Zugang zur DSP-Architektur. So gehören Caches oder Einheiten zur Sprungzielvorhersage (Branch Prediction Unit, BPU), bei Universalprozessoren der aktuellen Generation längst integriert, keinesfalls zur Standardausstattung von DSPs.

Der Maschinenbefehlssatz und die Registerstruktur von DSPs sind stark auf die Erfordernisse der Digitalen Signalverarbeitung zugeschnitten, was sich erschwerend auf die Implementierung von allgemeinen Algorithmen der Informationsverarbeitung auswirkt. Jedoch differieren die individuellen DSP-Architekturen sehr stark, so daß allgemeingültige Grundsätze schwer zu formulieren sind.

2.2.2 Leistungskennwerte aktueller DSPs

Dieser Abschnitt soll einen Überblick über das gegenwärtige Angebot von Signalprozessoren geben. Der Variantenreichtum und die Ausstattungsvielfalt der existenten Typen ist hierbei generell viel größer als bei Universalprozessoren und kann am ehesten mit dem Markt für Microcontroller verglichen werden.

In industrierelevanten Applikationen der Digitalen Signalverarbeitung wird im Gegensatz zu Personalcomputern und Workstations zumeist derjenige Prozessortyp ausgewählt, dessen Verarbeitungsleistung gerade noch ausreicht. Daher ist für einen modernen DSP keinesfalls eine ausgesprochen hohe numerische Verarbeitungsleistung kennzeichnend, sondern vielmehr ein optimales Preis/Leistungs-Verhältnis.

Der Markt wird zum gegenwärtigen Zeitpunkt von 4 amerikanischen Herstellern dominiert: Analog Devices, AT&T, Motorola und Texas Instruments, wobei letzterer einen geringen Vorteil gegenüber den Konkurrenten besitzt und damit den Marktführer stellt. Des weiteren existieren eine große Menge kleinerer Anbieter, deren gesamter Umsatz 10% des Marktvolumens nicht übersteigt, wie NEC, National Semiconductor, Philips u.a.

Grob kann man die Typenvielfalt in 4 Kategorien einteilen:

I. Billigste Prozessoren für den Einsatz in Geräten der Telekommunikation

Diese Gruppe zeichnet sich durch Festkomma-Darstellung, eine maximale Verarbeitungsbreite von 16 Bit und eine moderate Leistung aus. Da diese Prozessoren fast immer im standalone-Betrieb eingesetzt werden, verfügen sie über microcontroller-typische Funktionseinheiten, wie Zeitgeber, Watchdogs, integriertes RAM usw. Die Taktfrequenz beträgt etwa 10 bis 30 MHz. Einige Typen besitzen A/D- und D/A-Wandler mit einer für Telekommunikationsapplikationen ausreichenden Breite von 12-14 Bit integriert. Entwickelt für den massenhaften Einsatz in Geräten der Telekommunikation, beträgt der Preis etwa 10 bis 20 Dollar pro Exemplar.

II. Festkomma-DSPs größerer Leistungsfähigkeit

Im Vergleich zur Kategorie I sind die Vertreter dieser Gruppe deutlich leistungsfähiger. Sie werden schneller getaktet (ca. 30 bis 80 MHz) und besitzen eine Verarbeitungsbreite von 16 bis 24 Bit, wobei die Register oft die doppelte Größe aufweisen. Der Adreßraum umfaßt typisch ein oder mehrere Speicher von je 64 KWorten. Die Prozessoren besitzen mehrere Schnittstellen sowie Speicher größerer Kapazität integriert. Vorrangige Einsatzgebiete sind Festkomma-Applikationen mit notwendiger hoher Quantisierungsgenauigkeit, wie die professionelle Audio- und Videodatenverarbeitung. Die Preise für Prozessoren dieser Gruppe betragen etwa 30 bis 50 Dollar. Der für die vorliegende Arbeit genutzte DSP56001 von Motorola ist dieser Kategorie zuzurechnen.

III. DSPs mit Gleitkommaarithmetik

Gegenüber den Festkomma-DSPs verfügen diese Prozessoren über den Vorteil höherer Genauigkeit durch eine mindestens 32 Bit breite Gleitkomma-Zahlendarstellung, die jedoch nicht immer dem Standard IEEE 754 entspricht. Meist beträgt die Adreßbusbreite ebenfalls 32 Bit, so daß der Adreßraum 4 GWorte umfaßt. Moderne Typen erlauben die wahlweise Nutzung von Fest- oder Gleitkommadarstellung. Die Peripherieausstattung ist mit der der Kategorie II zu vergleichen. Gleitkomma-DSPs werden z.B. in hochauflösenden Meßgeräten oder als Coprozessoren eingesetzt. Durch den bedeutend höheren Aufwand für ALU, Hardware-Multiplizierer usw. sowie die niedrigeren Stückzahlen ist der Preis dieser Prozessoren durchweg höher als bei den Festkomma-DSPs der Kategorie II, er liegt bei etwa 50 bis 100 Dollar.

IV. High-End-Signalprozessoren für den Einsatz in Multiprozessorsystemen

Die Repräsentanten dieser Kategorie (momentan existieren 2 Typenreihen) sind die leistungsfähigsten, aber auch teuersten DSPs auf dem Markt. Sie verfügen über die gleichen grundlegenden Architekturmerkmale wie die Prozessoren der Kategorie III, sind jedoch besser ausgestattet und werden schneller getaktet. Hervorstechendes Merkmal ist die Unterstützung von Parallelverarbeitung in Befehlssatz wie Hardware. So sind alle Prozessoren mit jeweils 6 Schnittstellen zur Interprozessorkommunikation ausgerüstet. Des weiteren verfügen sie über integrierten Cache und mehrkanalige DMA-Controller. Diese Ausstattung, verbunden mit der ausgesprochen hohen numerischen Verarbeitungsleistung, determiniert den Einsatz in Parallelrechnern zur Abarbeitung rechenintensivster wissenschaftlicher Applikationen. Der Preis eines solchen Prozessors beträgt etwa 150 bis 400 Dollar.

Eine kontinuierliche Weiterentwicklung in kurzen Abständen sorgt für eine ständige Steigerung der Leistungsfähigkeit von DSPs. Dabei sind quantitative Verbesserungen, wie Erhöhung der Kapazität des integrierten Speichers, Optimierung und Erweiterung des Befehlssatzes und Beschleunigung der Taktfrequenz, vorherrschend. Bei dieser Evolution wird innerhalb der Typenreihen eines Herstellers gewöhnlich auf Kompatibilität geachtet.

Den momentan erreichten Leistungsstand soll der 1994 vorgestellte und 1995 verfügbare Typ ADSP-2106 „SHARC“ des Herstellers Analog Devices illustrieren, der der Kategorie IV zuzurechnen ist ([AD95]).

Dieser für Hochleistungsapplikationen konzipierte Prozessor besitzt die folgenden architekturellen Merkmale:

- 32 Bit Verarbeitungsbreite
- Wertepräsentation im Gleitkommaformat nach IEEE 754

- erweiterte Harvard-Architektur mit 3 separaten Speicherräumen
- Adreßräume von je 4 GWorten
- 2 unabhängige Rechenwerke zur Adreßgenerierung
- dreistufige Prozessorpipeline

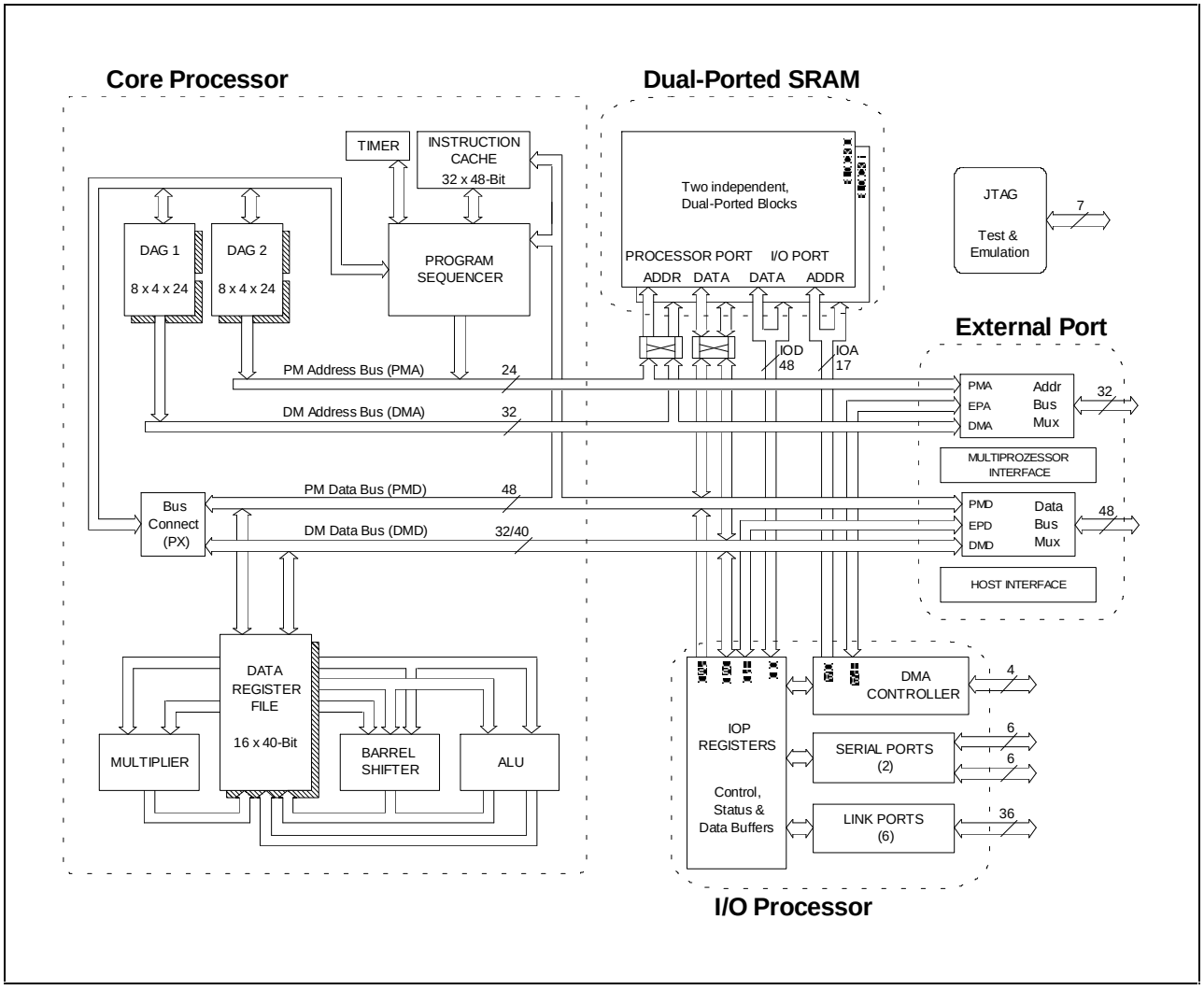

Abbildung 4: Blockdiagramm des ADSP-21060 SHARC

- 32 Worte umfassender Befehlscache
- 2 bzw. 4 MBit statisches RAM mit Zweifachzugriff (dual-ported)
- 6 Linkschnittstellen von 4 Bit Breite mit jeweils 40 MByte/s maximaler Transferrate
- 2 synchrone serielle Schnittstellen mit ebenfalls max. 40 Mbyte/s Transferrate
- zehnkanaliger DMA-Controller

Da bisher keine exakten Leistungsmessungen veröffentlicht wurden, sind quantitative Aussagen hierzu nicht möglich. Der Hersteller beziffert die Prozessorleistung auf 120 MFLOPS. Im Januar 1995 betrug der Preis für ein Exemplar des DSPs in der 2-MBit-Variante 195 Dollar (bei Abnahme von 1000 Stück). Zum gleichen Zeitpunkt kostete ein Intel Pentium-Prozessor zwischen 935 Dollar (100 Mhz Taktfrequenz) und 383 Dollar (60 MHz).

2.2.3 Der DSP Motorola 56001

Für den praktischen Teil der Arbeit wurde der verhältnismäßig preiswerte DSP56001 des Herstellers Motorola ausgewählt.

Es handelt sich hierbei um einen Festkommaprozessor mit vollständiger Harvard-Architektur. Die Datenbusbreite beträgt 24 Bit, die Adreßbusbreite 16 Bit. Es existieren damit Adreßräume von jeweils 64 KWorten, die für 1 Programm- und 2 Datenspeicher zur Verfügung stehen (Abbildung 5).

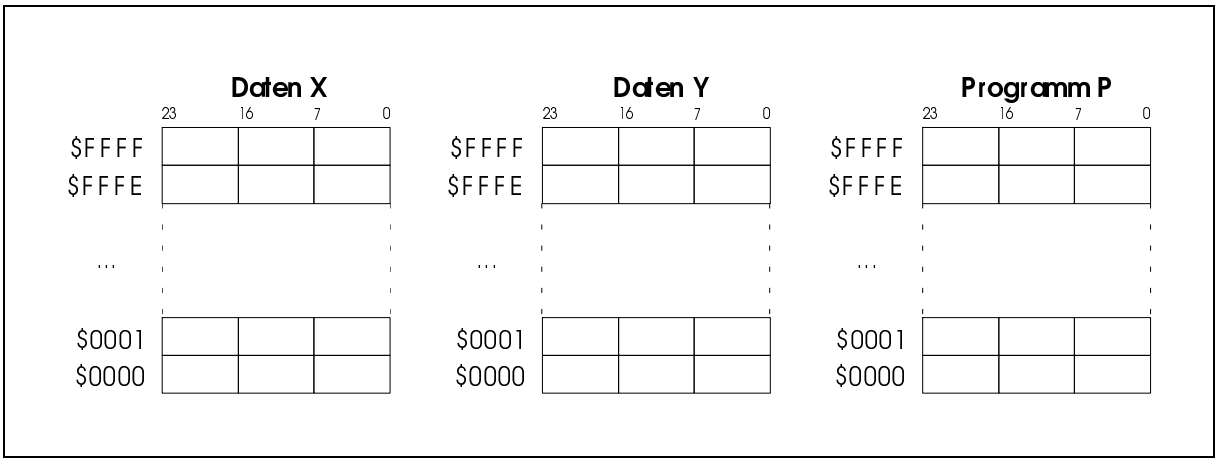

Abbildung 5: Speicherorganisation des DSP 56001

Für einen vollständigen Speicherausbau benötigt man also Bausteine mit 3*3*64k = 576kByte Kapazität.

Die Registerstruktur des DSPs zeigt Abbildung 6. Grundsätzlich unterscheidet man Register für die Ausführung arithmetischer Operationen (Akkumulatoren), Register für die Bereitstellung von Operanden (Eingangsregister), Register zur Generierung von Adressen (Adreßregister) sowie Register zur Programmsteuerung. Die Akkumulatoren sind jeweils 56 Bit breit, um 24x24-Bit-Multiplikationen ohne Überlauf iterieren zu können. Alle Registerteile sind separat ansprechbar.

Der Stack des Prozessors zählt ebenfalls zum Kern, da er aus Geschwindigkeitsgründen in Hardware realisiert ist (15x32 Bit). Reicht die Kapazität nicht aus, so ist eine Erweiterung im externen RAM möglich.

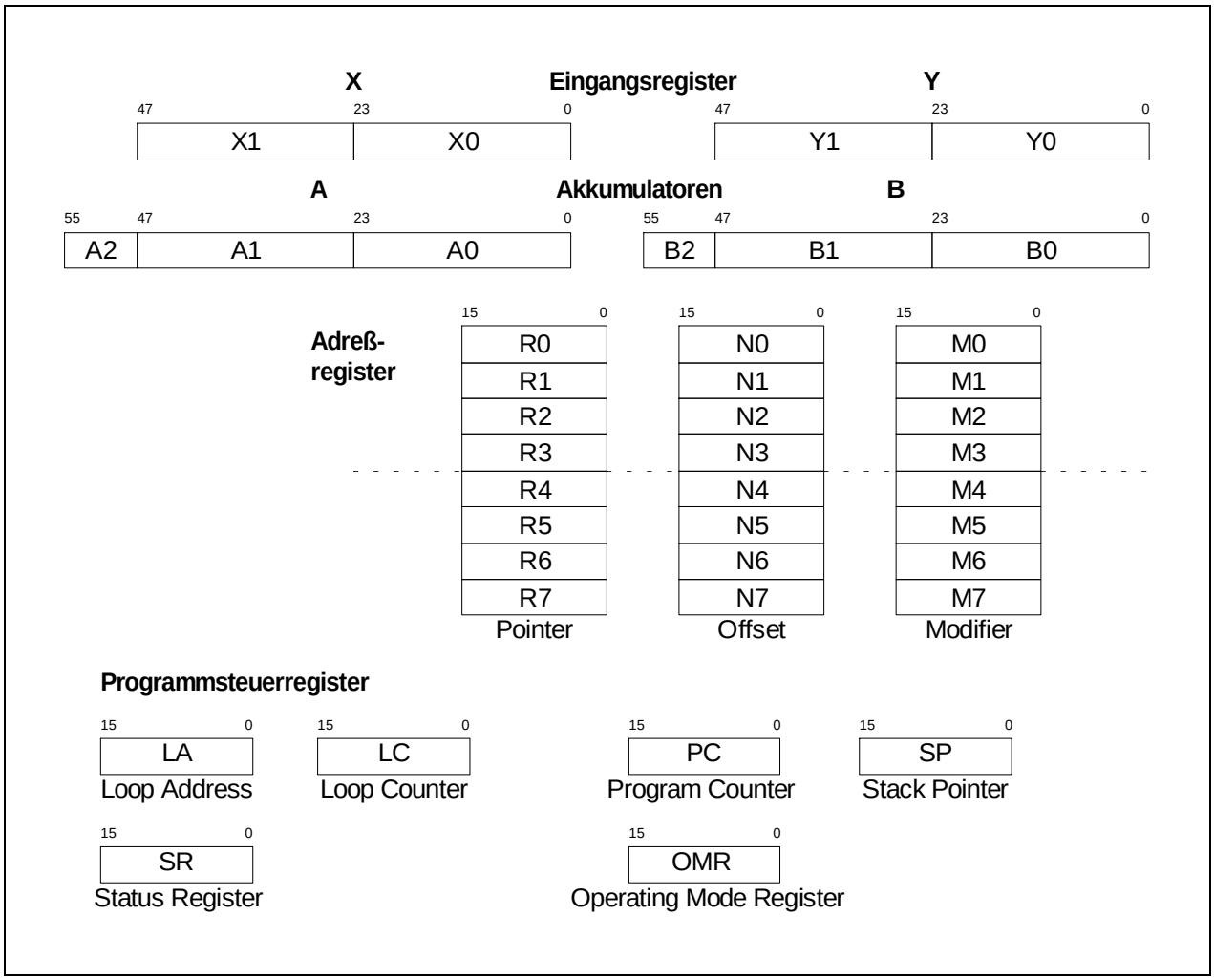

Abbildung 6: Programmiermodell des DSP56001

Zu den weiteren charakteristischen Merkmalen des Prozessors zählen:

- Adreßerzeugungseinheit getrennt von ALU
- dreistufige Befehlspipeline, für Programmierer nicht transparent
- 2 serielle Schnittstellen
- 1 parallele Schnittstelle (8 Bit breit) zum Anschluß von steuernden Prozessoren
- Wertepräsentation im gebrochenen Zahlenformat

- Befehlssatz umfaßt 62 Operationen
- hardwareunterstützte Iterationsbefehle
- 512 Worte Programm-RAM und 2*256 Worte Daten-RAM im Prozessor integriert
- Adressierungsarten beinhalten u.a. modulo-n- und Bit-reversed-Adressierung

Ausführlichere Aussagen zu den technischen Parametern des Signalprozessors sind in [Moto88A] zu finden, während [Moto90] detailliert das Programmiermodell beschreibt.

Der DSP56001 ist der erste Typ einer Reihe von sogenannten modularen DSPs mit identischem Prozessorkern, aber differierender Peripherieausstattung, die für diverse Zielapplikationsfelder zugeschnitten sind (Tabelle 2). Im Jahre 1988 wurde der DSP56001 zusammen mit seiner ROM-Variante DSP56000 am Markt eingeführt, beide Typen sind mittlerweile nicht mehr erhältlich. Statt dessen führte eine konsequente Weiterentwicklung des Konzepts zum Nachfolgetyp DSP56002, der 1992 präsentiert wurde. Zahlreiche Verbesserungen betreffen den Befehlssatz, die Funktionalität der Peripherieeinheiten und die zusätzliche Integration eines Ports zum on-chip-Debugging. Alle folgenden Typen der 56000-Reihe basieren auf dem DSP56002.

Typ	Charakteristik	Einsatzgebiet	Einführungsjahr
56000[1]	ROM-Variante des 56001 (maskenprogrammierbar)	industrielle Steuerungen	1988
56001[1]	Basistyp der Reihe	allgemeine DSV	1988
56002	verbesserte Variante des 56001, neuer Basistyp Entwicklung	allgemeine DSV	1992
56004	zusätzliche Interfaces zum Audioanschluß, integrierte Wandler	kostengünstige Audioapplikationen	1993
56005	zus. PWM-Einheit, mehr Speicher	industrielle Steuerungen	1995
56007	zusätzlicher Speicher	anspruchsvolle Audio-Applikationen	1995

Tabelle 2: DSP-Typenreihe 56000 von Motorola

Innerhalb der Reihe herrscht weitgehend Codekompatibilität, da alle Prozessoren auf dem gleichen Kern basieren. Von den einzelnen Typen existieren wiederum Subvarianten („Steppings“), die sich in bestimmten Details unterscheiden.

[1] Typ wird seit 1994 nicht mehr produziert.

2.3 Einsatzmöglichkeiten Digitaler Signalprozessoren

Eine Einteilung aller Applikationen von DSPs in Kategorien muß willkürlich erfolgen, die Grenzen verlaufen hierbei oft fließend. Charakteristisch ist, daß keine typenspezifischen Einsatzfelder existieren, d.h. ein bestimmter DSP-Typ kann prinzipiell in jeder Kategorie vertreten sein.

2.3.1 Nutzung als Eingebetteter Prozessor

Ursprünglich wurden DSPs ausschließlich für rechenintensive Applikationen in hochwertigen elektrotechnischen Geräten verwendet. Auch heute nutzt man die überwiegende Mehrzahl aller produzierten DSPs in Applikationen dieses Typs.

Während in früheren Jahren meist ein zusätzlicher Standard-Microcontroller Steuerungs- und Überwachungsfunktionen übernahm, ist infolge der gestiegenen Leistungsfähigkeit der DSPs gegenwärtig ein Trend zu Ein-Prozessor-Lösungen zu verzeichnen, der DSP übernimmt zusätzlich zu den Algorithmen der Digitalen Signalverarbeitung alle anfallenden Steuerungsaufgaben.

Folgende Merkmale zeichen Applikationen dieser Kategorie aus:

- Der Prozessor arbeitet mit minimaler Beschaltung, aufgrund der Kürze des Programms kann häufig auf externen Speicher verzichtet werden.
- Aus Gründen der Kostenminimierung wählt man denjenigen DSP-Typ aus, der die gestellten Anforderungen gerade noch erfüllen kann. Leistungsreserven verbleiben nicht.
- Es ist kaum Flexibilität seitens des DSP nötig, während der gesamten Lebensdauer des Gerätes wird ein- und derselbe Algorithmus abgearbeitet.
- Die Applikation setzt direkt auf die Hardware auf, betriebssystemähnliche Software zur Verwaltung von Ressourcen wird nicht benutzt.
- Die Erarbeitung der DSP-Software macht nur einen kleinen Anteil am gesamten Entwicklungsaufwand für das betreffende Gerät aus.
- Die Software-Komplexität ist relativ gering. Für Standardaufgaben existieren umfangreiche Applikationssammlungen und Software-Bibliotheken.
- Die Programmierung des Prozessors erfolgt zumeist in Assemblersprache, um das Leistungspotential voll zu nutzen.

– Es kommen für die Software-Entwicklung In-Circuit-Emulatoren zum Einsatz, die anstelle des Prozessors in die Schaltung eingefügt werden und über einen Hostrechner eine komfortable Oberfläche zur Programmierung bieten. Nachdem die Funktionsfähigkeit der Software erwiesen ist, wird der Emulator gegen den realen Prozessor ausgetauscht.

Anwendungsgebiet	Beispiele
Meßgeräte	Spectrum Analyzer, Signalgeneratoren, Tomographen
Telekommunikation	Leitungskodierung, Sprachverarbeitung
Funktechnik	Modulatoren, Demodulatoren, Satellitenempfänger
Audiotechnik	Musikinstrumente, Klangprozessoren (z.B. Dolby Surround, Equalizer, Rauschunterdrückung), Sprachsynthese
Videotechnik	Bildbearbeitung, Schärferegulierung, MPEG-Codec

Tabelle 3: Anwendungsbeispiele für DSPs in Eingebetteten Systemen

Tabelle 3 stellt einige Beispiele des DSP-Einsatzes für diese Kategorie zusammen.

2.3.2 DSPs in Computersystemen

Anfang der 90er Jahre begann man, Signalprozessoren in konventionellen Mikroprozessorumgebungen, also Personalcomputern oder Workstations, zu nutzen. Überwiegendes Einsatzziel stellte die Verarbeitung von Audiodaten dar, die bis dato eher schlecht unterstützt wurde bzw. teure Zusatzhardware erforderte, da D/A- und A/D-Wandlung hohe Echtzeitanforderungen stellen. Signalprozessoren in Verbindung mit geeigneter Wandlerhardware versprachen eine preiswerte Alternative.

Computer	DSP
ATARI Falcon 030	Motorola DSP56001
Silicon Graphics Iris Indigo	Motorola DSP56001
NextCube	Motorola DSP56001
Apple Centris 660 AV	AT&T 3210
Apple Quadra 840 AV	AT&T 3210

Tabelle 4: Computertypen mit integrierten DSPs

Es entwickelten sich zwei unterschiedliche Konzepte. Zum einen versuchte man, DSPs über das Bussystem eng an die CPU eines betreffenden Computers zu koppeln und damit eine universelle Nutzung des DSP zu ermöglichen. Beispiele für diese Architektur sind der Atari Falcon 030 und der NeXTCube, die beide einen DSP56001 von Motorola besitzen. Die verfügbaren Applikationen umfassen u.a. Musik-Synthese, Audio-Meßgeräte, Sprachverarbeitung,

Klangforschung oder die vollständige Realisierung eines Modems in Software. Tabelle 4 listet einige Vertreter dieses Konzepts mit den zugehörigen DSP-Typen auf.

Die Architektur des IBM PC/AT und seiner Nachfolger erlaubte eine solche Integration von DSPs nicht ohne weiteres. Daher verfolgte man hier eine andere Strategie: DSP-Hardware wurde als relativ autonomes System auf Einsteckkarten realisiert, die über Portadressen mit dem Hostrechner kommunizieren. Das IDEAL56-System, welches für diese Arbeit genutzt wurde, ist dieser Kategorie zuzurechnen. Nachteilig ist die fehlende Standardisierung der Einsteckkarten, so daß Software stets speziell für einen Kartentyp entwickelt werden muß.

Beiden Konzepten gemein sind folgende Merkmale:

- Der DSP besitzt privates RAM und ist meist in ein Subsystem zur Audiodaten-Ein- und -Ausgabe eingebettet.
- Die Programmierung des DSP erfolgt über den Hostrechner und ist je nach Umfang der Werkzeuge für die jeweilige Plattform mehr oder minder komfortabel möglich.
- Zum Einsatz kommen DSP-Typen, die universell nutzbar sind und eine gute Unterstützung von allgemeinen Algorithmen erwarten lassen.
- Der DSP kann frei genutzt werden, die Unterstützung durch das Host-Betriebssystem variiert allerdings.
- Auf dem DSP wird kein spezialisiertes Betriebssystem zur Ressourcenverwaltung genutzt, das Betriebssystem des Hosts wechselt einfach die Applikationen aus.
- Schwachpunkt beider Systeme ist die Anbindung des lokalen, in der Regel kleinen, DSP-Speichers an das RAM des Host-Prozessors, die Datenübertragung zwischen beiden Prozessoren ist häufig langsam. Auswege könnten hier ein großes, aber teures lokales RAM des DSP, die Übertragung per DMA oder die Kommunikation über dual-ported RAM sein.

Des weiteren werden zunehmend PC-Einsteckkarten zur Audiodatengenerierung entwickelt, die DSPs als hochspezialisierte Prozessoren nutzen, wie die Produkte des Herstellers Turtle Beach Systems, Inc. Nachteilig ist hierbei, daß der DSP nicht oder nur sehr aufwendig programmiert werden kann, so daß ein Großteil des Rechenpotentials ungenutzt bleiben muß.

2.3.3 Prozessor für Hochleistungsrechner

Ein weiteres wichtiges Anwendungsgebiet von DSPs sind parallelverarbeitende Systeme. Mitte der 80er Jahre sorgte das Konzept des Transputers für einen immensen Aufschwung der Parallelrechner, die damit erstmals kostengünstig realisiert werden konnten. Aus unterschiedlichen

Gründen unterblieb eine Weiterentwicklung der Transputer-Prozessoren. Gleichzeitig erschienen ausgesprochen leistungsfähige DSPs auf dem Markt, die ebenfalls über hervorragende Voraussetzungen zur Inter-Prozessor-Kommunikation verfügten, wie der TMS 320C40 (vgl. Abschnitt 2.3.3). In der Folge übernahmen diese DSPs mehr und mehr die Funktion der Transputer als preiswerte Komponente skalierbarer Multiprozessorsysteme, da sie weitaus leistungsfähiger waren. Selbst ausgesprochene Transputer-Software, wie das Betriebssystem HELIOS, wurde für den DSP portiert, zusätzlich erschienen neue, DSP-spezifische Echtzeitbetriebssysteme, wie z.B. SPOX des DSP-Spezialisten Spectron Microsystems, Inc.

Die Rechnersysteme dieses Typs sind meist modular aufgebaut, die einzelnen Prozessoren verfügen über lokalen Speicher und kommunizieren über schnelle serielle Schnittstellen. Für die Abwicklung der Kommunikation und die Verwaltung aller Ressourcen kommt im allgemeinen ein spezielles Betriebssystem zur Anwendung. Die Interaktion mit dem Benutzer erfolgt über einen Hostrechner. Anwendungsgebiete dieser Hochleistungscomputer sind numerisch anspruchsvolle Problemstellungen, z.B. aus der Echtzeitsimulation, Prozeßvisualisierung oder Bildverarbeitung.

Als Beispiel eines solchen Systems soll der ‘Personal-Supercomputer MUSIC’ der ETH Zürich genannt werden ([GBK93]). Es handelt sich hierbei um eine ringförmige MIMD-Architektur mit bis zu 63 Prozessorknoten, die jeweils aus einem mit 40 MHz getakteten DSP 96002 von Motorola sowie 3 MByte Speicher (1 MByte statisches und 2 MByte dynamisches RAM) bestehen. Der Hersteller gibt, entsprechende Programmierung vorausgesetzt, eine maximale Rechenleistung von 3.8 GFLOPS an.

2.4 Perspektiven und Alternativen

Außer den DSPs existieren eine ganze Reihe weiterer mehr oder minder innovativer Prozessorkonzepte, die auf das lukrative Marktsegment der Digitalen Signalverarbeitung abzielen. Grundsätzlich kann man dabei 2 konträre Klassen von Prozessoren unterscheiden, die fest für einen bestimmten Algorithmus entwickelten Spezialprozessoren und die Universal-Mikroprozessoren der Personalcomputer. DSPs befinden sich innerhalb dieser Einteilung gewissermaßen an der Grenze zwischen beiden Klassen.

2.4.1 Konkurrenz durch Spezialprozessoren

Ein Hauptnachteil der digitalen Signalverarbeitung mit DSPs ist die verhältnismäßig geringe obere Grenzfrequenz der verarbeitbaren Signale, die durch die Taktfrequenz des Prozessors determiniert wird. Gegenwärtig werden die schnellsten DSPs mit etwa 80 MHz getaktet, je nach Komplexität des Algorithmus können damit maximal Signale mit Frequenzen von bis zu einigen hundert Kilohertz bearbeitet werden. Die Verarbeitung höherfrequenter Signale bleibt

Spezialprozessoren vorbehalten, die durch eine feste Verdrahtung höhere Arbeitsfrequenzen erlauben. Sie sind für Standard-Algorithmenklassen verfügbar, wie die schnelle Fouriertransformation (FFT), digitale Filter (vgl. [Moto88C]) oder die diskrete Cosinustransformation (DCT). Für signalverarbeitungsfremde Algorithmen, die mit DSPs bearbeitet werden können, existieren ebenfalls Spezialprozessoren, z.B. die Grafikcoprozessoren und MPEG-Decoder. Aufgrund sehr hoher Stückzahlen können diese Schaltkreise bedeutend preiswerter als DSPs angeboten werden.

Ist größere Flexibilität bei hoher Verarbeitungsgeschwindigkeit erforderlich, so setzt man mehrfachprogrammierbare Schaltkreise, wie FPGAs oder PLDs, ein, deren Logikfunktionen durch den Anwender beliebig oft programmiert werden können. Nachteilig ist jedoch der deutlich höhere Preis sowohl der Prozessoren als auch der zugehörigen Entwicklungssysteme, sowie der erforderliche große Aufwand bei komplexen arithmeitschen Operationen, z.B. Multiplikationen.

2.4.2 Konkurrenz durch Universalprozessoren

Seit dem Eindringen von DSPs in die Architektur von Personalcomputern (vgl. Abschnitt 2.3.2) wird in der Fachwelt die Diskussion um den Sinn dieser Integration geführt. Kern der Debatte ist die Frage, ob die aktuellen Vertreter der Universalprozessoren (Intel Pentium, PowerPC, DEC Alpha) allein in der Lage sind, anfallende Algorithmen der Digitalen Signalverarbeitung und verwandter Gebiete zu bearbeiten, oder ob es vorteilhaft ist, zu deren Entlastung einen Digitalen Signalprozessor in das Computersystem zu integrieren.

Hauptargumente der Gegenseite dieser Entwicklung sind die folgenden (siehe u.a. [SPL92]):

- Modernste konventionelle CPUs arbeiten weitaus schneller als modernste DSPs, da ihre Architektur weiter entwickelt ist und sie mit viel höheren Taktraten betrieben werden.
- Die notwendige Kommunikation zwischen beiden Prozessoren ist unökonomisch und entfällt logischerweise in einer Ein-Prozessor-Lösung.
- Moderne konventionelle CPUs können die heute in PCs anfallenden relativ simplen Algorithmen der DSV (Abspielen digitalisierter Audioinformationen, Stand- und Bewegtbilddekodierung) allein bewältigen.
- Echtzeitanforderungen bestehen in konventionellen Betriebssystemen dabei nicht.
- Die Integration von DSPs in künftige Chipsätze bzw. Motherboards ist aufwendig und kaum standardisierbar.
- Es sind neue Betriebssysteme erforderlich, um DSPs auszunutzen.

- Leistungsfähige DSPs verteuern den Rechner.

- DSP-Software kann nur von Spezialisten erstellt werden, es ist ein DSP-Entwicklungssystem zusätzlich notwendig. Die Softwareentwicklung für ein Ein-Prozessor-System ist weitaus einfacher.

- Es existieren besser optimierende Compiler für CPUs als für DSPs, so daß große Projekte auf DSPs ineffizient implementiert werden.

Folgende Gründe sprechen für den Einsatz von DSPs:

- DSPs sind preiswert. Da Audio-Wandlerhardware auf jeden Fall in die Architektur moderner Computer integriert wird, fällt ein zusätzlicher DSP auf den Preis bezogen kaum ins Gewicht. Darüber hinaus ist der Anschluß von Wandlerhardware an DSPs besonders einfach.

- DSPs sind sehr gut zur Inter-Prozessor-Kommunikation geeignet. Der Aufwand für die Datenübertragung zwischen CPU und DSP bleibt daher in vertretbaren Grenzen.

- DSV-Algorithmen sind partitionierbar, DSPs sehr gut für Multiprozessorarbeit geeignet. Anstatt das gesamte Motherboard eines veralteten Rechners auszutauschen, ist es bei steigenden Anforderungen an die Signalverarbeitungsleistung sinnvoller und preiswerter, mehrere DSPs zu nutzen.

- Es existiert eine große Anzahl Algorithmen hoher Komplexität, die für Multimedia-Applikationen Bedeutung besitzen (z.B. Spracherkennung, Bildverarbeitung). Die Integration von DSPs in zukünftige Rechnerarchitekturen könnte die Nutzung dieser Algorithmen fördern.

- In künftigen Betriebssystemen sind Funktionen zu erwarten, die hohe Rechenleistungen mit digitalisierten Daten unter Echtzeitbedingungen erfordern (Multimedia-Dienste, Verarbeitung von Bewegtbildern und qualitativ hochwertigen Audioinformationen). DSPs sind ideal dafür geeignet.

- Ein flexibel programmierbares DSP-Subsystem könnte mehrere periphere Komponenten überflüssig machen (z.B. Videobeschleuniger, Klangerzeugung, Modem, Handschriftleser).

Nicht unerwähnt bleiben soll, daß die Meinung ‘DSPs sind in Computern überflüssig.’ besonders vehement von den Herstellern der Universalprozessoren (Intel, DEC, Apple) verfochten wird, da sich mit Ausnahme Motorolas in derem Angebot keine DSPs befinden. So startete Marktführer Intel Anfang 1995 unter der Bezeichnung ‘Native Signal Processing’ (NSP) eine Kampagne mit dem Ziel, die Ausführung von DSV-Algorithmen mit Hilfe des Intel Pentium zu

propagieren. Zu den ersten Ergebnissen dieses Projekts zählt sicherlich die Portierung des DSP-Echtzeitbetriebssystems SPOX ([Spec94], [Spec95]) auf den Intel Pentium unter der Bezeichnung SPOX-IA (~ -Intel Architecture).

Innerhalb der Hochleistungsrechner gibt es mehrere Alternativen zum DSP. Multiprozessorsysteme werden z.B. ebenfalls mittels schneller Universalprozessoren (PowerPC, Alpha AXP) mit numerischen Coprozessoren (Intel i860) und immer noch mit Transputern realisiert.

Im Bereich der industriellen Steuerungen stellen moderne Microcontroller (Motorola 68302, Intel i960) die größte Konkurrenz für DSPs dar. Ein hochinteressanter Schaltkreis ist in diesem Zusammenhang die Integration eines Microcontrollers und eines DSP auf einem Chip im Motorola 68356, der den Controller MC68302 und den DSP56002 vereinigt. Beide Prozessoren sind über das Host-Interface lose miteinander gekoppelt, können jedoch auch vollständig separat angesprochen werden.

Obwohl den Digitalen Signalprozessoren äußerst leistungsfähige konkurrierende Prozessorkonzepte gegenüberstehen, ist nach vorherrschender Expertenmeinung (siehe dazu aktuelle Diskussionen in den Internet-Foren comp.dsp bzw. comp.arch) kein Nachlassen der Bedeutung von DSPs abzusehen.

3 DSP als Coprozessor im PC-Betriebssystem

In diesem Kapitel soll das Konzept zur Lösung der Aufgabenstellung dargelegt werden. Es erfolgt zunächst eine Motivation zur Wahl der konkreten Realisierungsumgebung. Im Anschluß daran werden Anbindungsvarianten der DSP-Hardware an den Host diskutiert. Ein abschließender Abschnitt ist der Auswahl eines geeigneten Appplikationsbeispiels gewidmet.

3.1 Auswahl der Hardware

Um möglichst verallgemeinerbare Ergebnisse zu erzielen, ist für das Hostsystem die Nutzung generell verfügbarer, billiger IBM-PC-kompatibler Hardware zweckmäßig. Dabei müssen aktuelle Prozessortypen (Intel 486DX 100, Intel Pentium) zum Einsatz kommen. Wie im Abschnitt 2.3.2 erwähnt, sind PCs mit geringem Aufwand nur über Steckkarten zu erweitern, so daß das zu integrierende DSP-Subsystem auf diese Art eingebunden werden muß (Abbildung 7). Eine ebenfalls denkbare Kopplung zwischen PC und DSP-System über eine serielle Schnittstelle scheidet aufgrund zu geringer Datenübertragungsraten aus.

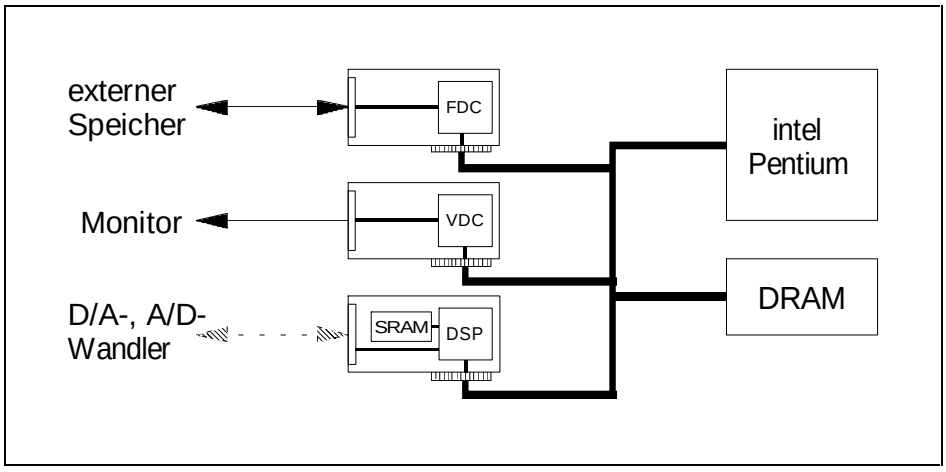

Abbildung 7: Hardwareseitige Einbindung eines DSPs in ein PC-System

Für die praktischen Untersuchungen dieser Arbeit wurde ein DSP mit einer leicht beherrschbaren, aber leistungsfähigen Festkomma-Architektur vorgeschlagen, außerdem sollte der Prozessorbefehlssatz möglichst universell sein, um auch Algorithmen, die nicht der Digitalen Signalverarbeitung zuzuordnen sind, realisieren zu können. Die Wahl fiel letztendlich auf den Typ Motorola DSP56001, dessen technische Charakteristika im Abschnitt 2.2.3 erläutert wurden. Dieser Prozessor ist neben den Vertretern der Reihe TMS320 von Texas Instruments einer der momentan am weitverbreitetsten DSPs.

Obwohl der Prozessor selbst kostengünstig ist, sind die nur von wenigen Anbietern vertriebenen Entwicklungssysteme teuer. Das IDEAL56-System der Berliner Firma RCN erwies sich als geeigneter Kompromiß. Neben der Einsteckkarte gehören Assembler, Linker, Simulator, Debugger und mehrere DSP-Bibliotheken zum Lieferumfang.

Aufgrund der finanziellen Rahmenbedingungen konnte keine Rücksicht auf die Integration von A/D- bzw. D/A-Wandlern auf der Einsteckkarte genommen werden, was die Zahl der sinnvollen Applikationen ein wenig einschränkt. Da alle relevanten Anschlüsse des DSPs auf einen peripheren Steckverbinder geführt sind, steht einer späteren Ergänzung des Boards über eine separate Platine jedoch nichts im Wege. Zum Zeitpunkt des Beginns der Arbeit existierten im übrigen noch keine PC-Einsteckkarten mit dem schnellen PCI-Bussystem, so daß auf eine konventionelle ISA-Karte zurückgegriffen werden mußte.

Das Board ist mit 32KWorten statischem RAM ausgerüstet, was auch umfangreiche Programme ermöglicht. Zusätzlich zum mit 20 MHz getakteten DSP wurde ein Zeitgeberbaustein Intel 8254 einbezogen, der z.B. zur Abtastratengenerierung oder als Watchdog eingesetzt werden kann. Die Verbindung zum PC erfolgt über eine 8 Bit breite parallele Schnittstelle, das sogenannte Hostinterface (HI). Es wird logisch auf 8 aufeinanderfolgende Adressen im Peripherieadreßraum des PC ([RCN92]) abgebildet. Leider ist es nicht möglich, von Seiten des Signalprozessors Unterbrechungen im PC auszulösen, was effektive Datenübertragung in dieser Richtung erschwert.

3.2 Anbindung an Betriebssystem des Hosts

3.2.1 Auswahl eines geeigneten Betriebssystems

Die Aufgabenstellung schlägt das experimentelle Betriebssystem VSTa als Basis aller praktischen Arbeiten vor und definiert gleichzeitig wesentliche Anforderungen:

- moderne Architektur
- auf Microkern basierend
- nach Möglichkeit echtzeitfähig

Der ca. 40kByte umfassende Kern des VSTa übernimmt ausschließlich die Verteilung von Nachrichten sowie die Verwaltung der Prozesse und des virtuellen Speichers. Obwohl VSTa nicht wirklich echtzeitfähig ist, bietet es eine Reihe Merkmale, die Echtzeitbetriebssysteme auszeichnen ([Vale95], [Jesk95]). Darüber hinaus existierten bereits umfangreiche Erfahrungen bei der Bearbeitung eines verwandten Problems, der Integration eines FPGA-Boards in VSTa, die für die vorliegende Arbeit eine Grundlage bildeten. Nicht zuletzt ist VSTa wegen seiner

Einfachheit und Anspruchslosigkeit bezüglich Speicher- und Prozessorressourcen eine ideale Umgebung für experimentelle Untersuchungen an Betriebssystemen.

3.2.2 Prinzip der Integration

Die im Lieferumfang des Boards enthaltenenen Routinen zur Kommunikation zwischen Hostrechner und DSP-Board sind ausschließlich für das Betriebssystem MS-DOS ausgelegt, welches als Cross-Entwicklungsplattform dient. Da als Zielplattform der PC unter VSTa fungieren soll, sind zuerst geeignete Mechanismen für die Kommunikation CPU-DSP unter VSTa zu entwickeln.

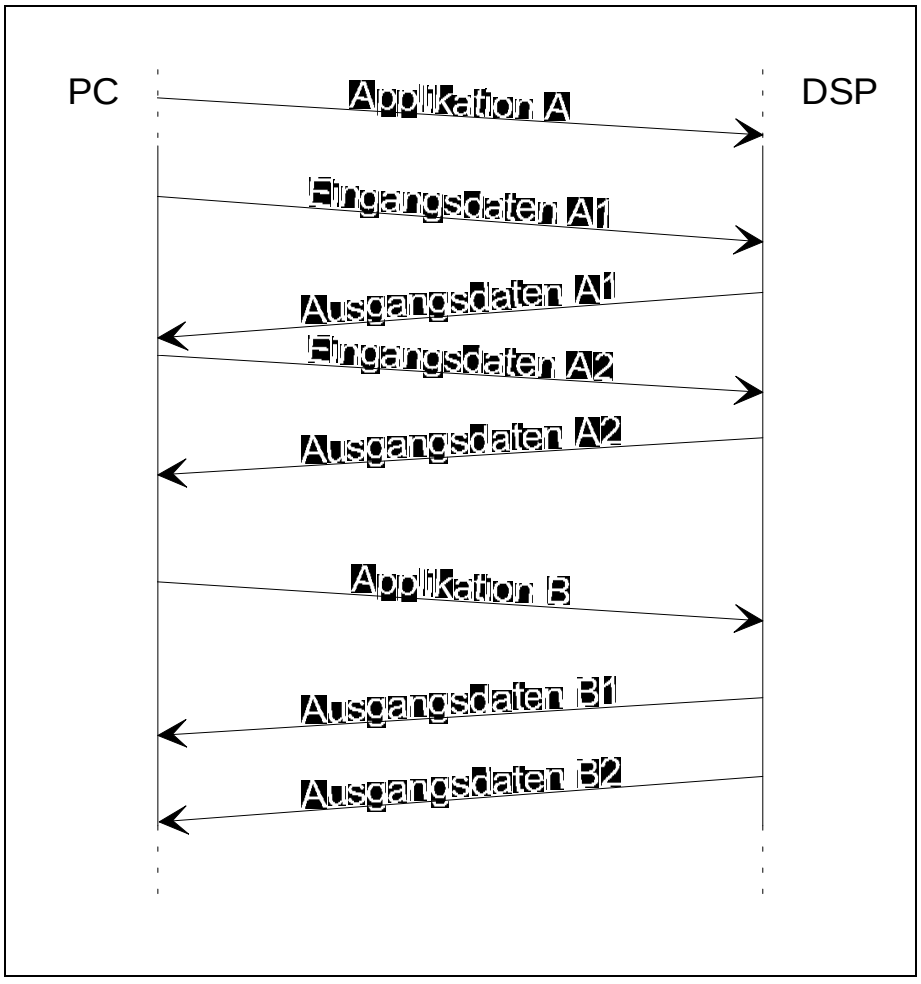

Abbildung 8: Beispielhaftes Kommunikationsszenario zwischen DSP und PC

Abbildung 8 zeigt einen angestrebten Kommunikationsablauf zwischen CPU und Signalprozessor innerhalb des um DSP-Dienste erweiterten VSTa. Da das DSP-Subsystem als flexibler Festkomma-Coprozessor genutzt werden soll, um die CPU des PC von rechenzeitintensiven bzw. zeitkritischen Aufgaben zu entlasten, ist es wünschenswert, die DSP-Applikationen zur Laufzeit des VSTa variabel zu halten. Im Beispiel könnte Applikation A z.B. einen Transcoder

zwischen den Grafikformaten JPEG und Targa repräsentieren und Applikation B einen Prozeß, der Audiosignale digitalisiert und anschließend verarbeitet, z.B. eine Rauschunterdrückung durchführt.

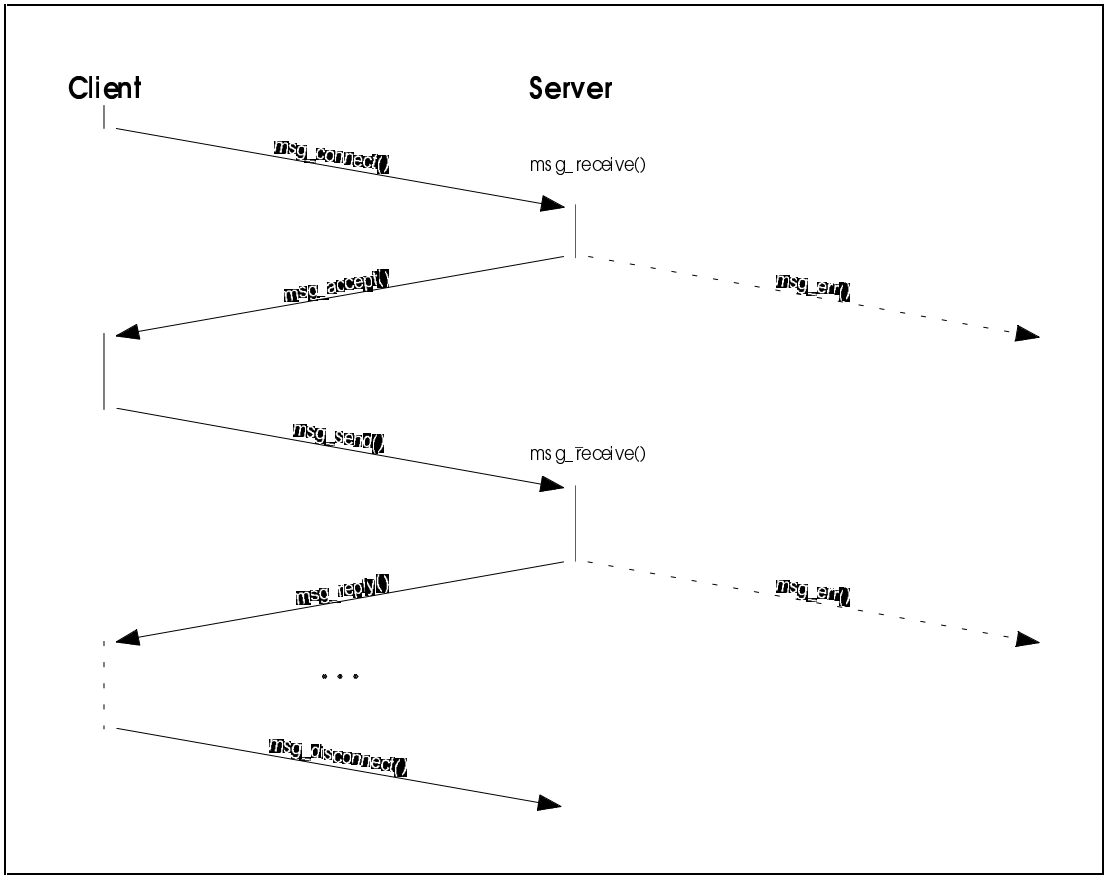

Abbildung 9: Client-Server-Kommunikation in VSTa

Prinzipiell handelt es sich beim DSP-Subsystem um periphere Hardware, was die Nutzung eines Gerätetreibers nahelegt. Im VSTa werden diese generell als Server innerhalb einer Client-Server-Architektur ausgeführt.

Eine definierte Menge von 9 Systemrufen zum Nachrichtenaustausch ist für die gesamte Interprozeßkommunikation im VSTa verantwortlich. Ein Prozeß, der Dienste des Servers nutzen will, muß dies zunächst beim Server beantragen. Dieser prüft daraufhin die Zugriffsrechte des Clients und akzeptiert oder verhindert die Kontaktaufnahme. Bei erfolgreichem Verbindungsaufbau kann nun der Client Dienste des Servers nutzen, jeder Ruf ist mit einer Quittung verbunden. Der abschließende Verbindungsabbau kann sowohl vom Server als auch vom Client erfolgen (Abbildung 9).

Alle Dienste außerhalb des Kerns werden auf diesen Mechanismus abgebildet, so existieren z.B. Server zur Verwaltung des Filesystems, der Konsole (zeichenorientierte Ein- und Ausga-

be) und des globalen Namensraums. Ein Server zur Nutzung des DSP-Systems muß nach dem gleichen Regime arbeiten.

3.2.3 Entwurf der DSP-Softwarestruktur

Für die Software des DSPs sind verschiedene Strukturen denkbar, die sich in Komplexität und Ressourcenausnutzung unterscheiden.

3.2.3.1 Bootvorgang

Wie bereits erwähnt, soll die DSP-Applikation zur Laufzeit des Systems variabel sein. Erschwerend wirkt, daß der DSP nach Erhalt eines Reset-Signals zuerst statisch 512 Worte Programmdaten einliest und diese anschließend abarbeitet. Dieser Mechanismus kann nicht umgangen werden. Da DSP-Applikationen die Größe von 512 Worten häufig überschreiten und ebenfalls Daten im X- und Y-Speicher beinhalten, wird vorgeschlagen, in dieser Phase einen separaten Lademodul zu übertragen, der danach seinerseits für die Übertragung einer kompletten DSP-Applikation verantwortlich ist.

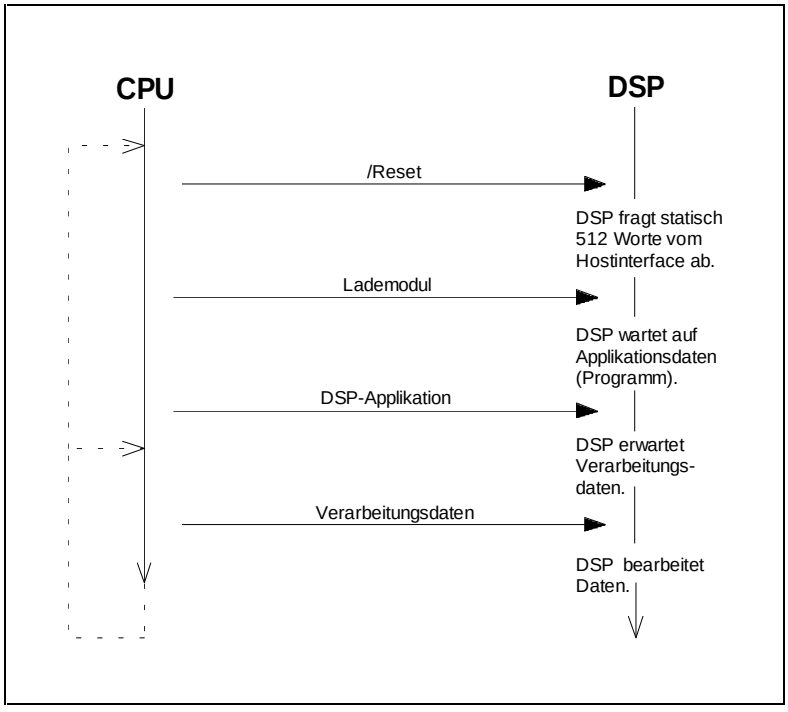

Abbildung 10: Phasen des DSP-Bootvorgangs

Die Abbildung 10 illustriert die einzelnen Phasen der Datenübertragung zum DSP, beginnend mit dem Reset-Signal bis zur Übermittlung der zu bearbeitenden Daten. Eine detaillierte Beschreibung des Formats der zu übertragenden Daten beinhaltet Abschnitt 4.3.3.

3.2.3.2 Datenübertragung des DSP mittels Polling

Jede Applikation, die durch den DSP abgearbeitet wird, kann in folgende Phasen gegliedert werden:

- Empfang der Eingangsdaten
- Transformation der Eingangs- in Ausgangsdaten
- Sendung der Ausgangsdaten

Die einfachste Struktur für die Software des DSP kann erzielt werden, indem dieser die Ein- und Ausgangsdaten durch statisches Polling seiner Übertragungsregister übermittelt, was einer streng sequentiellen Abarbeitung obiger Phasen entspricht. Die gesamte Funktionalität des DSP wird dazu in einen einzigen Prozeß integriert. Nachteilig ist, daß Übertragungspausen zwischen dem Empfang oder der Sendung von Datenworten nicht durch den Prozessor zur Weiterarbeit genutzt werden können, zumal die Ein- und Ausgabebefehle des Intel Pentium verhältnismäßig langsam ausgeführt werden.

Auf Seiten des PC ist ein statisches Polling zur Datenübertragung unbedingt notwendig, da die Hardware des IDEAL56-Systems keine Möglichkeit der Auslösung von PC-Interrupts bietet.

3.2.3.3 Datenübertragung des DSP unter Nutzung von Interrupts

Kommt Wandlerhardware für die Ein- oder Ausgabe von Daten zum Einsatz, so kann deren Ansteuerung nicht mit dem ineffizienten Polling-Verfahren erfolgen. Statt dessen muß eine Signalisierung per Interrupt vorgenommen werden, die Datenübertragung zum PC kann dabei wahlweise ebenfalls über Interruptroutinen oder weiterhin mittels Polling realisiert werden. Eine komplexere Prozeßstruktur der DSP-Software ist in jedem Fall die Folge. Einige Realisierungsbeispiele sollen in diesem Abschnitt diskutiert werden, wobei es sich als günstig erweist, die DSP-Applikationen nach der Richtung des Datenflusses zu klassifizieren.

Die erste Gruppe von Applikationen erlaubt ausschließlich einen unidirektionalen Datenstrom, d.h. Daten werden entweder nur vom PC zum DSP übertragen oder umgekehrt. Das DSP-Subsystem wird damit zum flexiblen Ein-/Ausgabegerät, z.B. für Audiosignale.

Abbildung 11 zeigt beispielhaft für die Übertragungsrichtung zum DSP zwei geeignete Prozeßmodelle. Im Modell a) übernimmt der Prozeß Π_1 abwechselnd die Datenübertragung vom PC und deren Verarbeitung, währenddessen Prozeß Π_2 ausschließlich für die Ausgabe der transformierten Daten an geeignete Wandlerhardware verantwortlich ist.

Modell b) besitzt hingegen für die Datenübertragung bzw. -Transformation zwei separate Prozesse, was den Vorteil hat, daß Pausen innerhalb der Datenübertragung vom PC zur Weiterarbeit durch Prozeß Π_2 genutzt werden können. Dies würde beispielsweise bei einer generellen

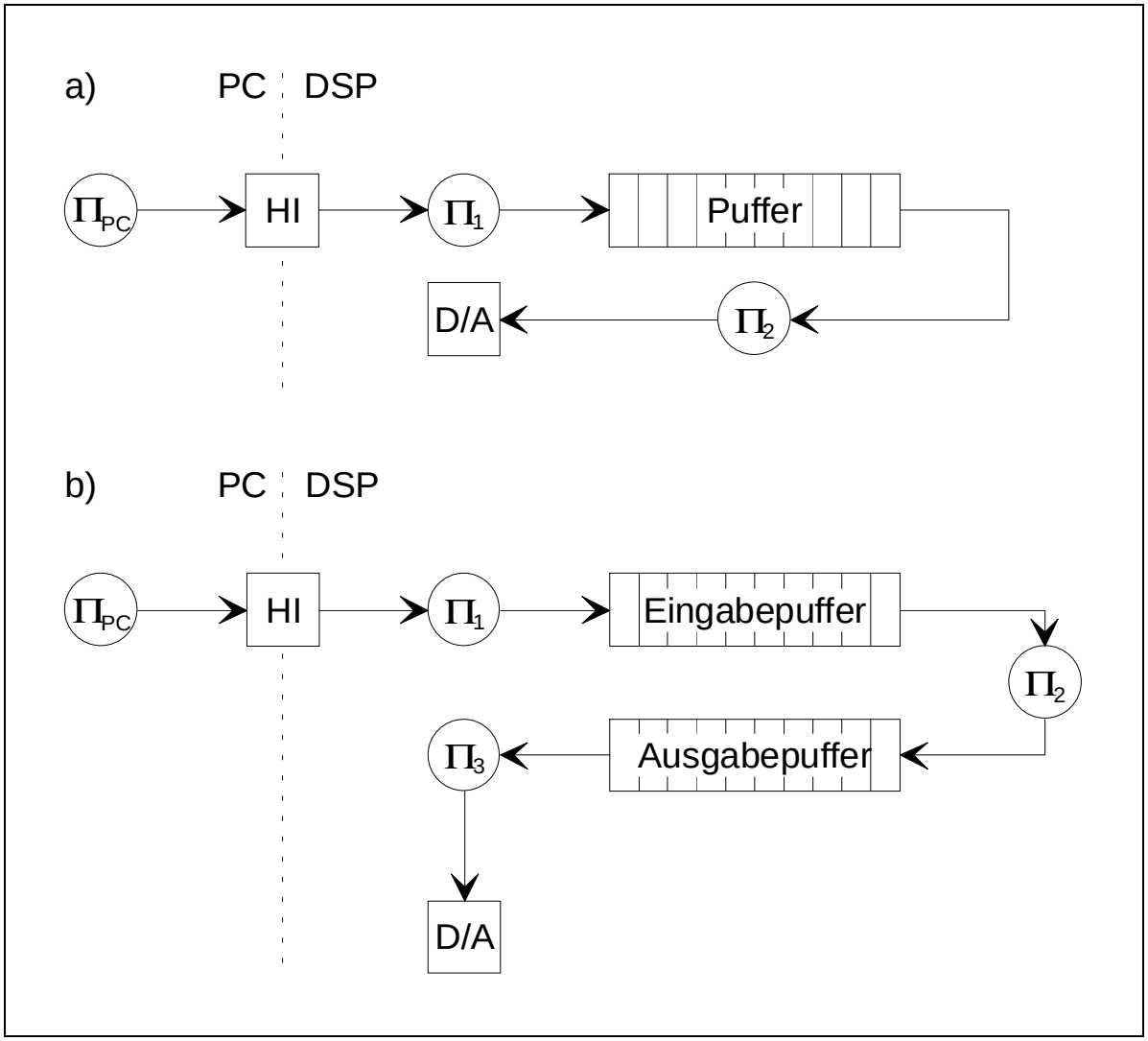

Abbildung 11: Prozeßmodell für unidirektionale Datenübertragung

langsamen Übertragung signifikante Leistungsvorteile gegenüber Variante a) bewirken. Π_2 kann im einfachsten Fall Daten aus dem Eingangs- in den Ausgangspuffer kopieren, falls beispielsweise die Übertragung 'roher' Sampledaten vom Hostrechner an den Wandler erwünscht ist. Im Normalfall wird er jedoch einen mehr oder minder komplexen Algorithmus ausführen, so daß auf ihn der weitaus größte Anteil an Prozessorzeit entfällt.

Es ist nicht möglich, Modell a) durch Kombination von Π_1 und Π_2 zu einem Prozeß weiter zu vereinfachen, da die Ausgabe von Daten an D/A-Wandler (bzw. ebenso die Abtastung mittels A/D-Wandler) in einem exakt einzuhaltenden, starren Zeitraster erfolgen muß, welches vollkommen asynchron zur eigentlichen Programmabarbeitung auf dem DSP liegt.

Die hier getroffenen Aussagen gelten ebenso für die "Rückrichtung", d.h. für den Datenfluß vom DSP zum Hostrechner, der durch Umkehrung aller Pfeilrichtungen in Abbildung Abbildung 11 veranschaulicht werden kann.

Der DSP könnte mit diesem Verarbeitungsmodell z.B. als flexibler Codec (kombinierter Coder und Decoder) zur Ein-/Ausgabe von Audiosignalen betrieben werden, wobei die Kodierungsalgorithmen je nach Erfordernis austauschbar sind.

Kommt keine Wandlerhardware zum Einsatz, so bedingt die Applikation meist einen bidirektionalen Datenstrom, Daten werden also gleichzeitig vom PC zum DSP und umgekehrt übertragen. Denkbare Prozeßmodelle zeigt Abbildung 12.

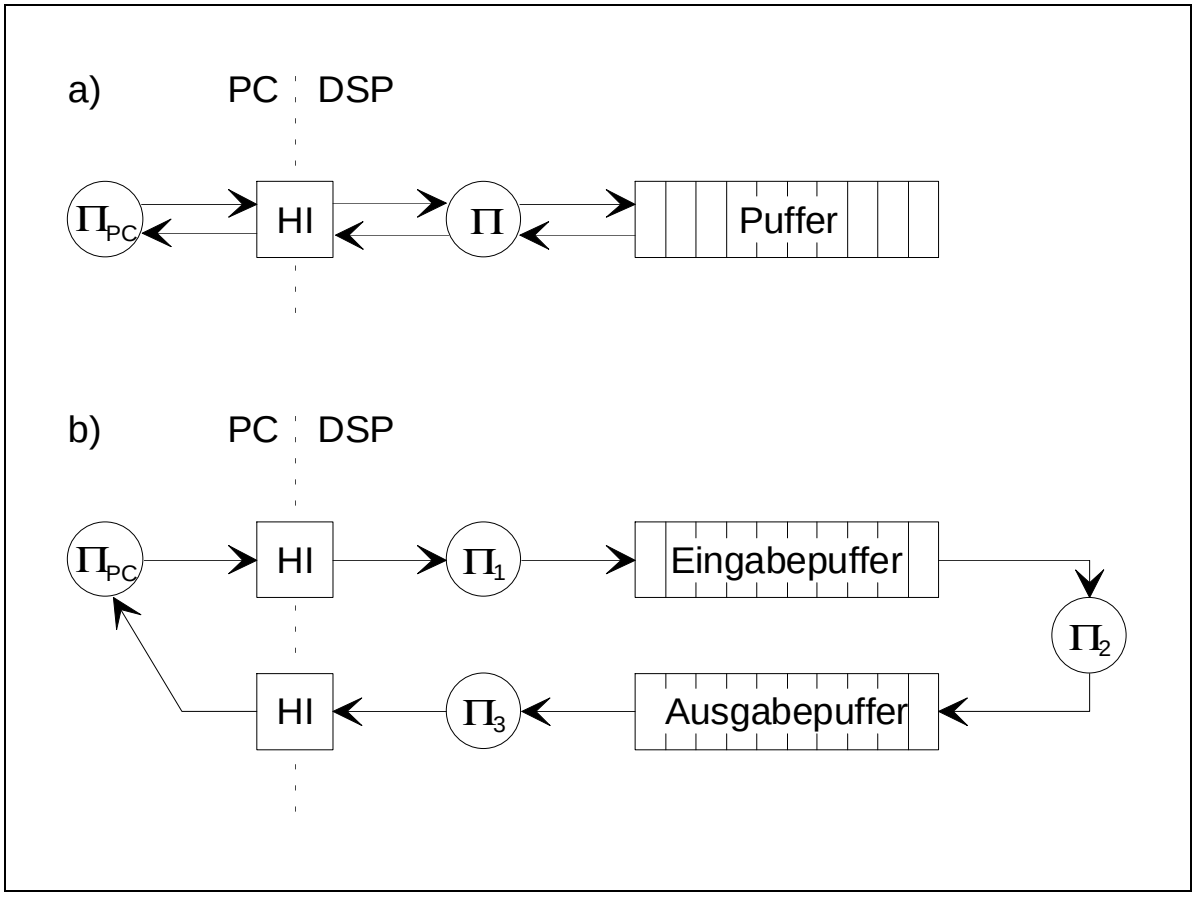

Abbildung 12: Prozeßmodelle für bidirektionalen Datenstrom

Modell a) zeigt die resultierende Prozeßstruktur beim Polling der Hostschnittstelle gemäß Abschnitt 3.2.3.2. Alle Aufgaben (Einlesen von Daten, Transformation in Ausgangsdaten, Ausgabe der Ergebnisse) werden sequentiell durch einen Prozeß abgearbeitet. Für die praktischen Untersuchungen wurde aus Aufwandsgründen zunächst dieses Modell verwirklicht.

Modell b) repräsentiert die leistungsfähigste Prozeßstruktur, da die Ein- und Ausgabe der Daten vollständig von deren Verarbeitung entkoppelt ist. Dieses Modell bedingt mit 3 voneinander unabhängigen Prozessen damit auch die komplexeste Programmstruktur des DSP.

3.3 Applikationen

3.3.1 Prämissen bei der Auswahl von geeigneten Algorithmen

Die starke Spezialisierung der Architektur von DSPs führt zu sehr günstigen Bedingungen für die Abarbeitung einer bestimmten Menge von Algorithmen, währenddessen andere Algorithmen weitaus aufwendiger zu implementieren sind und die resultierenden Programme ein ungünstigeres Laufzeitverhalten aufweisen. Bedingt durch die notwendige Kommunikation zwischen den Prozessoren kommt es bei Nutzung eines DSP gegenüber der DSP-losen Implementierung zu Zeitverlust. Die DSP-Applikation muß wenigstens um diesen Zeitverlust schneller arbeiten, damit sich der Einsatz des zusäätzlichen Prozessors rentiert. Generell gilt, daß das Verhältnis der Zeit für die Datenübertragung zur Zeit für die Verarbeitung der Daten möglichst klein sein muß; je mehr Operationen pro Datum ausgeführt werden, desto besser.

Für die Auswahl von Applikationen, die eine Beschleunigung durch Einsatz eines DSP erwarten lassen, sollen einige Regeln formuliert werden. Da die Architektur der DSPs in weiten Grenzen variiert, muß für konkrete Aussagen der zu benutzende Prozessor-Typ mit seinen individuellen Merkmalen bekannt sein. Generell gilt jedoch, daß moderne DSPs universeller sind als ältere Typen.

Folgende Merkmale eines Algorithmus lassen eine leistungsfähige Implementierung durch DSPs erwarten:

- Rechenoperationen erlauben die gleichzeitige Ausführung von Transferoperationen der Folgeoperanden, d.h. die durch die Harvard-Architektur ermöglichte Parallelität kann genutzt werden
- vorwiegende Benutzung von Additionen und Multiplikationen (besonders Summen über Produkten)
- Verzicht auf bedingte Sprünge, Iterationen statt dessen mit bekannter Anzahl von Wiederholungen
- Nutzung der DSP-eigenen Zahlendarstellung (Gleit- oder Festkomma, gebrochenes Zahlenformat), besonders gut als Operanden sind Abtastwerte geeignet
- Verarbeitungsdaten besitzen feste Länge und konstantes Format

- Nur ein fester Anteil der Gesamtheit der anfallenden Daten wird für einen Rechenschritt benötigt („Windowing“, Ringpuffer-Organisation)
- Verzicht auf Divisionen
- Operandenadressierung der FFT nutzbar („Bit-Reversed“- Adressierung)

3.3.2 Beispiele

Einige konkrete Algorithmen können nun mittels der eben beschriebenen Merkmale auf ihre Eignung für DSP-Implementierungen untersucht werden..

Ein Beispiel für eine geeignete Applikation ist der Bildkomprimierungsalgorithmus JPEG (Joint Photographic Experts Group), der obige Anforderungen recht gut erfüllt. Zu kodierende Bilder werden in 8x8-Pixel-Matrizen partitioniert, die jeweils separat bearbeitet werden. Kernpunkt des Algorithmus ist die Diskrete Cosinustransformation DCT, die für jeden Bildpunkt folgende Operation erfordert (zweidimensionale DCT):

$$C(u,v) = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos\frac{(2y+1)u\pi}{16} \cos\frac{(2x+1)u\pi}{16}$$

Diese Rechenvorschrift erfordert u.a. pro Koeffizient C die Ausführung von 515 Multiplikationen und 168 Additionen. Die Funktionswerte des Cosinus können über eine Tabelle bestimmt werden, da nur eine endliche Anzahl Argumente möglich ist.

Die DCT gilt als recht genau untersucht, es existieren eine ganze Anzahl schneller Algorithmen, daher wird die obige Berechnungsvorschrift nicht direkt in Programmcode umgesetzt. Besipielsweise kann die Transformation auf Matrixmultiplikationen abgebildet werden. In jüngster Zeit erschienen sogar Algorithmen, die die FFT zur Berechnung der DCT einsetzen.

Als Applikationsbeispiel dieser Arbeit wurde ein JPEG-Decoder mit beschränktem Funktionsumfang ausgewählt.

Da bei der Bewegtbild- und Audiokodierung nach MPEG ähnliche Rechenoperationen wie bei JPEG erforderlich sind, gelten obige Aussagen genauso für diese Algorithmen. Aufgrund des hohen Datenaufkommens bleiben sie aber nur sehr leistungsfähigen DSPs vorbehalten, währenddessen die entsprechenden Dekoder weitaus leichter zu realisieren sind.

Eine weitere gut mit DSPs zu realisierende Klasse von Algorithmen sind die Neuronalen Netze. Eine typische Aktivierungsvorschrift eines Neurons ist z.B.:

$$y_j = f_a\left\{\sum_{i=0}^{N} x_i w_{ij} - \Theta\right\}$$

Auch in diesem Fall kommen vorwiegend Multiplikationen und Additionen zum Einsatz, die Operation wird über einer größeren Menge von Neuronen ausgeführt, und die verarbeiteten Werte besitzen ein einheitliches Format, meist Gleitkommadarstellung. Des weiteren bietet sich die einfache Anschaltung von Peripherie, z.B. in Form von Sensoren, bei Neuronalen Netzen förmlich an, da diese häufig in der Sprach- und Bilderkennung zum Einsatz kommen.

Alle Arten der Verarbeitung vektorieller Größen und Matrizen können ebenfalls als DSP-geeignet klassifiziert werden. Aus diesem Grunde existieren mehrere DSP-Applikationen als Vektorgrafik-Coprozessor. Allerdings besteht der Nachteil, daß hierbei oft Ergebnisse voneinander abhängen, was die maximal erreichbare Parallelität einschränkt.

Beispiele für zur DSP-Umsetzung ungeeignete Algorithmen sind Such- und Sortierfunktionen (hohe Anzahl bedingter Sprünge, schlechte Parallelisierbarkeit auf Harvard-Struktur, keine Arithmetik) oder Operationen mit Zeichenketten (variable Länge der Operanden).

3.3.3 Entwicklungsetappen des ausgewählten Beispiels

Wie im vorigen Abschnitt erwähnt, wurde als zu realisierende Applikation ein Decoder für das Standbildkomprimierungsverfahren JPEG ausgesucht. Da JPEG ein äußerst komplexer Standard ist, mußte eine Beschränkung auf eine Untermenge aller möglichen JPEG-Verfahren und Formate erfolgen, die die wesentlichen Aspekte berücksichtigt. Des weiteren soll nur der Teil der gesamten Funktionalität des JPEG-Decoders auf den DSP ausgelagert werden, der einen Zeitgewinn erwarten läßt. Details hierzu enthält Kapitel 4.

Ausgehend von den frei verfügbaren Programmquellen der Portable Video Research Group (PVRG) der Stanford University ([Hung93]) sollte eine Version für VSTa implementiert werden, die den DSP für wesentliche Verarbeitungsschritte nutzt, sowie eine Version mit äquivalentem Funktionsumfang, die ohne diesen arbeitet.

Die Entwicklungsarbeiten sollten folgende Etappen umfassen:

1. Implementierung der elementaren Dienste zur DSP-Kommunikation im VSTa
2. Implementierung der darauf aufbauenden komplexeren Funktionen (u.a. Interpreter für DSP-Programmfiles)
3. Erarbeitung einer Implementierung für die Inverse Diskrete Cosinus-Transformation (IDCT) mittels DSP

4. Entwicklung eines JPEG-Decoders mit minimalem Leistungsumfang (Dekodierung einzelner Matrizen, feste Parameter) als Standalone-Applikation unter MS-DOS (ohne Integration des DSP) zum Verständnis grundlegender Mechanismen, dabei schrittweise Implementation und Kontrolle der Resultate bei

   - Analyse der Filestruktur (JPEG File Interchange Format JFIF, vgl. [Hami92])
   - Huffman- und Lauflängen-Dekodierung
   - „De-Zigzagging“ und Dequantisieren der Koeffizientenmatrizen
   - Inverser Diskreter Cosinus-Transformation (IDCT)
   - spaltenweisem Auslesen der rekonstruierten Bilddaten

5. Portierung dieser Minimalversion nach VSTa

6. Konstruktion eines VSTa-Servers aus der vorangegangenen Version (Ersetzung der Fileoperationen durch Message-Passing-Funktionen, Entwurf eines beispielhaften Clients)

7. Verbesserung der Funktionalität des Servers (Zulassen variabler Bildgrößen, Optimierung des Codes, Erarbeitung einer stabilen Lösung)

8. Austausch der Teile des Decoders, die durch den Signalprozessor bearbeitet werden sollen, Integration der Kommunikation zum DSP-Board

Die Migration zum DSP-Server sollte erst nach Fertigstellung der DSP-losen Variante erfolgen, um die Unterschiede zwischen beiden Realisierungsformen zu minimieren.

4 Beschreibung der Implementation

4.1 Betriebssystem VSTa

Als beispielhafte Applikation mit höherer Komplexität wurde ein JPEG-Decoder ausgesucht und implementiert. Als geeignete Betriebssystemabstraktion für die Verwaltung der DSP-Hardware erwies sich der Server. Für vergleichende Messungen war es notwendig, die DSP-gestützten Funktionen mittels einer rein PC-basierten Lösung zu simulieren. Dies führte zu einem etwas kuriosen Ergebnis: einem Server, der ausschließlich zur Dekodierung von JPEG-Strömen dient.

4.1.1 Zusammenspiel zwischen Client und Server

Unabhängig von der Implementierungsvariante kann die Arbeit des Servers in

- Übertragung des JPEG-codierten Datenstroms,
- Dekodierung der Bilddaten,
- Rückübertragung des restaurierten Bildes

strukturiert werden. Zwischen dem Client-Prozeß und dem Server läuft dabei die in Abbildung 13 verdeutlichte Kommunikation ab (auf die Darstellung von Verbindungsauf- und -abbau wurde verzichtet). Der dargestellte zeitliche Ablauf ist grundsätzlich möglich, jedoch abhängig von der konkreten Prozeßsituation. Es kann z.B. nicht garantiert werden, daß nach serverseitigem Aufruf von `msg_reply()` der anfordernde Client sofort zur Abarbeitung kommt, sicher ist nur, daß er nicht *vor* diesem Aufruf wieder aktiviert wird.

Die Funktionalität des Servers mußte zweigeteilt werden, da die Datenübertragung zwischen Prozessen in VSTa immer nur unidirektional möglich ist. Die Unterscheidung der Dienste

- Empfang, Dekodierung und Speicherung eines JPEG-Stroms,
- Sendung eines bereits dekodierten Stroms

erfolgt über das `m_op`-Feld der Nachrichtenstruktur `msg_t`. Die Operationen FS_READ und FS_WRITE wurden der Einfachheit halber dazu „zweckentfremdet“.

Da Server über einen eigenen Namensraum verfügen, ist es nicht möglich, durch den Client nur den Namen der zu verarbeitendenden Datei zu übermitteln, den Server die Filearbeit zu überlassen und die Übertragung der kompletten Dateien zwischen Client und Server einzusparen.

Der zusätzliche Aufwand ist aber zu vertreten, da die Daten nicht physisch kopiert, sondern in den Adreßraum des Empfängers abgebildet werden.

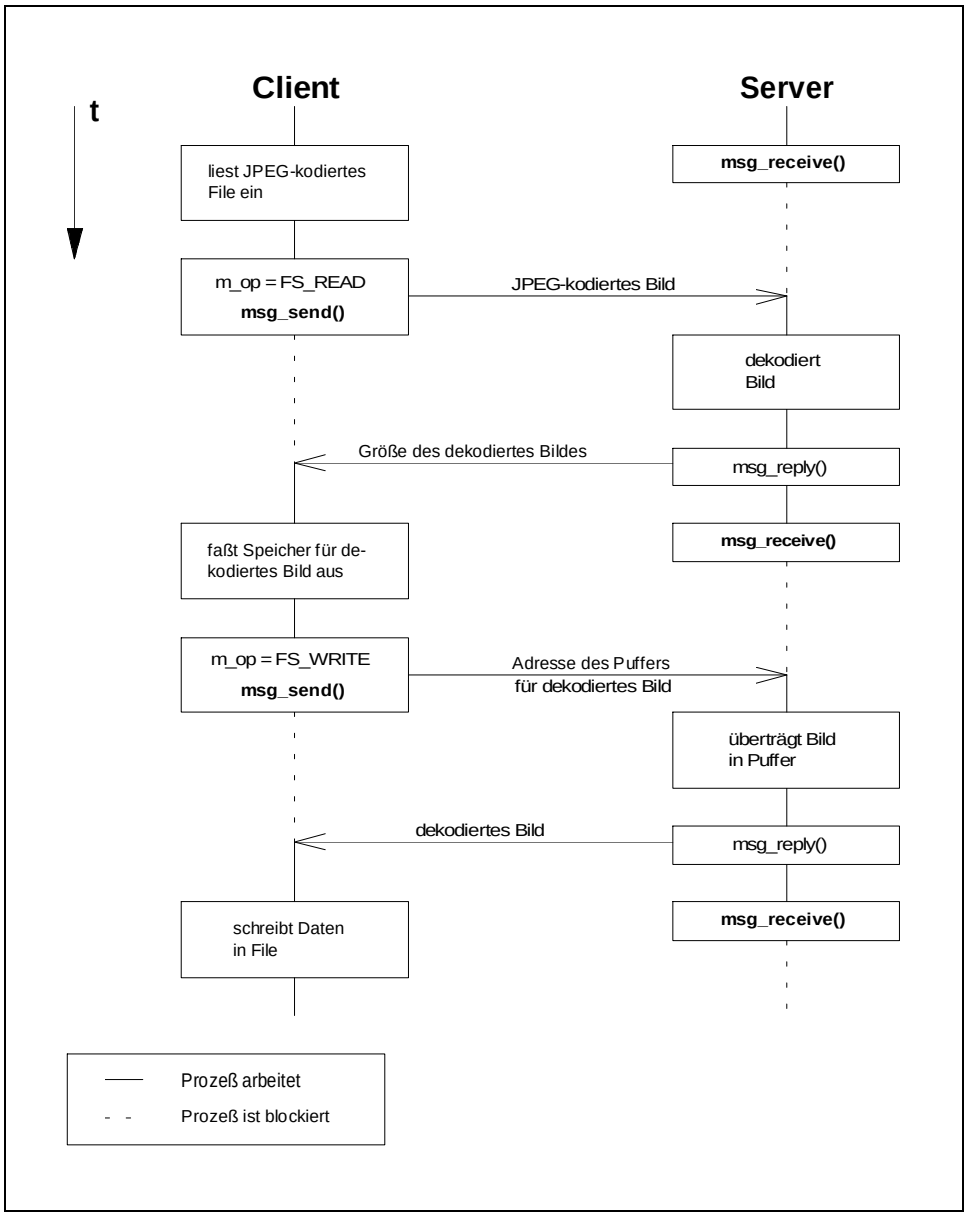

Abbildung 13: Kommunikation zwischen Client und Server

Die eigentlich vorgesehene Anzeige des Bildes auf einem grafikfähigen Bildschirm konnte in VSTa nicht realisiert werden, da kein Zugriff auf das Video-BIOS der Grafikhardware möglich ist. Eine direkte Programmierung der Register der Grafikkarte über Portadressen mußte aus Aufwandsgründen unterbleiben.

4.1.2 Struktur des Servers

Alle Server im VSTa sind nach einem einheitlichen Muster aufgebaut. Nach einer Initialisierung betritt der Server eine endlose Schleife zum Empfang und zur Beantwortung von Nachrichten.

Er führt eine Eingabeoperation `msg_receive()` aus und wird daraufhin blockiert. Sobald ein Client eine Nachricht an den Server überträgt, wird dieser wieder in das Scheduling aufgenommen. Er klassifiziert die empfangene Nachricht nach der Art der angeforderten Operation, führt diese aus und beantwortet die Nachricht. Danach führt ein erneutes `msg_receive()` wieder zur Suspendierung usw.

Die Initialisierung des realisierten Servers ist recht kurz: er läßt vom System einen Port generieren, eine Adresse, über die Clientprozesse Kontakt mit ihm aufnehmen können. Anschließend informiert er das System über seine Existenz und betritt die Nachrichtenschleife.

Tabelle 5 führt die vom Server verarbeiteten Operationen (Feld `m_op` der Struktur `msg_t`) und die damit verbundenen Aktionen auf.

Operation	Aktionen des Servers
M_CONNECT	Falls bereits ein anderer Client mit dem Server kommuniziert, wird die Verbindungsaufnahme verweigert, ansonsten wird sie durchgeführt.
M_DISCONNECT	Existiert im Server bereits ein dekodiertes Bild, so wird dieses verworfen. Der Server ist nun zur Verbindungsaufnahme mit einem neuen Client bereit.
FS_READ	Existiert im Server bereits ein dekodiertes Bild, so wird dieses verworfen. Anschließend erfolgt die Dekodierung der soeben übertragenen Daten sowie die Speicherung des Resultats. Der Ruf wird mit der Größe des dekodierten Bildes beantwortet.
FS_WRITE	Liegt kein dekodiertes Bild vor, erfolgt eine Fehlermitteilung. Ansonsten wird dieses Bild in den übergebenen Puffer kopiert und dieser zurückgeschickt. Das eben übertragene Bild wird danach gelöscht.

Tabelle 5: Vom Server verarbeitete Operationen

Alle anderen Operationen werden mit einer Fehlermitteilung beantwortet. Wünschenswert wäre bei M_CONNECT die Einbeziehung einer korrekten Prüfung der Zugriffsrechte (Capabilities) des Clients.

Der Beschreibung der eigentlichen Dekodierung ist der folgende Abschnitt gewidmet.

4.2 Standbildkodierung nach JPEG

Zur effektiven Ausnutzung der Bandbreite moderner Übertragungsmedien sind leistungsfähige Komprimierungsalgorithmen notwendig. Innerhalb von Multimedia-Diensten nehmen fotorealistische Darstellungen, wie z..B. digitalisierte Fotografien, aber auch durch Raytracing u.ä. generierte Grafiken, breiten Raum ein. Ein speziell für die Reduktion dieser Daten konzipierter Algorithmus ist die Kodierung nach JPEG (Joint Photographic Experts Group, die ursprüng-

liche Bezeichnung für das mit der Standardisierung beauftragte Expertengremium). Im folgenden sollen das Verfahren und die hier gewählte Implementierung beschrieben werden.

4.2.1 Merkmale

JPEG arbeitet grundsätzlich verlustbehaftet, d.h. die Ausgangsinformationen sind nicht exakt restaurierbar. Es werden jedoch nur solche Bildinformationen eliminiert, die für die menschliche Wahrnehmung irrelevant sind. Man nutzt dabei die Eigenschaft des menschlichen Auges, Helligkeitsdifferenzen viel besser als Farbunterschiede wahrnehmen zu können.

Die JPEG-Kodierung, entwickelt Ende der 80er Jahre, ist ein Verfahren, mit dem farbige fotorealistische Darstellungen bis zum Verhältnis 1:20 ohne sichtbare Qualitätseinbuße reduziert werden können. Die Komprimierung von Grauwert-Bildern ist ebenfalls möglich, die erreichbaren Verdichtungsraten sind allerdings kleiner, da diese Darstellungen ausschließlich Helligkeitsinformationen enthalten. Die zueinander umgekehrt proportionalen Parameter Kompressionsrate und Qualität des komprimierten Bildes sind während der Kodierung in weiten Grenzen wählbar.

Die Komprimierung basiert des weiteren auf der Tatsache, daß bei fotorealistischen Bildern Unterschiede zwischen benachbarten Pixeln in Helligkeit und Farbe statistisch gesehen relativ klein und selten sind. Eine spezielle Transformation ermöglicht es, die Informationen solcher ‘homogener’ Bereiche eines Bildes in wenigen Koeffizienten zusammenzufassen. Dies ist ein Grund für die Untauglichkeit des Verfahrens für die Datenreduktion von kontrastreichen Strichvorlagen, Schwarz-Weiß-Zeichnungen usw. Zum anderen induziert JPEG besonders bei hoher Kompressionsrate kleine Fehler in den kodierten Vorlagen, die bei fotorealistischen Bildern für das menschliche Auge unsichtbar sind, sich aber bei Strichvorlagen störend bemerkbar machen, wie das ‘Verwischen’ exakter Linien ([Lane95]).

Der Standard ISO 10918 legt kein Dateiformat für JPEG fest, es existiert momentan nur ein Entwurf für das sog. JFIF (JPEG File Interchange Format, [Hami92]). Dieses Format bietet eine Grundlage für den plattformübergreifenden Austausch von JPEG-kodierten Bildinformationen, verzichtet jedoch auf viele Möglichkeiten des sehr komplexen JPEG-Standards.

4.2.2 Stufen der Kodierung und Dekodierung

An dieser Stelle kann nur ein kurzer Überblick über die Interna der JPEG -Kodierung gegeben werden. Eine umfassende Einführung beinhalten [Gail95] und [Wall91], während für Detailfragen der ISO-Standard 10918 herangezogen werden muß.

Die Kodierung einer Bilddatei umfaßt die folgenden Schritte:

1. Um Farb- und Helligkeitsinformationen getrennt kodieren zu können, werden die Bilddaten, falls notwendig, in das Farbmodell YCbCr transformiert. Die Komponente Y determiniert hierbei die Luminanz (Helligkeit) währenddessen die Komponenten Cb und Cr die Chrominanz (Farbinformation) eines Bildpunktes beinhalten.

2. Optional werden die Farbinformationen mit einstellbarer Schrittweite gröber gerastert, d.h. jeweils benachbarte Pixelpaare werden zu einem Block mit identischer Farbe zusammengefaßt (sog. „Downsampling“). Dieser Schritt halbiert mindestens den Umfang der Farbinformationen.

3. Jede Komponente des Gesamtbildes wird nun in 8x8 Werte große Matrizen partitioniert, die folgenden Bearbeitungsschritte erfolgen identisch für jede Matrix.

4. Jeder Wert einer Matrix wird mittels Verschiebung aus dem Intervall $[0, 2^p-1]$ auf das Intervall $[-2^{p-1}, 2^{p-1}-1]$ abgebildet. Der Wert *p* hängt von der Farbtiefe des zu kodierenden Bildes ab und kann 8 oder 24 betragen.

5. Die Diskrete Cosinustransformation (DCT, auch FDCT für Forward- ~) ermittelt für jede Matrix der Bildinformationen eine gleichgroße Koeffizienten-Matrix, die eine Darstellung der Bildfrequenzen des bearbeiteten Blockes repräsentiert (Berechnungsvorschrift siehe S.32). Dieser Schritt ist bis auf Rundungsungenauigkeiten verlustfrei.

6. Jeder Wert der Koeffizientenmatrix F(u,v) wird quantisiert mit:

$$F^Q(u,v) = \mathrm{Round}\left(\frac{F(u,v)}{Q(u,v)}\right)$$

   Die Wahl der Quantisierungskoeffizienten Q(u,v) beeinflußt maßgeblich die Qualität des kodierten Bildes. Sie bilden ebenfalls eine 8x8-Matrix, die gewöhnlich mit den Bilddaten übertragen wird. Dieser Verarbeitungsschritt ist der eigentlich verlustbehaftete Teil, da viele der höherfrequenten Koeffizienten zu 0 quantisiert werden und damit nicht mehr restaurierbar sind.

7. Die quantisierten Koeffizienten $F^Q(u,v)$ werden danach in einem Zickzack-Muster neu in der Matrix angeordnet. Zweck dieser Umordnung ist die Gruppierung der Koeffizienten ungleich 0 im linken oberen Teil der Matrix und der zu 0 quantisierten Koeffizienten im rechten unteren Teil. Die anschließende Lauflängenkodierung wird durch diesen Schritt effektiviert.

8. Der erste Koeffizient jeder Matrix $F^Q(0,0)$ gibt den durchschnittlichen Farb- bzw. Helligkeitswert des Blocks an, er ist in starkem Maße abhängig von den $F^Q(0,0)$ der benachbarten Matrizen. Er wird daher differentiell zu seinem Vorgängerwert kodiert.

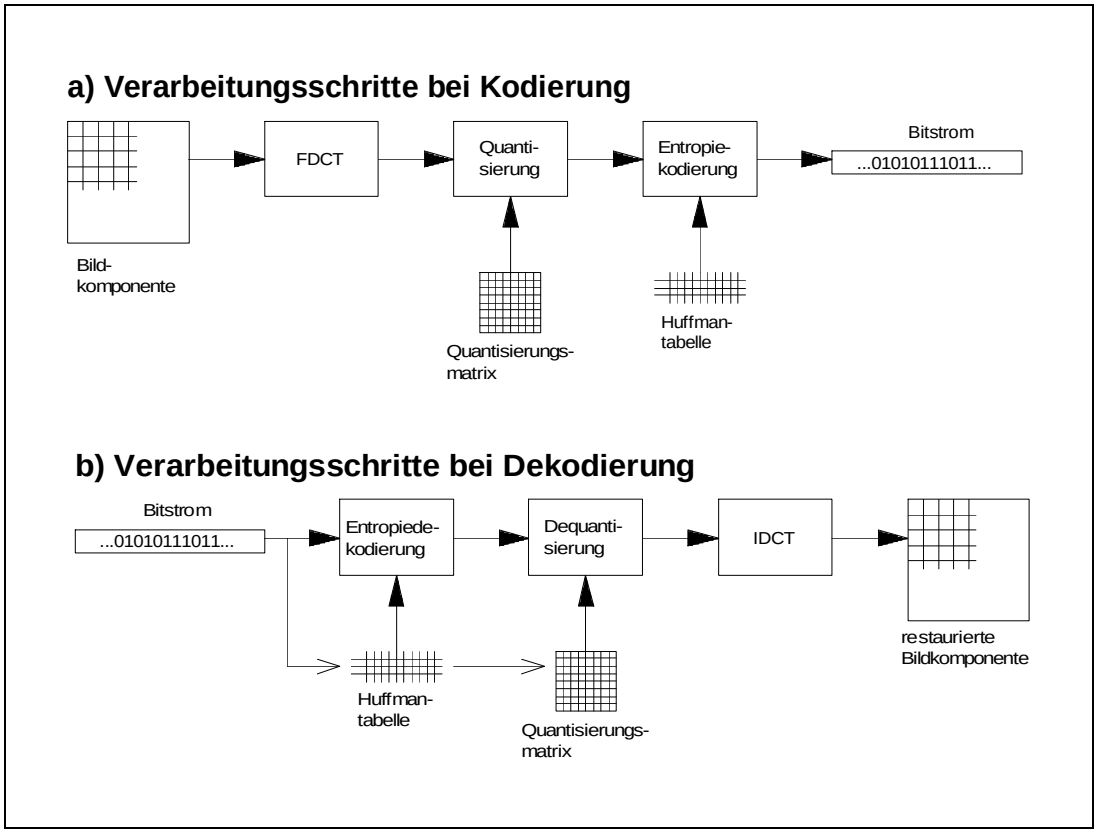

Abbildung 14: Verarbeitungschritte beim JPEG -Verfahren (nach [Wall91])

9. Die resultierenden Matrizen werden Lauflängen- und anschließend Huffman-kodiert (Entropiekodierung). Die notwendige Symbolzuordnungstabelle kann im Coder angelegt werden, oder man benutzt eine durch den Standard vorgeschlagene. In jedem Fall wird diese Tabelle in die Bilddaten integriert.

Die Dekodierung eines JPEG-Bitstroms erfolgt durch Umkehrung der aufgeführten Schrittfolge. Abbildung 14 faßt die wesentlichen Schritte für beide Richtungen noch einmal zusammen.

Der ISO-Standard definiert darüber hinaus weitere Teilverfahren, die wahlweise eingesetzt werden können. So ist neben der Huffman- auch eine arithmetische Kodierung möglich, die um 5-10% bessere Kompressionsergebnisse liefert, aber rechenintensiver ist. Des weiteren existiert ein verlustfreier JPEG-Modus (lossless JPEG), der aber verhältnismäßig ungebräuchlich ist. Zuletzt soll noch die hierarchische Kodierung genannt werden, die den Einschluß von Varianten ein- und desselben Bildes mit wachsender Auflösung in einem Bitstrom regelt.

4.2.3 JPEG-Decoder für Intel Pentium

Dieser Abschnitt gibt einen Überblick über die wesentlichen Implementationsaspekte der Realisierungsvariante ohne DSP. Der anschließende Abschnitt, der der DSP-gestützten Variante vorbehalten ist, geht auf wesentliche Unterschiede in der Programmierung ein und dokumentiert die DSP-Software.

Aufgrund der außerordentlichen Komplexität des JPEG-Standards mußte eine Beschränkung auf eine sinnvolle Untermenge der gesamten Funktionalität vorgenommen werden. Folgende Limitationen betreffen die vorliegende Implementierung:

- Kodierung nur nach Baseline-JPEG möglich (keine arithmetische Kodierung, Bild besteht aus Werten von 8 Bit Länge)
- nur Bilder, die aus einer Komponente bestehen, dekodierbar (d.h. nur monochrome Bilder)
- Länge und Breite des zu dekodierenden Bildes müssen durch 8 teilbar sein
- pro Datenstrom ist nur 1 Bild zulässig

Der gesamte Decoder basiert auf den frei verfügbaren Quellen der Portable Video Research Group (PVRG) an der Stanford University [Hung93]. Diese äußerst umfangreiche Implementation wurde um wesentliche Komponenten reduziert und nach VSTa portiert.

Die Funktionalität des Decoders ist in folgenden Dateien lokalisiert:

dct.c	Inverse Diskrete Cosinus-Transformation
huffman.c	Huffman- und Lauflängenkodierung betreffende Funktionen
jana.c	restliche Funktionen, Initialisierung, Eintrittspunkt

Jede JPEG-kodierte Datei ist in verschiedene Sektionen gegliedert, die durch bestimmte Bytekombinationen, die sog. Marker, eingeleitet werden. Die Funktion `decodejpg()` übernimmt nach der Initialisierungsphase die Analyse der Syntax des Datenstroms und das Einlesen der betreffenden Daten. Tabelle 6 listet die bearbeiteten Marker und ihre Bedeutung auf ([Holt94]).

Aus den Angaben im SOF-Block ist die Größe des Originalbildes determinierbar und damit auch die Anzahl der 8x8-Matrizen, in die es bei der Kodierung partitioniert wurde. Sobald ein SOS-Marker erkannt wird, beginnt mit dem Aufruf von `DecodeStream()` die Bearbeitung des kodierten Datenstroms. Für jede Matrix wird nun die in Abbildung 15 dargestellte Folge von Funktionsaufrufen ausgeführt.

Marker	Bedeutung	enthaltene Daten
SOI	Start Of Image	keine
APP0	Datei benutzt JFIF ([Hami92])	allgemeine Angaben zu den Folgedaten
DHT	Definition of Huffman Table	Huffman-Tabelle
DQT	Definition of Quantization Table	Quantisierungstabelle
SOF	Start Of Frame	Größe und Farbanzahl des Bildes
SOS	Start Of Scan	Zuordnung der Tabellen zu Komponenten, Beginn der Bilddaten

Tabelle 6: Marker in einer JPEG-Datei

Die Funktion `DecodeDC()` ermittelt den jeweils ersten Koeffizienten jeder Matrix, da diese differierend von den restlichen 63 Werten kodiert werden (z.B. nutzt man separate Huffman-Tabellen). Danach erfolgt mittels `DecodeAC()` die Lauflängen- und Huffman-Dekodierung der verbleibenden 63 Koeffizienten der bearbeitenden Matrix. Es folgt deren Umordnung entgegengesetzt zur während der Kodierung vorgenommenen zickzackförmigen Aufreihung in

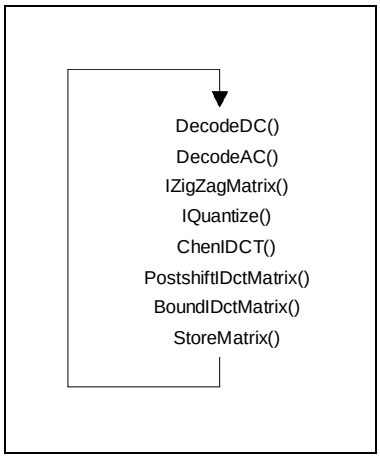

Abbildung 15: Etappen der Dekodierung innerhalb von DecodeStream()

Funktion `IZigZagMatrix()`. In `IQuantize()` wird nun die Dequantisierung der Matrix vorgenommen, bevor `ChenIDCT()` die Inverse Diskrete Cosinus-Transformation ausführt. Es handelt sich hierbei um einen sehr effektiven Festkomma-Algorithmus, der ähnlich wie die schnelle Fouriertransformation arbeitet. Da VSTa keine Gleitkomma-Unterstützung bietet, konnte an dieser Stelle nicht das gleiche Verfahren wie für den DSP implementiert werden. Im Anschluß daran werden die Matrixwerte um einen Betrag verschoben (`PostshiftIDctMatrix()`), auf Werte im Intervall [0, 255] begrenzt (`BoundIDctMatrix()`), und es erfolgt in `StoreMatrix()` die Speicherung der vollständig dekodierten Matrix im Zwischenspeicher `mxBuf`, der eine komplette horizontale Reihe Matrizen aufnimmt.

Sobald die letzte Matrix einer Reihe komplettiert wurde, übernimmt die Funktion `Write-Buffer()` deren Auslesen in den Speicher für das dekodierte Bild. Die Anfangsadresse dieses Speichers ist im Zeiger `*picture` abgelegt. Diese Umspeicherung ist erforderlich, da eine Matrixreihe pixelzeilenweise ausgelesen wird (vgl. Abbildung 16), d.h. zuerst werden die Reihen 0 der Matrizen (1,1) ... (1,m) ausgelesen, danach die Reihen 1 (in Abbildung 16 schraffiert dargestellt) usw. , zuletzt die Reihen 7.

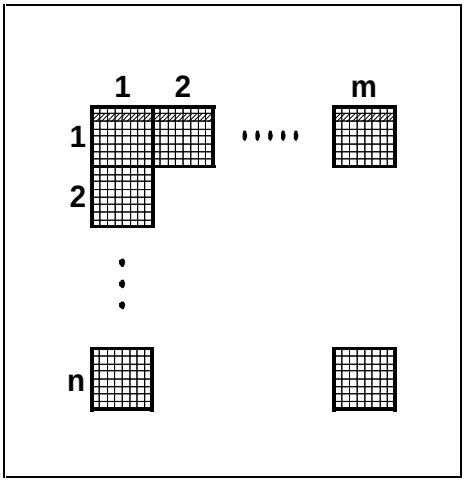

Abbildung 16: Partitionierung eines Bildes in Matrizen

Das physische Lesen aus dem Datenstrom übernimmt die Funktion `readb()`, die die als Argument übergebene Anzahl Bits aus dem Strom ausliest und den korrespondierenden Wert zurückliefert.

Weitere Details der Realisierung, z.B. zur Huffman-Dekodierung und Fehlerbehandlung, sind aus den kommentierten Quelltexten entnehmbar, die vollständig im Anhang aufgeführt sind.

4.3 DSP-gestützte Realisierung

4.3.1 Migration zur DSP-Variante

Die Analyse der gesamten Dekoderfunktionalität ergab, daß nicht alle Teile des Dekoders eine Auslagerung auf den DSP rechtfertigen. Die Algorithmen der Huffman- und Lauflängen-Dekodierung erfüllen die in Abschnitt 3.3.1 aufgeführten Kriterien für DSP-geeignete Algorithmen nicht. Sie bestehen zu einem großen Teil aus Suchoperationen (d.h. bedingten Sprüngen) über den Dekodertabellen, außerdem arbeiten sie mit Operanden variabler Länge. Darüber hinaus fallen kaum numerische Operationen an.

Die DSP-Software übernimmt außer den Funktionen zur Datenübertragung daher folgende Aufgaben:

– Dequantisierung

– Inverse DCT (Hauptanteil des Aufwands)

– Verschiebung und Begrenzung der rücktransformierten Werte

– pixelzeilenweises Auslesen der dekodierten Matrixreihe

Die Analyse des JPEG-Stroms, die Huffman- und Lauflängendekodierung sowie die Zickzack-Umordnung der Koeffzienten verbleibt auf Seiten des PCs.

4.3.2 Kommunikationsdienste

DSP und PC kommunizieren über die physisch 8 Bit breite Parallelschnittstelle des Host Interface (HI). Es wird logisch auf 8 aufeinanderfolgende Adressen im Portadreßraum des PC ab-

Funktion	Aufgabe (Sichtweise PC)
`to_host()`	Senden eines 24-Bit-Wortes zum DSP
`from_host()`	Lesen eines 24-Bit-Wortes vom DSP
`host_command()`	Auslösen einer Unterbrechung im DSP
`reset_board()`	Zurücksetzen der Einsteckkarte, Reset

Tabelle 7: Elementare Kommunikationsdienste

gebildet, der DSP verfügt über 3 entsprechende Steuerregister. Gemäß der Datenbusbreite des DSP werden (byteweise übertragene) 24-Bit-Worte über das Interface ausgetauscht.

Funktion	Aufgabe (Sichtweise PC)
`send_loader()`	Übertragung des Lademoduls zum DSP
`send_app()`	Interpretation einer LOD-Datei, Sendung ihres Inhalts zum DSP
`send_matrix()`	Übertragung einer Matrix zum DSP
`send_word()`	Übertragung eines einzelnen Wortes zum DSP
`receive_word()`	Empfang eines einzelnen Wortes vom DSP
`do_hc()`	Auslösen einer Unterbrechung im DSP

Tabelle 8: Komplexe Kommunikationsdienste

Innerhalb des PC wurden für die Kommunikation zum DSP 2 Software-Ebenen entwickelt. Die Funktionen der niederen Ebene sind für elementare Aufgaben der Kommunikation verantwortlich (Tabelle 7). Eine Zeitschranke überwacht die Terminierung der Funktionen in endlicher

Zeit (Timeout-Bedingung), für die Behandlung erkannter Fehler sind jedoch die Dienste der höheren Ebene verantwortlich. Des weiteren wurde der Zugriff auf die sogenannten Hostflags implementiert (`hfx_hi()` bzw. `hfx_lo()`), die in der aktuellen Version noch nicht zum Einsatz kommen. Die Datei `ll.c` beinhaltet die eben besprochenen Funktionen.

Die höhere Ebene der Kommunikation wird aus den Funktionen der Datei `hl.c` gebildet. Sie bietet unter Nutzung der elementaren Funktionen leistungsfähige Dienste für die Inter-Prozessor-Kommunikation zwischen Intel-CPU und DSP (Tabelle 8). Es existieren sowohl Funktionen zum Senden der kompletten DSP-Applikation, als auch zum Empfang und zur Sendung einzelner Worte. Die untersten 3 Funktionen der Tabelle sind Pendants der elementaren Dienste unter Einbeziehung der Fehlerbehandlung.

4.3.3 Lademodul des DSP

Im Abschnitt 3.2.3.1 wurde die Entwicklung eines separaten Lademoduls für den DSP motiviert. Der Server überträgt diesen während seiner Initialisierungsphase. Sobald diese Übertragung abgeschlossen ist, fragt der DSP kontinuierlich das Hostinterface ab, er erwartet die blockweise Sendung einer Applikation. Jeder Block hat das in Abbildung 17 dargestellte For-

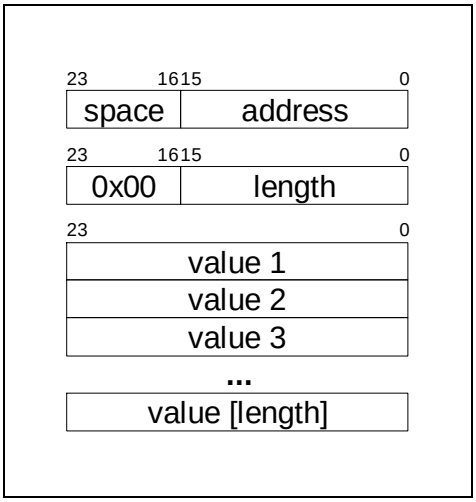

Abbildung 17: Format eines DSP-Datenblockes

mat. Das Feld *space* legt den Speicherraum (vgl. Tabelle 9) und das Feld *address* die Startadresse fest, ab der die Daten abzulegen sind. Der Wert *length* gibt die Länge des nun folgenden Datenblocks an. Die Übertragung selbst geschieht durch Polling der Eingangsregister des Host-Interface.

Nach beendeter Übertragung der Applikation sendet der DSP eine Quittung und beginnt, die Applikation zu bearbeiten.

Das funktionale Gegenstück auf PC-Seite ist die Funktion `send_app()`. Sie eröffnet die als Argument übergebene Datei, die eine DSP-Applikation im LOD-Format beinhaltet, analysiert

space	Bedeutung
1	Folgedaten im P-RAM ablegen
2	Folgedaten im X-RAM ablegen
3	Folgedaten im Y-RAM ablegen
0	Ende der Übertragung, Feld *adr* gibt Einsprungadresse an

Tabelle 9: Bedeutung des Feldes *space*

diese und sendet den Inhalt im oben beschriebenen Format zum DSP. Dateien dieses Formats bestehen aus ASCII-Text, sind damit zwischen heterogenen Rechnerplattformen austauschbar und gelten als Quasi-Standard bei der Software-Entwicklung für Motorola-Prozessoren. Eine detaillierte Beschreibung des Formats enthält [Arie90]. Mit der Funktion `send_app()` ist es möglich, zur Laufzeit des VSTa Applikationen an den DSP zu senden. Diese Applikationen müssen vor Start des Systems vorliegen, da unter VSTa keine Entwicklungswerkzeuge für den DSP existieren.

4.3.4 DSP-Software

Für die Implementierung der Inversen Diskreten Cosinus-Transformation wurde der Ansatz aus [Blak93] gewählt:

Wenn Z eine $N \times N$-Matrix DCT-transformierter Koeffizienten ist und C eine $N \times N$-Matrix, für deren Elemente c(k,n) die folgende Beziehung gilt (N=7 für JPEG)

$$c(k,n) = \begin{cases} \frac{1}{\sqrt{N}} & k = 0,\ 0 \le n \le N-1 \\ \sqrt{\frac{2}{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) & 0 \le k \le N-1,\ 0 \le n \le N-1, \end{cases}$$

dann erhält man die inverse cosinus-transformierte Matrix von Z mit der Beziehung

$$X = C^T Z C.$$

Die IDCT wird durch zwei Matrixmultiplikationen berechnet. Dieser Algorithmus ist nicht optimal, jedoch mit vertretbarem Aufwand zu implementieren und zu testen. Die Gleitkommaoperanden ersetzt man durch eine äquivalente Festkommadarstellung, indem alle Operanden mit 2^{24} multipliziert und am Ende wieder durch diesem Wert dividiert werden (auf die Division

verzichtet man, indem das höherwertige 24-Bit-Wort aus den 48 Bit langen Registern entnommen wird).

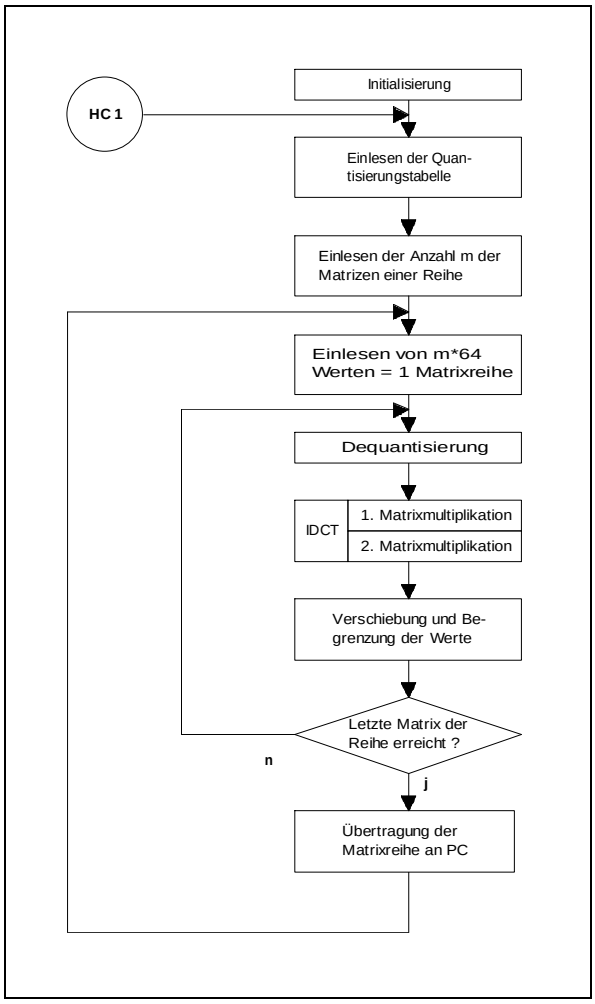

Abbildung 18: Programmfluß der DSP-Applikation

Den Programmfluß der DSP-Applikation zeigt Abbildung 18. Es wurde die einfachste Prozeßstruktur gemäß Abschnitt 3.2.3.2 realisiert, da Versuche mit komplexeren Modellen im vorgesehenen Zeitraum zu keinen funktionsfähigen Lösungen führten

Nach der Initialisierung, der Übertragung der Quantisierungsmatrix und der erforderlichen Puffergröße für eine Matrixreihe liest der DSP in einer Endlosschleife mittels Polling eine komplette Reihe Matrizen vom PC ein, verarbeitet diese und überträgt das Resultat wiederum per Polling zurück an den PC. Er übernimmt ebenfalls das zeilenweise Auslesen der Daten aus der bearbeiteten Matrixreihe.

Eine vom PC vorgenommene Unterbrechung (Host Command, HC1) veranlaßt den Signalprozessor, die Arbeit an einem neuen Bild fortzusetzen. Dieser Mechanismus besitzt den Vorteil, daß bei Dekodierungsfehlern des PC (z.B. fehlerhafte Huffmankodierung, verstümmeltes Bild)

eine Neusynchronisation des DSP leicht möglich ist und auf ein Rücksetzen der gesamten Einsteckkarte verzichtet werden kann.

Weitere Details zur Implementierung der DSP-Variante beinhaltet das kommentierte Quelltext-Listing im Anhang.

4.3.5 Befehlssatzbedingte Nachteile

Im Verlauf der Implementationsarbeiten offenbarten sich gewisse Beschränkungen in Befehlssatz und Architektur des DSP56001, die die Programmierung allgemeiner Aufgaben erschweren. Die wichtigsten Restriktionen sollen kurz angesprochen werden.

- Der DSP verfügt über keine In- und Dekrementbefehle für die Register (Ausnahme: Adreßregister). Die Operation muß statt dessen durch explizite Addition von ±1 realisiert werden, was 4 Befehlszyklen benötigt. Der Nachfolgetyp DSP56002 wurde um diese Funktionalität für beide Akkumulatoren ergänzt, die Eingangsregister X und Y wurden jedoch nicht berücksichtigt.

- Die Multiplikation von Festkomma-Werten erfordert infolge des gebrochenen Zahlenformats eine anschließende Verschiebungsoperation um 1 Bitstelle nach rechts (vgl. Abschnitt 2.2.1). Tabelle 10 beinhaltet ein Beispiel ([Moto88A]). Das bei der Multiplikation der in X0

Operand	Register	Inhalt (Hex)	dezimale Interpretation gebrochenes Zahlenformat	dezimale Interpretation Festkomma-Zahlenformat
Faktor	X0	000002	$2.3841858*10^{-7}$	2
Faktor	X1	000138	$3.7193298*10^{-5}$	312
Produkt	A	00:000000:0004E0	$8.8675733*10^{-12}$	1248

Tabelle 10: Beispiel zur Festkomma-Multiplikation

und X1 lokalisierten Faktoren resultierende Bitmuster im Akkumulator ist nur bei Interpretation im gebrochenen Zahlenformat korrekt; eine Interpretation im Festkommaformat ist um den Faktor 2 zu groß. Eine Schiebeoperation um 1 Bit nach rechts korrigiert das Ergebnis. Diese Besonderheit ist eine häufige Ursache für Programmierfehler.

- Schiebeoperationen sind pro Befehl nur um 1 Bitstelle möglich. Mehrfache Shifts müssen entweder durch eine Schleife oder besser durch eine Multiplikation mit einer Konstanten realisiert werden. Beispiel:

```
move #$080000,x1
```

```
mpy x0,x1,a
```

Dieses Fragment verlagert den Operanden aus Register X0 nach Akkumulator A und verschiebt ihn dabei um 4 Bits nach rechts ([Moto88A]).

- Der Aufwand zur Optimierung von Programmteilen ist verhältnismäßig hoch. Um die durch die Harvard-Architektur ermöglichte Parallelität der Datenzugriffe und arithmetischen Operationen zu nutzen, ist eine ausgefeilte Strategie der Operandenspeicherung notwendig. Erschwerend wirkt, daß arithmetische Operationen grundsätzlich nur in den beiden Akkumulatoren möglich möglich sind.

- Der 15*32 Bit große Stapelspeicher ist bei komplexer Programmstruktur (verschachtelte Interrupts, Schleifen und Unterprogrammaufrufe) schnell erschöpft. Es ist zwar möglich, einen Software-Stack zu realisieren, doch müssen dafür Register zweckentfremdet werden, der Aufwand ist zudem recht groß.

5 Leistungsmessung

Zur Beurteilung der Leistungsfähigkeit von Mikroprozessoren werden häufig Angaben über MIPS bzw. MOPS herangezogen, wobei MIPS für die durchschnittliche Zahl von abgearbeiteten Maschinenbefehlen pro Sekunde steht und MOPS die Anzahl Elementaroperationen pro Sekunde angibt.

Der DSP56001 kann im Extremfall bis zu 7 Operationen in einem Befehlszyklus ausführen, z.B. durch die wiederholte Abarbeitung des folgenden Befehls:

```
macr x0,y0,a     (r1)+,x0     (r4)-,y0
```

Dieser Maschinenbefehl realisiert die bereits im Abschnitt 2.2.1 erläuterte MAC-Operation, rundet darüber hinaus das Resultat, führt 2 Speicher-Register-Transfers aus und in- bzw. dekrementiert außerdem noch jeweils 1 Adreßregister. Die Abarbeitung des Befehls dauert 2 Taktzyklen, vorausgesetzt der angeschlossene Speicher benötigt keine Wartezyklen. Legt man eine Taktfrequenz von 20 MHz zugrunde, so erhält man rein rechnerisch für die wiederholte Abarbeitung obiger Programmzeile 70 MOPS und 10 MIPS. Diese Angaben sind jedoch für praktische Belange wertlos, da sie (wie im obigen Beispiel) gewöhnlich unter praxisfernen Annahmen getroffen werden, um für die Vermarktung des Prozessors vermeintlich günstige, möglichst hohe Zahlenwerte zu erreichen. Es ist daher nicht sinnvoll, auf Basis dieser Werte vergleichende Aussagen über die Leistungsfähigkeit von Prozessoren zu treffen.

Statt dessen sollen einige genauere Untersuchungen an praktischen Beispielen die Leistungsfähigkeit von Signal- im Vergleich zu Universalprozessoren ermitteln. Dabei wird als DSP-Vertreter der 56001 von Motorola, getaktet mit 20 MHz, zur Referenz eingesetzt. Der PC verfügte über einen mit 60 MHz betriebenen Pentium-Prozessor der Firma Intel und war mit 8 MByte RAM sowie 256kByte sekundärem Cache ausgerüstet.

5.1 Einfache Tests

Mehrere elementare Algorithmen sollen in diesem Abschnitt sowohl für die DSP- als auch für die Intel-Architektur implementiert und hinsichtlich Laufzeit und Speicherkomplexität verglichen werden. Die Datenübertragung zum DSP bleibt unberücksichtigt. Die Auswahl der Algorithmen umfaßt dabei sowohl signalverarbeitungstypische als auch nichtnumerische Probleme. Die Programmierung erfolgt ausschließlich in prozessorspezifischer Assemblersprache, um möglichst optimale Lösungen zu erhalten.

5.1.1 Meßmethode

Zur Ermittlung von Zyklen- und Befehlszahlen sowie weiterer Ereignisse bestimmter Codefragmente besitzt der Pentium die sog. ‘Model Specific Registers’ ([Ludl94A]). Das MS-DOS-Programm stat.exe des Autors Christian Ludloff gestattet es, beliebige Programme unter Nutzung dieser Register zu analysieren ([Ludl94B]). Für die durchzuführenden Messungen stand damit ein ideales Hilfsmittel zur Verfügung.

Da die zu vergleichenden Routinen sehr kurz sind, mußte für aussagekräftige Messungen deren Abarbeitung wiederholt werden. Der entsprechende Testrahmen organisierte eine zyklische Abarbeitung der zu testenden Routine sowie die ordnungsgemäße Beendigung des Programms per Interrupt des MS-DOS:

```
        mov ax,cs
        mov ds,ax
turn:
        ; insert code to be tested here
        dec CNT
        jnz turn
;
        mov ax,4c00h
        int21h
```

Der Wert CNT beträgt je nach Zeit für den Algorithmus 10^5 bis 10^7. Für CNT = 10^6 benötigt der leere Testrahmen auf dem P60 eine Zeit von t = 0.79 s (Mittelwert aus 10 Messungen).

Als Testumgebung fungierte MS-DOS 6.22, da hierbei der Intel Pentium im Real Mode betrieben werden kann, komfortable Entwicklungswerkzeuge (Assembler, Debugger) zur Verfügung standen und stat.exe nur unter MS-DOS lauffähig ist. Um die unvermeidliche Streuung der ermittelten Werte zu minimieren, wurden alle Messungen mehrfach wiederholt.

Aufgrund der einfacheren Struktur des DSP 56001 wurde für die Laufzeitermittlung hier einer analytischen Bestimmung der Zyklenzahlen nach [Moto90] der Vorzug gegeben. Einige Resultate wurden anschließend durch Messungen überprüft, um die Korrektheit der bestimmten Zyklenzahlen zu verifizieren.

Nach Abschluß der Messungen wurden die ermittelten Zeiten zusätzlich auf 90 MHz (Intel Pentium) bzw. 80 MHz Taktfrequenz (Motorola 56001) hochgerechnet, um Anhaltswerte für moderne Vertreter der Typen zu erhalten.

Es wurden ebenfalls Versuche durchgeführt, um den Einfluß der Cachespeicher des Intel Pentium zu eliminieren. Zu diesem Zweck wurde der Testrahmen um die Anweisung

```
invd
```

erweitert, die den internen Cachespeicher des Intel-Prozessors als ungültig deklariert ([Nels91]). Jedoch führte die Abarbeitung mittels Meßprogramm stat.exe zu einem Abbruch. Im Anschluß daran wurde der Befehl `invd` durch `wbinvd` ersetzt. Diese Instruktion veran-

laßt zusätzlich zum Invalidieren des internen Caches ein Zurückschreiben des externen Caches des Prozessors in den Hauptspeicher, was zu einer etwa 1000fach längeren Abarbeitungszeit für den Testrahmen führte. Das Meßprogramm stat.exe akzeptierte nun die Testroutinen, und die ermittelten Abarbeitungszeiten und Zyklenzahlen wiesen weitaus kleinere Schwankungen auf. Trotzdem führten die Versuche zu unsinnigen (d.h. viel zu großen) Resultaten der Abarbeitungszeiten, so daß auf weitere Messungen unter diesen Bedingungen verzichtet wurde. Es muß Folgearbeiten überlassen bleiben, den Grund für dieses Verhalten des Intel-Prozessors zu ermitteln.

Die erarbeiteten Routinen sind im Anhang vollständig aufgeführt. Beim Lesen der Assemblersyntax ist unbedingt zu beachten, daß Quelle und Ziel im Befehlsmnemonic beider Prozessoren jeweils vertauscht sind. Der Befehl für DSPs der Typenreihe 56000 wird stets mit

cmd source,destination

kodiert, währenddessen alle Prozessoren der Intel-x86-Reihe die Notation

cmd destination,source

erfordern.

5.1.2 Vergleich von Zeichenketten

Eine häufig anfallende Aufgabe innerhalb der Datenverarbeitung ist der Vergleich von Zeichenketten. Im vorliegenden Beispiel sollen die ASCII-Werte von aufeinander folgenden Bytes zweier Zeichenketten gleicher Länge miteinander verglichen werden. Groß- und Kleinschreibung von Buchstaben bleibt unberücksichtigt.

Der Intel-Prozessor verfügt zur Lösung dieses Problems über einen Spezialbefehl, der in Verbindung mit einer Hardware-Schleife zu einem sehr guten Ergebnis führt. Nach der Initialisierung der benötigten Zeiger- und Zählerregister ist nur noch die Anweisung

```
repz cmpsb
```

erforderlich. Ähnlich günstige Implementierungen sind für das Suchen von Werten innerhalb von Zeichenketten gegeben.

Für den DSP56001 ist der Zeichenkettenvergleich aufgrund der bedingten Sprünge eine ungünstige Aufgabe. Darüber hinaus sollten beide Operanden im gleichen Speicherbereich abgelegt werden, was maximale Parallelität der Zugriffe verhindert. Die Schleife zum Vergleich hat folgende Form:

```
do #LGTH,ENDCMP              ; organize loop with length of strings
sub x0,a     x:(r0)+,x0      ; compare and get next char of string 1
jeq ONWARD
enddo                        ; leave loop if operands are not equal
```

```
        jmp ENDCMP
ONWARD  move x:(r1)+,a              ; get next char of string 2
ENDCMP
```

Besonders nachteilig sind die Restriktionen bezüglich der vorzeitigen Beendigung einer do-Schleife durch das enddo-Statement (es darf nicht letzter oder vorletzter Befehl der Schleife sein), was Folge der Pipelinearchitektur des DSP ist. Wäre die Operation innerhalb der Schleife komplexer, so würde dieser Umstand nicht ins Gewicht fallen. Des weiteren kann nur das Subtrahieren zweier Werte und das Laden eines Folgeoperanden parallelisiert werden.

Vorausgesetzt, die zu vergleichenden Zeichenketten unterscheiden sich in der *n*-ten Stelle, so benötigt der DSP für die Aufgabe 18+8n Zyklen, die sich aus der Initialisierung, aus n-1 Schleifendurchläufen mit Vergleichsresultat 0 und einem Durchlauf ungleich 0 zusammensetzen.

Alle den Intel-Prozessor betreffenden Werte wurden durch Abfrage der MSR-Register innerhalb des Programms stat.exe ermittelt. Die Aktivitäten von Prozessoreinheiten zur Erhöhung der Verarbeitungsleistung, wie Cachespeicher, Translation Lookaside Buffer und V-Pipeline, führte zu einer recht hohen Streuung der Meßwerte. Aus diesem Grund wurden stets Mittelwerte über 5 Einzelversuche gebildet.

Prozessor	n=1		n=5		n=10		n=100	
	Zyklen	t_a [ns]	Zyklen	t_a [ns]	Zyklen	t_a [ns]	Zyklen	t_a [ns]
56001/20	26	1300	58	2900	98	4900	818	40900
56001/80		325		725		1225		10225
P60	23	380	28	470	66	1100	393	6550
P90		250		310		740		4370

Tabelle 11: Programmlaufzeiten bei Zeichenkettenvergleich

Tabelle 11 faßt die wesentlichen Meßergebnisse zusammen. Die Stelle, in der die Zeichenketten differieren, wird durch *n* bezeichnet, t_a verkörpert die Ausführungszeit für eine komplette Abarbeitung des Algorithmus, korrigiert um die Zeit für den leeren Rahmen. Der DSP ist klar unterlegen, er arbeitet mehr Zyklen ab und besitzt die längere Taktzykluszeit, dennoch fällt der Leistungsverlust noch moderat aus. Der Rechenzeitbedarf für größeres *n* erhöht sich für den Intel Pentium nicht linear, sondern langsamer. Ursache für dieses Verhalten dürften die Cachemechanismen des Prozessors sein.

Die DSP-Realisierung benötigt 11 Worte, also 33 Byte Programmspeicher, die Intel-Variante hingegen nur 16 Bytes.

5.1.3 Addition von Vektoren

Als mathematische Standardoperation soll zunächst die Addition von Vektoren mit *n* Komponenten untersucht werden. Alle Komponenten besitzen Festkomma-Darstellung und sind 16 Bit (Intel) bzw. 24 Bit (Motorola) lang.

Die Implementierung für den Intel Pentium besitzt folgende Form:

```
add_vektor:
             mov ax,[si]                    ; _A component
             mov bx,[si+2*CMPTS]            ; _B component
             add ax,bx
             mov [si+4*CMPTS],ax            ; _C component is result
             add si,02h
             loop add_vektor
```

Das Register SI dient als Zeiger zu den Elementen der Vektoren. Eine Komponentenaddition umfaßt 5 Befehlszyklen:

- Komponente A laden
- Komponente B laden
- Addition
- Komponente C (Summe) speichern
- Zeigerregister inkrementieren

Im Gegensatz zum Beispiel im Abschnitt 5.1.2 werden beide Speicherräume des DSP benutzt. Vektor A befindet sich im X-Speicher, Vektor B im Y-Speicher. Die DSP-Realisierung kann daher parallelisiert werden, darüber hinaus werden 3 verschiedene Zeiger für die Vektoren eingesetzt (r0 zeigt auf Vektor A, r1 auf Vektor B und r4 auf Vektor C):

```
do #CMPTS,ENDADD
move x:(r0)+,a1
add y0,a     y:(r4)+,y0
move a1,x:(r1)+
```

Damit sind nur noch 3 Befehlszyklen notwendig:

- Komponente A laden und Zeiger für A inkrementieren
- Addition und Komponente B für nächste Addition laden und Zeiger für B inkrementieren
- Komponente C speichern

Eine solche Programmstruktur erfordert, daß bei Eintritt in die eigentliche Verarbeitungsschleife die ersten Operanden (hier: Komponente aus B) bereits geladen sind, die Pipeline also initialisiert werden muß. Dieses Merkmal ist typisch für DSP-Algorithmen.

Die Verarbeitungsgeschwindigkeit wurde für 3, 8, 32 und 128 Komponenten pro Vektor ermittelt, die Bestimmung der Zyklenzahl beim Intel-Prozessor unterblieb aus Aufwandsgründen

jedoch. Der DSP benötigt allgemein für eine Addition zweier Vektoren mit je *n* Komponenten 14+6n Zyklen.

Prozessor	t_a [ns] für n=3	t_a [ns] für n=8	t_a [ns] für n=32	t_a [ns] für n=128
56001/20	1600	3100	10300	39100
56001/80	400	775	2575	9775
P60	2300	6200	6700	21400
P90	1500	4100	4500	14300

Tabelle 12: Programmlaufzeiten der Vektoraddition

Der Algorithmus zeigt für kleines *n* deutliche Vorteile für den DSP. Für n=3 käme DSP-seitig in der Realität eine lineare Programmierung zum Einsatz, die 26 statt der hier veranschlagten 32 Zyklen benötigt und damit noch eine kleine Beschleunigung erreicht. Vektorverarbeitung ist folgerichtig auch ein oft durch DSPs bearbeitetes Aufgabengebiet. Je größer *n*, desto stärker holt der Intel Pentium auf. Auch hier dürften der Cache und eventuell der Translation Look-aside Buffer (TLB) für das nichtlineare Verhalten verantwortlich sein. Besonders zu beachten ist der verschwindend geringe Rechenzeitanstieg beim Übergang von n=8 auf n=32. Es handelt sich hierbei um keinen Meßfehler ! Eine Analyse dieses Verhaltens muß aus Aufwandsgründen weiterführenden Arbeiten vorbehalten bleiben.

Der Ressourcenverbrauch an Programmspeicher ist fast identisch: die DSP-Routine wird in 27 Bytes, die Intel-Lösung in 23 Bytes kodiert.

5.1.4 Vektor-Matrix-Multiplikation

Ein eng verwandter Algorithmus ist die Multiplikation von Matrizen. An dieser Stelle soll nur der Spezialfall Vektor-Matrix-Multiplikation untersucht werden. Der Vektor besteht aus 3 Elementen, die Matrix aus 3 Zeilen und 3 Spalten ([1x3] * [3x3] = [1x3]), alle Elemente sind ganzzahlig. Der Algorithmus besitzt in der Computergrafik große Bedeutung, er dient zur Transformation von Pixelkoordinaten im dreidimensionalen Raum.

Der Quelltext der Implementierungen ist bereits etwas umfangreicher und daher nur im Anhang aufgeführt. Für den DSP wurde per linearer Programmierung die leistungsfähigste Variante realisiert. Die Berechnung eines Elements des Ausgangsvektors C erfolgt mit:

$$c_1 = a_1 b_1 + a_2 b_2 + a_3 b_3$$

und benötigt 5 Befehlszyklen:

```
mpy x0,y0,a     x:(r0)+,x0     y:(r4)+,y0
mac x0,y0,a     x:(r0)+,x0     y:(r4)+,y0
mac x0,y0,a     x:(r0)+,x0     y:(r4)+,y0
```

```
asr a
move a0,x:(r1)+
```

Die Schiebeoperation im 4.Befehl ist aufgrund der gebrochenen Zahlendarstellung des DSP erforderlich. Da Vektor A im X-Speicher und Matrix B im Y-Speicher lokalisiert sind, können parallel zu jeder arithmetischen Operation noch 2 Ladebefehle (sowie 2 Adreßregisterinkremente) ausgeführt werden. Für die Abarbeitung der vollständigen Routine erfordert der DSP 40 Zyklen.

Die Realisierung für den Intel Pentium erfordert dagegen eine viel größere Anzahl Transportoperationen, die aufgrund ihrer Abhängigkeit voneinander auch kaum von beiden Integer-Pipelines des superskalaren Prozessors parallel bearbeitet werden können. Das Analyseprogramm ermittelt ca. 330 Zyklen für die gesamte Multiplikation.

Entsprechend fällt der Vergleich der Laufzeiten aus:

Prozessor	t_a [ns]
56001/20	2000
56001/80	500
P60	5500
P90	3700

Tabelle 13: Programmlaufzeiten der Vektor-Matrix-Multiplikation

Der DSP besitzt klare Vorteile, selbst die 20-MHz-Variante ist fast doppelt so schnell wie der P90 von Intel. Die Intel-Implementierung benötigt 56 Bytes, die DSP-Realisierung 60 Bytes Speicher.

5.1.5 Digitales Filter

Innerhalb der Digitalen Signalverarbeitung nehmen Algorithmen zur Realisierung digitaler Filter breiten Raum ein ([Götz90], [John91] u.a.). Diese Filter sind gegenüber ihren analogen Pendants wesentlich flexibler, leichter zu entwerfen und weisen ein stabileres Betriebsverhalten auf. Dem steht die relativ niedrige Grenzfrequenz aller DSV-Systeme als Nachteil gegenüber. Der Aufschwung der Theorie digitaler Filter war ein wesentlicher Impuls zur Entwicklung spezieller Prozessoren für die digitale Signalverarbeitung, weshalb Filteralgorithmen auch stets hervorragend zur Abarbeitung auf Signalprozessoren geeignet sind.

Im folgenden soll ein sog. FIR-Filter für den DSP 56001 und den Intel 80x86 realisiert werden. Dieser Filtertyp zeichnet sich durch eine endliche Impulsantwort (finite impulse response) aus, d.h. nach der Einspeisung eines einzelnen Impulses in das Filter erreicht dessen Ausgangswert

nach endlich vielen diskreten Verarbeitungsschritten wieder den Wert 0. FIR-Filter werden beispielsweise eingesetzt, wenn exakt lineare Phasengänge notwendig sind.

5.1.5.1 Beschreibung des Algorithmus

Die Beschreibung des Algorithmus erfolgt anhand des Strukturdiagramms in Abbildung 19. Zu diskreten äquidistanten Zeitpunkten t wird jeweils ein Digitalwort x(t), welches beispielsweise durch einen vorgeschalteten A/D-Wandler generiert wird, in das System eingespeist. Die

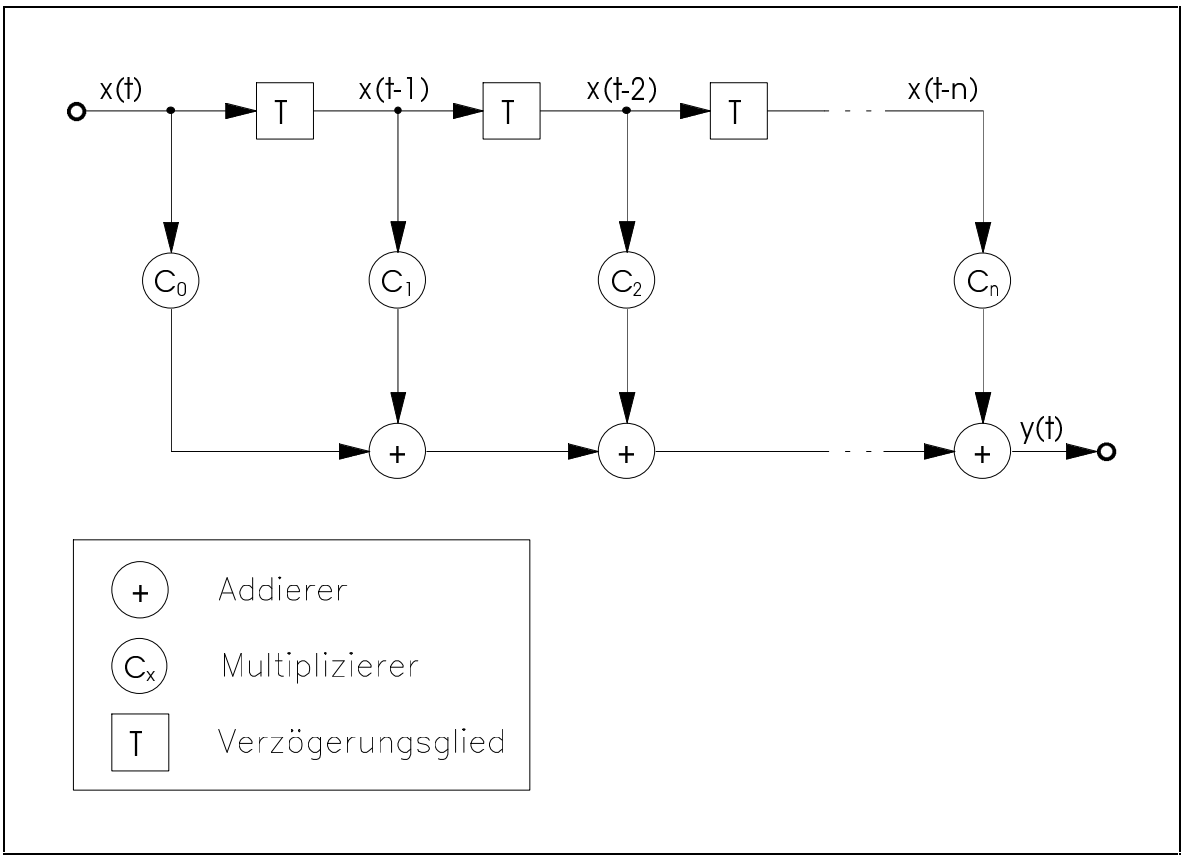

Abbildung 19: Strukturdiagramm eines FIR-Filters

Wortbreite soll im Beispiel 16 Bit betragen, andere Auflösungen sind ebenfalls denkbar. Das Wort x(t) wird nun zum einem mit dem Filterkoeffizienten C_0 multipliziert, zum anderen in ein Verzögerungsglied übertragen. An dessen Ausgang erscheint gleichzeitig das Digitalwort des vorangegangenen Zeitpunktes x(t-1), welches wiederum einerseits mit einem Filterkoeffizienten (C_1) multipliziert wird, andererseits dem folgenden Verzögerungsglied zugeführt wird. Der Vorgang wiederholt sich analog für alle weiteren (senkrechten) Stränge des Filters. Die Produkte aller Multiplikationen werden noch im gleichen Verarbeitungsschritt zum Ausgangswort y(t) des Zeitpunktes t aufaddiert. Damit gehören zu jedem Zeitpunkt t ein Eingangswort x(t) und ein gefiltertes Resultat y(t).

Mathematisch stellt der Algorithmus eine Aufsummierung von Produkten dar:

$$y(t) = x(t) \cdot C_0 + x(t-1) \cdot C_1 + \ldots + x(t-n) \cdot C_n$$

$$y(t) = \sum_{k=0}^{n} x(t-k) \cdot C_k$$

Die Anzahl der Stränge *n* (in Anlehnung an die schrittweise Verarbeitung häufig ‘Taps’ genannt) sowie die Werte der Koeffizienten determinieren das Verhalten des Filters (Cutoff-Frequenz, Signalabstand zwischen Durchlaß- und Sperrband, Linearität des Phasenverlaufs u.a.).

Jedes Verzögerungsglied nimmt jeweils ein Wort auf, so daß bei *n* Strängen des Filters die letzten n Eingangsworte x(t-1), x(t-2), ..., x(t-n) im Filter gespeichert sind. Zu jedem diskreten Zeitpunkt t wird ein aktuelles Eingangswort x(t) in die Verzögerungskette aufgenommen, die Worte x(t-1) bis x(t-(n-1)) rücken einen Schritt in der Kette weiter (in Abbildung 19 also nach rechts), und das letzte gespeicherte Wort x(t-n) verläßt den Speicher. Zur Organisation dieses Mechanismus wird gewöhnlich ein Ringpuffer genutzt.

Damit sind 2 Speicher mit der Größe *n* für den Algorithmus notwendig: ein Ringpuffer zur Aufnahme der Eingangsworte x(t) und der lineare Puffer der Koeffizienten C_x.

Für die Beispiel-Realisierung gelten folgende Festlegungen:

- Die Wortbreite aller Operanden beträgt 16 Bit. Der DSP verarbeitet 24-Bit-Worte, von denen das höchstwertige Byte ungenutzt bleibt, der 80x86 benutzt 16-Bit-Worte.
- Messungen werden für n=16, n=32 und n=64 Taps durchgeführt.
- Die Eingabeworte x(t) sowie die Resultate y(t) werden in der DSP-Variante von bzw. zu Speicherstellen der internen Peripherie übertragen, die angeschlossene Geräte (D/A- bzw. A/D-Wandler) symbolisieren. Diese Art der Anschaltung („memory-mapped“) ist Standard in DSP-Systemen.
- Die Eingabeworte x(t) sowie die Resultate y(t) der 80x86-Realisierung werden unter Nutzung der entsprechenden Maschinenbefehle an bzw. zu (symbolischen) Portadressen übertragen.
- Der DSP ist für jeden seiner Adreßräume mit statischem Speicher hinreichend kleiner Zugriffszeit ausgestattet, so daß Waitstates beim Speicherzugriff ausgeschlossen sind, was ebenfalls typisch für DSP-Hardware ist.

- Die Konfiguration des 80x86-Systems entspricht normaler PC-Hardware (dynam. Speicher, 70 ns Zugriffszeit).

- Programm und Daten für den 80x86 werden in einem gemeinsamen Segment lokalisiert, um Nachteile durch segmentierte Adressierung zu vermeiden.

- Es wird von der unterschiedlichen Zahlendarstellung beider Prozessortypen abstrahiert. Der DSP interpretiert Worte als Gleitkommawerte im Intervall $\{-1, 1-2^{23}\}$, der 80x86 verarbeitet vorzeichenlose Festkommawerte. In realen Systemen muß dieser Unterschied beachtet werden !

- Überläufe bei Multiplikation oder Addition bleiben unbehandelt, da sie bei korrekten Operanden nicht vorkommen.

- Ermittelt wird die Laufzeit des Programmfragmentes für die Filterung eines Eingangswertes: alle Register werden initialisiert, der Eingangswert x(t) wird in den Ringpuffer gelesen, ein Ausgangswert nach dem oben erläuterten Algorithmus ermittelt und ausgegeben sowie alle erforderlichen Register für einen erneuten Durchlauf aktualisiert.

Die erarbeiteten Lösungen für beide Prozessoren sind im Anhang vollständig aufgeführt.

5.1.5.2 Realisierung für DSP 56001

Bedingt durch die Hardware-Struktur des DSP, fällt das zugehörige Programm sehr kurz aus. Um maximale Parallelität im Programmfluß zu erreichen, befindet sich der Ringpuffer der Eingabewerte im X-Speicher, währenddessen die Koeffizienten im Y-Speicher abgelegt werden.

Register r0 fungiert als Zeiger in den Ringpuffer, wobei die DSP-Hardware selbständig für ein korrektes Rücksetzen sorgt, falls ein In- oder Dekrement die Puffergrenze überschreitet. Der Einfachheit halber ist der Speicher für die Koeffizienten ebenfalls als Ringpuffer organisiert, wodurch der entsprechende Zeiger (r4) nach einem kompletten Filterdurchlauf automatisch wieder auf der Anfangsadresse des Speichers steht.

Die Multiplikationen und Additionen werden durch den mac-Befehl realisiert, der in einem Befehlszyklus jeweils einen Eingangswert mit einem Koeffizienten multipliziert und das Ergebnis auf den Akkumulatorinhalt addiert. Gleichzeitig werden der nächste Eingabewert und der nächste Koeffizient aus dem Speicher geladen. Die letzte Multiplikation/Addition erfolgt separat, da das erhaltene Ergebnis zusätzlich gerundet wird, was eine DSP-spezifische Besonderheit ist und aus dessen Zahlendarstellung resultiert.

Der Code für einen Filterdurchlauf ist sehr kompakt, die benötigten Taktzyklen für jeden Befehl sind rechts angegeben:

```
movep y:INPUT,x:(r0)                                          4
clr a     x:(r0)+,x0                                          2

rep #TAPS-1                                                   4
mac x0,y0,a     x:(r0)+,x0     y:(r4)+,y0                     2*(n-1)
macr x0,y0,a     (r0)-                                        2

movep a,y:OUTPUT                                              4
```

Damit ist die Anzahl erforderlicher Taktzyklen pro Filterdurchlauf analytisch bestimmbar. Inklusive der notwendigen Initialisierung benötigt der DSP 56001 für ein Filter mit *n* Taps genau 22+2n Taktzyklen.

Für jede Zeile des Programms wird genau ein 24-Bit-Wort zur Kodierung benötigt. Die Filterschleife ist daher 6 Worte = 18 Bytes lang, die gesamte Routine einschließlich Initialisierung benötigt 10 Worte = 30 Bytes Programmspeicher.

5.1.5.3 Realisierung für Intel 80x86

Die Von-Neumann-Architektur des Intel 80x86-Prozessors bedingt eine Sequentialität aller Operationen. Die Speicherbereiche für Eingangswerte (IN_BUFFER) und Koeffizienten (COEFF_BUFFER) folgen direkt aufeinander. Das Register SI dient als Zeiger in den Ringpuffer der Eingangswerte. Das Zurücksetzen nach Über- oder Unterschreitung der Puffergrenze erfolgt recht effektiv durch eine binäre AND-Funktion, was jedoch nur möglich ist, wenn die Puffergröße eine Zweierpotenz ist und die Anfangsadresse (Offset) ein Vielfaches der Puffergröße ist (z.B. 0). Register BP zeigt in die Tabelle der Koeffizienten, die in diesem Fall keine Ringpuffer-Struktur besitzt, da der Aufwand hierfür ungleich höher ist als im DSP.

Da das Ergebnis einer Multiplikation immer im Registerpaar DX:AX abgelegt wird, andererseits auf jede Multiplikation eine Addition folgt, kann Register AX nicht als Akkumulator benutzt werden, Register DI übt statt dessen diese Funktion aus.

BX und AX dienen als Zwischenspeicher während der Rechenoperationen, CX organisiert die Hardware-Schleife mittels ‘loop’.

Der Code der Filterschleife besitzt folgende Form:

```
                mov dx,INPUT
                in ax,dx                        ; read in new sample from port
                mov [si],ax                     ; save sample in circ buffer

                lea bp,COEFF_BUFFER             ; bp points to coefficients
                xor di,di                       ; clear 'accu' di
calc_tap:
                mov bx,[si]                     ; read actual sample S
                add si,02h                      ; increment sample adr pointer
                and si,2*TAPS-1                 ; wrap around if neccessary
                mov ax,[bp]                     ; read actual coefficient C
                add bp,02h                      ; increment coeff adr pointer
                mul bx                          ; ax := S*C
                add di,ax                       ; di := di + S*C
                loop calc_tap

                mov dx,OUTPUT
                mov ax,di
                out dx,ax                       ; write out filtered sample
```

```
sub si,02h                      ; update adr for first element
and si,2*TAPS-1                 ; ... in circ buffer
```

Für die Ermittlung der Laufzeit mußten die in- und out-Befehle auskommentiert werden, da keine reale Hardware an diesen Adressen existierte.

Die innere Schleife benötigt 43 Bytes, inklusive Initialisierung werden 49 Bytes Programmspeicher benötigt.

5.1.5.4 Vergleich der Ergebnisse

Tabelle 14 beinhaltet die ermittelten Abarbeitungszeiten für die getesteten Parameter. Die Zyklenangaben für den Intel-Prozessor sind Meßwerte (stat.exe) und mit Unsicherheit behaftet.

Prozessor	n=16		n=32		n=64	
	Zyklen	t_a [µs]	Zyklen	t_a [µs]	Zyklen	t_a [µs]
56001/20	54	2.7	86	4.3	150	7.5
56001/80		0.675		1.075		1.875
P60	420	7.1	860	14.4	1560	26.1
P90		4.7		9.6		17.4

Tabelle 14: Programmlaufzeiten des FIR-Filters

Bei diesem Algorithmus kommen alle Vorteile der DSP-Architektur zum Tragen, da dieser, wie schon im Kapitel 2 erläutert, genau für diese Klasse Algorithmen optimiert wurde. Der hier erzielte Leistungsvorsprung (der DSP56001/20 bearbeitet die Aufgabe etwa 3-4 mal schneller als der P60) ist mit Ausnahme der FFT-Algorithmen der maximal erreichbare.

5.1.6 Iterative Berechnung

Schließlich soll ein iterativ arbeitender Algorithmus für beide Prozessorarchitekturen verglichen werden, wie er für numerische Methoden, z.B. der Nullstellenbestimmung, Verwendung findet. Konkret wurde ein Iterationsalgorithmus für eine populäre fraktale Grafik, das sog. Apfelmännchen, unter Nutzung der Festkomma-Zahlendarstellung programmiert. Er basiert auf folgender Iteration:

$$x(n+1) = x(n)^2 + c \qquad x, c \in C$$
$$x(0) = 0$$

Iteriert wird in der komplexen Zahlenebene für variables, diskretes c in den Grenzen

$$-2 \le \mathrm{Re}(c) \le 1$$
$$-1 \le \mathrm{Im}(c) \le 1.$$

Sobald der Betrag von x eine bestimmte Grenze überschreitet, bricht die Iteration ab, und das den Koordinaten von c entsprechende Pixel erhält eine mit der Iterationstiefe korrespondieren-

Prozessor	t_a [s]
56001/20	10.2
56001/80	2.5
P60	5.4
P90	3.6

Tabelle 15: Programmlaufzeiten der iterativen Berechnung

de Farbe. Wird die Grenze nach Abarbeitung einer maximalen Anzahl von Iterationen nicht überschritten, so bleibt der Bildpunkt dunkel. Das resultierende Bild besitzt die charakteristische Form ([PeSa88]).

Grundlage der Implementation war eine Assemblerroutine aus der Public Domain, die für die Registerstruktur des Intel i386 optimiert vorlag. Sie wurde um die für die Zeitmessung irrelevanten Funktionen zur Bildausgabe und Nutzerkommunikation reduziert. Anschließend erfolgte die Portierung für den DSP56001.

Infolge der Komplexität der Programmfolge war eine analytische Bestimmung der benötigten Zyklenzahlen unmöglich. Statt dessen wurden Messungen der Abarbeitungszeit für den P60 und den DSP56001/20 vorgenommen und danach für P90 und 56001/80 hochgerechnet (Tabelle 15).

Obwohl die Kodierung für den Pentium mehr Befehle erfordert, wird sie schneller abgearbeitet. Grund hierfür ist die DSP-seitige schlechte Parallelisierbarkeit von Lade- und arithmetischen Operationen: von 34 Befehlen können nur 10 mit gleichzeitigen Ladeoperationen ausgeführt werden. Trotzdem erzielt der DSP für diesen klar nicht auf seine Architektur zugeschnittenen Algorithmus ein respektables Ergebnis.

5.2 Komplexes Applikationsbeispiel

Der im Kapitel 4 detailliert erläuterte Decoder für das Standbildkodierungsverfahren JPEG fungierte als Demonstrationsbeispiel einer komplexeren DSP-Applikation. Die erzielten Leistungen sollen im folgenden verglichen und diskutiert werden.

5.2.1 Ermittlung der Verarbeitungszeiten

Die Dekodierung eines JPEG-Files erfolgt in den in Abbildung 20 veranschaulichten Phasen.

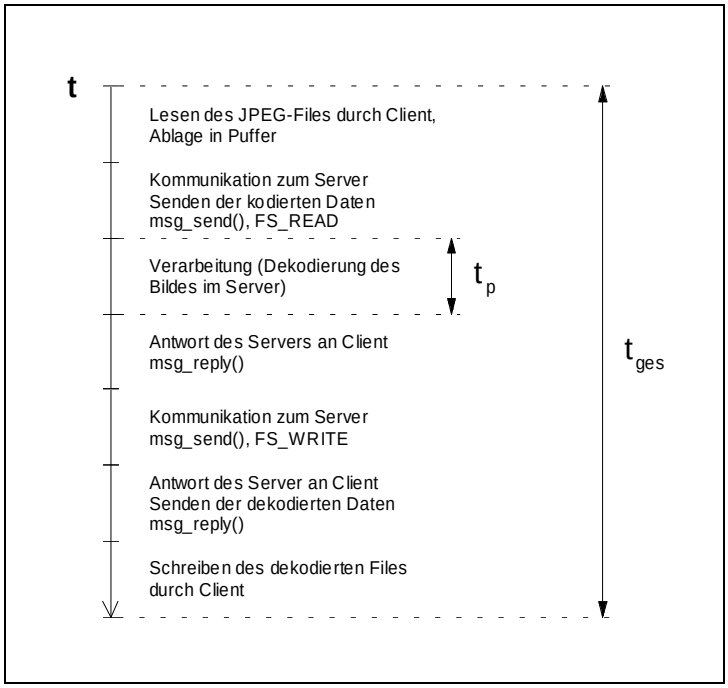

Abbildung 20: Phasen der Dekodierung eines Bildes

Für den Vergleich der Implementierungen sind jedoch nur die Zeiten für die Verarbeitung der zu dekodierenden Daten im Server ($t_{p,DSP}$ und $t_{p,PC}$) interessant, alle weiteren Phasen der Bearbeitung sind für beide Lösungen identisch.

Beide Realisierungen wurden zunächst mit einer Auswahl von JPEG-kodierten Bildern verschiedener Größe getestet und die über mehrere Aufrufe gemittelten Abarbeitungszeiten bestimmt ($t_{ges,DSP}$ bzw. $t_{ges,PC}$). Im Anschluß daran erfolgte eine analoge Bestimmung der Abarbeitungszeit t_{empty} für eine Servervariante ohne Dekodierungsfunktionalität (Verzicht auf Aufruf von decodejpg(), statt dessen Übertragung zufälliger Daten). Aus den Differenzen dieser Werte sind die Verarbeitungszeiten t_p beider Server zu errechnen:

$$t_{p,PC} = t_{ges,PC} - t_{empty}$$
$$t_{p,DSP} = t_{ges,DSP} - t_{empty}$$

In Tabelle 16 sind die entsprechenden Ergebnisse zusammengefaßt (die rein PC-basierte Realisierung benutzt den P60, die DSP-gestützte Variante setzt zusätzlich den 56001/20 ein).

Auflösung	Pixelanzahl	$t_{ges,PC}$ [s]	$t_{ges,DSP}$ [s]	t_{empty} [s]	$t_{p,PC}$ [s]	$t_{p,DSP}$ [s]
640x480	307200	4.1	14.0	3.9	0.2	10.1
768x512	393216	5.3	18.8	3.9	1.4	14.9
800x600	480000	6.2	22.8	5.4	0.8	17.4
1024x768	786432	11.3	37.8	8.4	2.9	29.4
1200x1024	1228800	18.0	59.0	13.3	4.7	45.7

Tabelle 16: Abarbeitungszeiten der Decoder-Varianten

Im Ergebnis zeigt sich ein gewaltiger Leistungsvorsprung der Intel-Pentium-Variante: sie arbeitet etwa um den Faktor 10 schneller als die Realisierung unter Zuhilfenahme des DSP.

5.2.2 Bestimmung des Kommunikationsaufwands zum DSP

Für eine genauere Beurteilung der DSP-Realisierung ist es angebracht, den Zeitaufwand für die Kommunikation zwischen DSP und Intel-Prozessor vom Aufwand für die DSP-seitige Verarbeitung separieren zu können. Die Gesamtverarbeitungszeit im Server $t_{p,DSP}$ kann gemäß Abbildung 21 in die Abschnitte

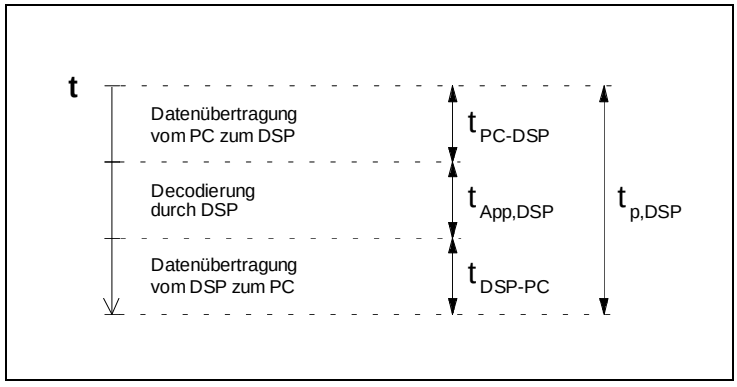

Abbildung 21: Phasen der Verarbeitung im DSP-Server

- Datenübertragung einer Reihe Koeffizientenmatrizen von der CPU zum DSP,
- Dekodierung der Reihe, d.h. Dequantisierung, IDCT, Verschiebung und Begrenzung der Koeffizienten,
- Datenübertragung der dekodierten Pixelinformationen vom DSP an die CPU

weiter gegliedert werden.

Die Zeit für die gesamte Kommunikation zwischen DSP und CPU (t_{comm}) ist offensichtlich:

$$t_{comm} = t_{PC-DSP} + t_{DSP-PC}$$

Mittels einer DSP-Applikation, die den Verarbeitungsteil ausspart, also die empfangenen Daten unbearbeitet zurücksendet, kann t_{comm} bestimmt werden. Da in diesem Fall $t_{App,DSP} = 0$ ist, gilt:

$$t_{comm} = t_{p,DSP}$$

Aus den Verarbeitungszeiten für die DSP-Variante mit Decoderfunktionalität ($t_{p,DSP}$) und den soeben ermittelten Zeiten für die Interprozessorkommunikation (t_{comm}) können abschließend die Zeiten für die reine DSP-Applikation (d.h. ausschließlich Dequantisierung, IDCT, Verschiebung, Begrenzung) errechnet werden:

$$t_{App,DSP} = t_{p,DSP} - t_{comm}$$

Tabelle 17 beinhaltet die nach dieser Methode ermittelten Zeiten. Beachtenswert ist die Tatsache, daß etwa 60% der gesamten Verarbeitungszeit des DSP ($t_{p,DSP}$) auf die Kommunikation zwischen den Prozessoren entfallen (t_{comm}), während die Abarbeitung der Applikation ($t_{App,DSP}$) die verbleibenden ca. 40% benötigt. Die Notwendigkeit einer stark verbesserten Kommunikation zwischen den Prozessoren für reale Anwendungen ist offensichtlich.

Bildauflösung	Pixelanzahl	t_{comm} [s]	$t_{App,DSP}$ [s]
640x480	307200	6.8	3.3
768x512	393216	9.0	5.9
800x600	480000	11.0	6.4
1024x768	786432	17.8	11.6
1200x1024	1228800	27.8	17.9

Tabelle 17: Verhältnis zwischen Kommunikations- und Verarbeitungsaufwand für DSP-Variante

Ferner gilt es zu bedenken, daß die Zeit für die reine Applikation $t_{App,DSP}$ allein durch Einsatz eines zeitgemäßen Signalprozessors (z.B. DSP56002 mit 80 MHz getaktet) auf ein Viertel reduziert werden könnte, womit dessen Leistung etwa die der Intel-P60-Variante erreichen würde.

5.3 Diskussion

Für die Beantwortung der eingangs der Arbeit aufgeworfenen Frage nach dem Sinn von DSPs innerhalb von Personalcomputern ist eine Interpretation der in den vorangegangenen Abschnitten des Kapitels durchgeführten Leistungsvergleiche notwendig.

Die im Abschnitt 5.1 untersuchten einfachen Applikationsbeispiele belegen, daß das Leistungsvermögen eines DSPs in hohem Maße vom Typ des bearbeiteten Algorithmus determiniert wird. Einerseits ist ein DSP durchaus in der Lage, bestimmte Algorithmen schneller als ein Universalprozessor abzuarbeiten. Andererseits existieren viele, besonders nichtnumerische Algorithmen, die von DSPs wesentlich ineffektiver als von Universalprozessoren bearbeitet werden. Offensichtlich eignen sich DSPs nicht als Ersatz für Universal-Mikroprozessoren.

Das komplexe Applikationsbeispiel verdeutlicht jedoch die zu erwartenden Probleme bei der Integration von DSPs in Betriebssysteme zur Entlastung der CPU von rechenzeitintensiven Aufgaben.

Die hauptsächliche Ursache für die schlechte Leistung der DSP-gestützten Variante ist die ungünstige Realisierung der Kommunikation zwischen DSP-Board und der CPU, die Folge hard- und softwareseitiger Kompromisse ist:

1. Es wurde innerhalb der Kommunikationsroutinen des DSP statisches Polling der Host-Register implementiert. Eine interruptbasierte Lösung wäre weitaus effektiver, da damit parallel zur Datenübertragung eine Verarbeitung zuvor empfangener Daten möglich ist. Die resultierende Prozeßstruktur des DSP ist aber sehr komplex und mittels der verfügbaren Entwicklungsumgebung schwer zu testen.

2. Die fehlende Signalisierungsmöglichkeit vom DSP zum PC verhindert innerhalb des VSTa-Servers eine effiziente Implementierung der Kommunikation, es kommt statt dessen ebenfalls Polling zum Einsatz.

3. Der Austausch von Daten geschieht über die langsame Portadressierung (in-, out-Maschinenbefehle) zum 8 Bit breiten Hostinterface des DSP. Diese Schnittstelle ist nur bedingt zum schnellen Austausch großer Datenmengen geeignet, ihre eigentliche Funktion ist der Austausch von Debugging- und Steuerungsinformationen zwischen Hostrechner und Signalprozessor. Abhilfe könnte hier ein modernerer DSP mit einer Hochgeschwindigkeitsschnittstelle, die direkt an den PCI-Bus des Hostrechners gekoppelt ist, schaffen.

4. Der realisierte JPEG-Decoder erfordert bidirektionale Übertragung von Daten (vgl. Abschnitt 3.2.3.3). Algorithmen mit ausschließlich unidirektionalem Datenfluß halbieren offensichtlich den Kommunikationsaufwand. Ein entsprechendes Beispielprogramm würde im

Vergleich zur DSP-losen Realisierung besser abschneiden. Es ist daher von außerordentlicher Bedeutung für die Akzeptanz von DSPs, A/D- und D/A-Wandler in das DSP-System zu integrieren.

Die reine Verarbeitungsleistung des DSP ist akzeptabel. Durch Nutzung eines schneller getakteten Prozessors (z.B. DSP56002/80, Preis ca. US$ 40) ist eine Steigerung bis auf das Vierfache leicht möglich. Außerdem ist unbedingt zu beachten, daß zur Realisierung der Inversen Diskreten Cosinus-Transformation nur für die DSP-lose Variante ein optimaler Algorithmus zum Einsatz kommt. Die für den DSP gewählte Implementierung auf Basis zweier Matrix-Multiplikationen besitzt ein schlechteres Laufzeitverhalten als der im PC realisierte Algorithmus nach Chen.

Generell ist festzustellen, daß die Unterstützung des DSP-Entwicklungssystems für das gewählte Zielbetriebssystem entscheidenden Einfluß auf die erreichbare Komplexität der zu erarbeitenden DSP-Software hat. Ohne effektive Möglichkeiten zum Test (interaktive Debugger etc.) können nur DSP-Projekte begrenzter Komplexität verwirklicht werden.

6 Zusammenfassung

6.1 Wesentliche Erkenntnisse

Die grundlegenden Erkenntnisse der vorliegenden Arbeit können in den folgenden Aussagen zusammengefaßt werden:

1. Digitale Signalprozessoren besitzen im Hinblick auf die Integration von Multimedia-Diensten wachsende Bedeutung für die Architektur künftiger Computersysteme, da sie hervorragend für die Bearbeitung digitalisierter Informationen geeignet sind.

2. DSPs sollten für eine universelle Nutzbarkeit mit geeigneter Wandlerhardware zu einem Subsystem innerhalb des Computers erweitert werden. Es müssen Prozessoren des aktuellen Entwicklungsstandes zum Einsatz kommen.

3. Die hardwareseitige Kopplung zwischen Computer und DSP-Subsystem muß so effektiv wie möglich erfolgen, insbesondere ist ein direkter Zugang zum Hauptspeicher des PC erforderlich. Ideal ist eine Kommunikation über shared memory, die Nutzung vergleichsweise langsamer serieller und paralleler Schnittstellen ist dagegen nicht sinnvoll.

4. Die softwareseitige Anbindung an microkern-basierte Betriebssysteme kann durch einen Serverprozeß erfolgen, der einen Gerätetreiber implementiert.

5. DSPs sind aufgrund ihrer Spezialisierung nicht für alle Algorithmen gut geeignet. Es können heuristische Regeln formuliert werden, die eine Beurteilung der Eignung von Algorithmen für die Abarbeitung auf DSPs erlauben.

6. Um den Aufwand für die Datenübertragung zu rechtfertigen, ist nur die Bearbeitung von rechenintensiven Applikationen sinnvoll.

7. Die starke Spezialisierung der Architektur von DSPs erfordert exakte Optimierung des Programmcodes, die kaum durch Softwarewerkzeuge übernommen werden kann.

8. Für die Implementation komplexer Projekte ist die Integration der DSP-Entwicklungsumgebung in das Zielbetriebssystem unumgänglich. Cross-Werkzeuge sind für diesen Einsatzzweck ungeeignet.

6.2 Ausblick

Abschließend soll ein kurzer Ausblick auf Fragestellungen gegeben werden, die direkt aus der vorliegenden Arbeit resultieren. Sie könnten Ausgangspunkte für Folgearbeiten zum Themenkreis „Integration von Signalprozessoren in moderne Betriebssysteme“ darstellen.

Als grundlegend für weitere Untersuchungen wird die Erarbeitung einer ‘konkurrenzfähigen’ Lösung unter Einbeziehung des DSP erachtet. Es sollte sowohl eine Erweiterung der Funktionalität des JPEG-Decoders als auch eine Verbesserung der Verarbeitungsleistung vorgenommen werden. Die im Kapitel 5 vorgenommene Analyse der Schwachstellen von Hard- und Software liefert dafür erste Anhaltspunkte.

Es sollte weiterhin untersucht werden, inwiefern die Einbeziehung eines DSP-spezifischen Echtzeit-Kerns zur Ressourcenverwaltung eine Erhöhung der Flexibilität des DSP-Systems bewirken kann.

Des weiteren sollte die Eignung weiterer DSP-Typen für die Integration in Computersysteme geprüft werden. Weiterführende Studien könnten danach versuchen, eine geeignete Abstraktion für allgemeine DSP-Systeme zu finden, die es gestattet, unterschiedliche DSP-Hardware in Computer zu integrieren, ohne Teile des Betriebssystems modifizieren zu müssen. Eine für den Nutzer transparente, von der aktuellen Ressourcensituation im Host abhängige Verlagerung von Funktionalität in den DSP wird damit ebenfalls möglich.

Ausgehend von der Analyse aktueller Multimedia-Algorithmen könnte eine Integration von Signalprozessor, lokalem Speicher, A/D- und D/A-Wandlern sowie weiterer Komponenten zu einem komplexen, modular konzipierten Subsystem zur Verarbeitung unterschiedlichster Datenströme erfolgen. Für die Anbindung eines solchen Systems an das Betriebssystem des Hosts wären neue Mechanismen erforderlich.

Glossar

Analog/Digital-Wandler: elektronische Schaltung zur Transformation einer kontinuierlichen Spannung in ein proportionales Digitalwort endlicher Länge (umgekehrte Richtung: Digital/Analog-Wandler)

Bit-reversed Addressing: spezielle Addressierungsart für Algorithmen der Schnellen Fouriertransformation, erspart das eigentlich notwendige Umsortieren der Operanden

Codec: (Akronym für Coder/Decoder) Hard- oder Software, die Kodierungs- und Dekodierungsfunktionalität kombiniert

Compander: (Akronym für Compressor/Expander) Klasse von Algorithmen der Digitalen Signalverarbeitung, die bei der Signalübertragung zur besseren Ausnutzung der Bandbreite des betreffenden Mediums eingesetzt werden

Eingebetter Prozessor (embedded processor): Nutzung eines Mikroprozessors mit minimaler Beschaltung für industrielle Steuer- und Regelapplikationen innerhalb von Geräten, für den Anwender i.allg. transparent

Filter: Hard- oder Software zur Selektion bzw. Unterdrückung bestimmter Komponenten eines Signals

Gebrochenes Zahlenformat: Interpretation eines Digitalworts als Gleitkomma-Wert im Intervall {-1,+1}, typisch für Algorithmen der Digitalen Signalverarbeitung

Harvard-Architektur: Separation der Speicherräume für Daten- und Programminformationen eines Mikroprozessors, typisches Merkmal von Digitalen Signalprozessoren

In-Circuit-Emulator: Gerät zum Testen von Mikroprozessorsoftware in komplexen Schaltungen, das Verhalten des zu testenden Prozessors und wird über einen separaten Rechner bedient.

MAC-Einheit: Baugruppe von Signalprozessoren zur hardwareunterstützten Aufsummierung von Produkten

Microcontroller: für industrielle Steuer- und Regelapplikationen in eingebetteten Umgebungen entwickelter, preiswerter Mikroprozessor

Polling: aktives zyklisches Abfragen eines Registers oder einer Speicherstelle durch den Prozessor

Neuronale Netze: Klasse von Algorithmen, die stark abstrahierte Kommunikations- und Adaptionsmechanismen von Nervenzellen (Neuronen) simulieren

Waitstate: Warten des Prozessors auf die Beendigung eines Speicherzyklus beim Zugriff auf langsamen dynamischen Speicher

Watchdog: elektronische Schaltung, die beim Ausbleiben eines regelmäßigen Signals nach Verstreichen einer bestimmten Zeitspanne einen Alarm auslöst, beispielsweise zur Überwachung von Mikroprozessoren genutzt

Literaturverzeichnis

[AD89] Analog Devices Incorporated: *ADSP-2100 Family Applications Handbook.* Vol.1-3. Norwood, MA, 1989

[AD95] Analog Devices Incorporated: *ADSP-2106x SHARC User's Manual.* Norwood, MA, 1995

[Alth93] Althaus, Martin: *Das neue PC Profibuch.* Düsseldorf: Sybex, 1993

[Arie90] Ariel Corporation: *Motorola DSP56000 Macro Assembler/Linker/Librarian : Reference Manual.* Highland Park, NJ, 1990

[Arie91] Ariel Corporation: *BUG-56 Monitor/Debugger for Ariel's DSP 56001 Board.* (Softwarebeschreibung). Highland Park, NJ, 1991

[Blak93] Blake, Steven L.: *Notes on the Discrete Cosine Transform.* Posting im Internet-Forum comp.dsp, 24.11.1993

[Brum89] Brumm, Penn ; Brumm, Don: *80386 : Das Handbuch für Programmierer und Systementwickler.* Haar bei München: Markt & Technik, 1989

[Brüs92] Brüse, Claudius: *Die Ohren des Falken : Audio und der Falcon 030.* In: Keyboards 10(1992), S. 46-55

[Buck93] Buck, Jürgen: *Hilfreicher Verlust : Der JPEG-Algorithmus.* In: mc 6(1993), S. 94-101

[Gail95] Gailly, Jean-Loup: *comp.compression FAQ.* Internet-Dokument, z.B. rtfm.mit.edu, /pub/usenet/news.answers, 1995

[GBK93] Gunzinger, Anton ; Bäumle, Bernhard ; Kohler, Peter: *MUSIC - ein Personal-Supercomputer.* Informationsmaterial der ETH Zürich, Institut für Elektronik, 1992

[Götz90] Götz, Hermann: *Einführung in die digitale Signalverarbeitung*. Stuttgart: Teubner, 1990

[Hami92] Hamilton, Eric: *JPEG File Interchange Format, Version 1.02*. Standardvorschlag, Milpitas, CA, 1992

[Heue90] Heuer, Gert: *Digitale Signalprozessoren*. In: Mikroprozessortechnik, Verlag Technik Berlin, 8(1990), S.4-9

[Holt94] Holtorf, Klaus: *Das Handbuch der Grafikformate*. München: Franzis, 1994

[Hung93] Hung, Andy C.: *PVRG-JPEG CODEC 1.1*. (Softwaredokumentation), Stanford, 1993

[Jesk95] Jeske, David: *Frequently Asked Questions About VSTa.* Internet-Dokument, z.B. ftp.cygnus.com:pub/embedded/vsta, 1995

[John91] Johnson, Johnny R.: *Digitale Signalverarbeitung*. München: Hanser, 1991

[Kesy93] Kesy, Oliver: *Bilder schrumpfen : Neue Methoden und Programme zur Bildkompression*. In: c't 11(1993), S.120-126

[KrSc93A] Krauß, Matthias ; Scheidt, Walter: *Klangwunder : DSP-Programmierung Teil 1*. In: c't 9(1993), S.226-229

[KrSc93B] Krauß, Matthias ; Scheidt, Walter: *Klangwunder : DSP-Programmierung Teil 2*. In: c't 10(1993), S.212-216

[KrSc93C] Krauß, Matthias ; Scheidt, Walter: *Klangwunder : DSP-Programmierung Teil 3*. In: c't 11(1993), S.260-264

[Lane95] Lane, Tom: *JPEG image compression FAQ.* Internet-Dokument, z.B. rtfm.mit.edu, /pub/usenet/news.answers, 1995

[LaWe95] Lapsley, Phil ; Weller, Franz: FAQs (Frequently asked questions with answers) on Digital Signal Processing. Internet-Dokument, z.B. http://www.bdti.com/dsp_faq.htm, 1995

[Ludl94A] Ludloff, Christian: *Zwischen den Zeilen : 'Intel inside'-Wissen zur Programmoptimierung.* In: c't 11(1994), S.266-270

[Ludl94B] Ludloff, Christian: *Informationen zu STAT und INFO.* Softwaredokumentation, Chemnitz, 1994

[MaEw93] Marven, Craig ; Ewers, Gillian: *A simple approach to Digital Signal Processing.* Oxford: Alden Press, 1993

[Merz94] Merz, Thomas: *Gut verpackt : Drucken von JPEG-Bildern mit PostScript Level 2*. In: c't 6(1994), S.236-243

[Meye82] Meyer, Gernot: *Digitale Signalverarbeitung.* Berlin: VEB Verlag Technik, 1982

[Moto88A] Motorola Incorporated: *Fractional and Integer Arithmetic Using the DSP56000 Family of General-Purpose Digital Signal Processors*. Phoenix, AZ, 1988

[Moto88B] Motorola Incorporated: *56-Bit General Purpose Digital Signal Processor Advance Information*. Phoenix, AZ, 1988

[Moto88C] Motorola Incorporated: *Cascadable-Adaptive Finite-Impulse-Response (CAFIR) Digital-Filter Chip Advance Information.* Phoenix, AZ, 1988

[Moto90] Motorola Incorporated: *DSP56000/56001 Digital Signal Processor User's Manual*. Phoenix, AZ, 1990

[Nels91] Nelson, Ross, P.: *Microsoft's 80386/80486 programming guide.* Redmont, WA: Microsoft Press, 1991

[PeSa88] Peitgen, Heinz-Otto ; Saupe, Dietmar (ed.): *The Science of Fractal Images.* New York: Springer, 1988

[RCN92] RCN: *DSPC56-Handbuch.* Berlin, 1992

[RePa92] Reid, Christopher ; Passin, Thomas: *Signal processing in C.* New York: Wiley & Sons, 1992

[Saue91] Sauer, Horst: *Bilder in die Zange genommen*. In: CHIP special. Animation auf dem PC, Würzburg: Vogel, 1991

[Spec94] Spectron Microsystems Incorporated: *SPOX - The DSP operating system : A Technical Overview.* Santa Barbara, CA, 1994

[Spec95] Spectron Microsystems Incorporated: *IA-SPOX : Native Signal processing Software Development Kit.* Informationsmaterial, Santa Barbara, CA, 1995

[SPL92] Steward, Lawrence C. ; Payne, Andrew C. ; Levergood, Thomas M.: *Are DSP Chips Obsolete?*. Cambridge, MA: Cambridge Research Laboratory, 1992, - Technical Paper

[Tane92] Tanenbaum, Andrew S.: *Modern Operating Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1992

[TI86] Texas Instruments Incorporated: *Digital Signal Processing Applications.* Houston, TX, 1986

[Vale95] Valencia, Andrew: *An Overview of the VSTa Microkernel.* Internet-Dokument, z.B. ftp.cygnus.com: pub/embedded/vsta, 1995

[Wall91] Wallace, Gregory K.: *The JPEG Still Picture Compression Standard*. In: Communications of the ACM 4(1991), S.30-44

Selbständigkeitserklärung

Hiermit erkläre ich, daß ich diese Arbeit nur mit den zugelassenen und aufgeführten Hilfsmitteln und ohne fremde Hilfe erstellt habe.

Hohenstein-Ernstthal, den 01.11.1995 Robert Baumgartl