Großer Beleg

zu dem Thema

Audio-MPEG-Decoder für DSP

TECHNISCHE UNIVERSITÄT DRESDEN
FAKULTÄT INFORMATIK
INSTITUT BETRIEBSSYSTEME, DATENBANKEN UND RECHNERNETZE

LEHRSTUHL BETRIEBSSYSTEME

eingereicht von:	Kai Stuhlemmer
geboren am:	10. Mai 1971 in Cottbus
Betreuer:	Dipl.-Inform. Robert Baumgartl
Betreuender Hochschullehrer:	Prof. Dr. H. Härtig
eingereicht am:	2. Juli 1997

Inhaltsverzeichnis

1 Einleitung **3**

2 Grundlagen und Stand der Technik **5**

2.1 Der Standard ISO/IEC 11172-3 5

2.1.1 Prinzip der Kodierung 6

2.1.2 Die Dekodierung 8

2.2 Das Zielsystem 10

2.3 Stand der Technik 10

2.3.1 Aktuelle Entwicklungstendenzen 10

2.3.2 Vergleichbare Arbeiten 11

3 Entwurf **12**

3.1 Entwicklungsetappen 12

3.2 C versus Assembler 13

3.3 Schnittstelle zum Betriebssystem 13

4 Ausgewählte Details der Implementation **15**

4.1 Aufrufkonventionen für Funktionen 15

4.2 Das Synthese-Filter 15

4.3 Speicherorganisation 20

4.4 *Degrouping* 20

5 Leistungsbewertung **22**

5.1 Meßmittel 22

5.2 Definition der Echtzeitanforderung 22

5.3 Echtzeitanforderung erfüllt? 23

5.3.1 Einfluß der Natur des Audio-Signals 23

5.3.2 Einfluß der Bitrate 24

5.3.3 Einfluß der Speicherbelegung 24

5.3.4 Einfluß konkurrierender DMA-Prozeße 24

6 Ergebnisse und Zusammenfassung **25**

Kapitel 1

Einleitung

Am Wendepunkt von der Industriegesellschaft zur Informationsgesellschaft stellen leistungsfähige Kommunikationsmittel eine wichtige Schlüsselkomponente dar. In diesem Zusammenhang wandelt sich der klassische Rechner in immer stärkeren Maße vom reinen „Rechenknecht“ zum Kommunikationsmittel. Immer mehr Menschen bedienen sich der *EMail* und „surfen“ im *World Wide Web*. Aber auch visuelle und auditive Informationskanäle spielen eine immer größer werdende Rolle. Anwendungen, wie z.B. Videokonferenzen oder digitale Bild- und Tonbearbeitung, sind längst etabliert. Die Integration solcher Anwendungen in Rechnersysteme stellen neue hohe Anforderungen an *Hardware* und *Software*. Herkömmliche Rechner und Betriebssysteme können diese Anforderungen nur unzureichend erfüllen, weshalb nach neuen Ansätzen und Ideen gesucht werden muß.

Neben der Entwicklung geeigneter, echtzeitfähiger Betriebssysteme ist auch die Erweiterung der *Hardware* um spezielle Subsysteme ein möglicher Schritt, um den neuen Anforderungen gerecht zu werden. Ein solches Subsystem kann dedizierte rechenintensive Aufgaben übernehmen und das Hauptsystem entlasten. Eine große Gruppe dieser Aufgaben besteht in der Verarbeitung von Bild- und Tonsignalen. In diesem Zusammenhang sind Digitale Signalprozessoren (DSP) gut für die Lösung solcher Aufgaben geeignet, da sie speziell auf Algorithmen der Signalverarbeitung zugeschnitten sind.

Das Ziel dieser Arbeit war die Realisierung einer Signalverarbeitungsaufgabe mit zeitkritischen Anforderungen auf einem DSP-Subsystem. Konkret sollte ein Dekoder für das MPEG-1-Audio-Format implementiert und auf seine Echtzeitfähigkeit hin untersucht werden.

Die Ergebnisse der Arbeit sind in diesem Papier niedergelegt und erläutert. In Kapitel 2 wird der Leser mit den nötigen Grundlagen vertraut gemacht. Kapitel 3 befaßt sich mit allgemeinen Fragen des Entwurfs. Auf einige Aspekte der Implementation wird im Kapitel 4 eingegangen. Inhalt des Kapitels 5 ist eine Analyse der erzielten Leistungsfähigkeit des Dekoders. Das Kapitel 6 zieht ein Fazit und faßt die Ergebnisse der Arbeit noch einmal zusammen.

Kapitel 2

Grundlagen und Stand der Technik

2.1 Der Standard ISO/IEC 11172-3

ISO/IEC 11172-3 [4] ist der dritte Teil eines vier Teile umfassenden Standards zur Kodierung von Video-Strömen mit zugehörigem Ton, welcher besser bekannt ist als MPEG-1. In dem hier interessierenden Teil wird die Kodierung von Audio-Signalen beschrieben. Der Standard unterstützt PCM-Signale mit Abtastraten von 32, 44.1 und 48 kHz. Es ist die Kodierung von Mono-, Stereo- und Zweikanal-Signalen möglich.

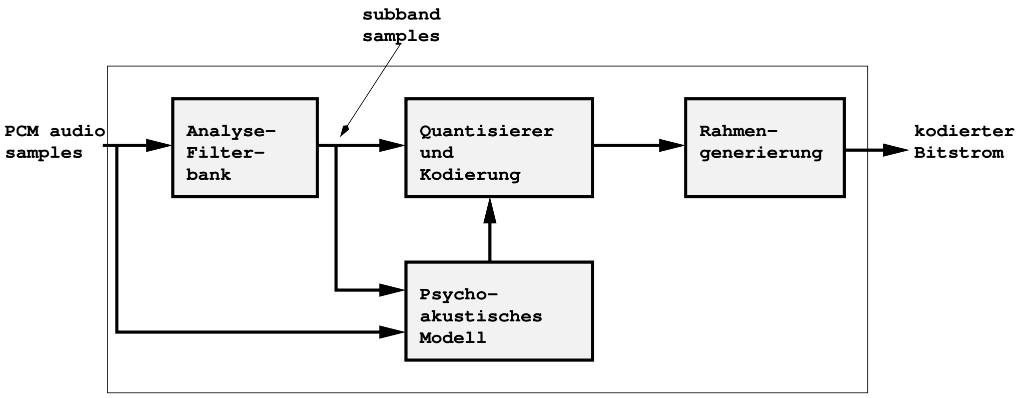
Das kodierte Signal ist ein in Rahmen gegliederter Bit-Strom mit einer festen Datenrate. Diese liegt zwischen 32 und 448 kBit/s. An dieser Stelle ist zu bemerken, daß ausschließlich der Aufbau dieser Rahmen, sowie deren Dekodierung durch den Standard definiert wird. Der Aufbau eines Koders ist nicht festgelegt. Es wird lediglich gefordert, daß das von ihm erzeugte Format dem Standard gemäß dekodiert werden kann. Der Standard beschreibt mögliche Koder, um die der Kompression zugrundeliegenden Ideen zu erläutern. Diese Beschreibungen haben aber keinen verbindlichen Charakter.

Definiert werden drei verschiedene Formate (Schicht I, II, III), welche sich hinsichtlich der Komplexität des Kodier/Dekodier-Prozesses, aber auch in der bei vergleichbarer Güte erreichbaren Kompression unterscheiden:

- **Schicht I** ist das einfachste Format und ist für Bitraten über 128 kBit/s pro Kanal geeignet. Ein Anwendungsbeispiel ist die *Digital Compact Disk* von Philips.

- **Schicht II** ist etwas komplexer und ist gut geeignet für Bitraten um 128 kBit/s. Sie findet z.B. Anwendung bei *Digital Audio Broadcasting*.

Abbildung 2.1: Aufbau eines MPEG-1 Audio-Koders



- **Schicht III** ist die mit Abstand komplexeste Kodierung und erreicht auch bei einer Bitrate von 64 kBit/s pro Kanal noch eine gute Signalqualität. Sie ist deshalb gut für Übertragungen über Schmalband-ISDN geeignet.

Da die Schicht III nicht Gegenstand dieser Arbeit ist, wird auf sie im weiteren Verlauf nicht mehr eingegangen.

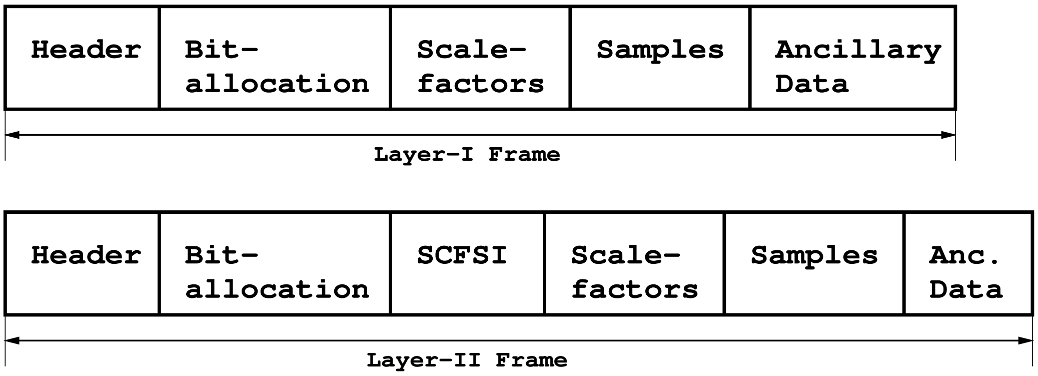
2.1.1 Prinzip der Kodierung

Das Kodierverfahren baut auf Wahrnehmungseigenschaften des menschlichen Gehörs auf. Es macht keine Annahmen über die Natur des akustischen Signals, wie z.B. das „*vocal tract*"-Modell, welches bei der Kodierung von Sprache Anwendung findet. Das Verfahren ist verlustbehaftet, d.h. das Orginalsignal kann im allgemeinen nicht wieder vollständig reproduziert werden. An dieser Stelle kann der Kodierprozeß nur grob umrissen werden. Für den interessierten Leser sei auf [4, 6, 5] verwiesen.

In Abbildung 2.1 ist der Aufbau des Koders skizziert. Der zu kodierende *PCM*-Datenstrom wird sowohl der Analyse-Filterbank, als auch dem Psychoakustischen Modell zugeführt. Die Filterbank teilt das Audio-Signal in 32 äquidistante Frequenzbänder (*Subbands*) auf. Die Berechnung wird für jeweils 32 neue *PCM samples* durchgeführt. Als Ergebnis entsteht für jedes Frequenzband ein *subband sample*, ingesamt also eine Gruppe von 32 *subband samples*. In einem Rahmen werden jeweils 12 Gruppen für Schicht I bzw. 3×12 Gruppen für Schicht II dieser 32 *subband samples* kodiert.

Das Psycho-Akustische Modell wird einmal pro Rahmen berechnet, also für alle 384 bzw. 1152 *Samples*. Es analysiert das Frequenzspektrum des

Abbildung 2.2: Aufbau eines Rahmen bei Schicht I und II



Audiosignals und berechnet für jedes der 32 Frequenzbänder einen Verhälniswert (*Signal-to-mask ratio*, *SMR*). Die Berechnung dieser Werte stützt sich auf zwei Eigenschaften des Gehörs:

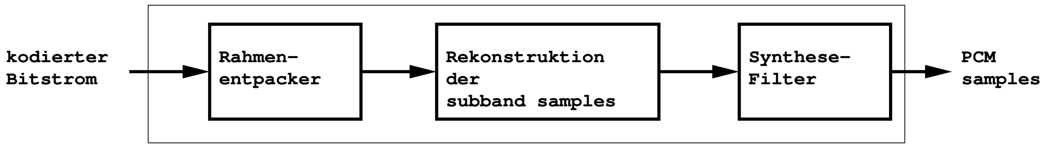
1. Das menschliche Gehör besitzt eine untere Wahrnehmungsschwelle, d.h. Signale mit einem Schalldruck unterhalb dieser Schwelle werden nicht gehört. Die Wahrnehmungsschwelle ist frequenzabhängig und steigt besonders bei hohen Frequenzen stark an.

2. Starke Audiosignale überdecken in einer bestimmten spektralen Umgebung schwächere Signale, d.h. es wird nur das starke Signal wahrgenommen.

Der Quantisierer hat die Aufgabe die Werte für die einzelnen Frequenzbänder mit so wenig Bits wie möglich zu quantisieren. Da eine geringere Anzahl der Quantisierungsstufen mit einem erhöhten Quantisierungsrauschen verbunden ist, steht als Nebenbedingung, daß dieses Rauschen möglichst nicht wahrnehmbar ist. Der Quantisierer stützt sich hierbei auf die Verhältniswerte, welche das Psycho-Akustische Modell liefert. Die Festlegung der Quantisierungsstufen für die einzelnen Bänder stellt einen Minimierungsprozeß dar, wobei die in einem Rahmen zur Verfügung stehenden Bits so verteilt werden, daß das über das gesamte Frequenzspektrum hinweg hörbare Quantisierungsrauschen minimal wird. Angestrebt wird also, daß das Rauschen nicht wahrnehmbar ist. Bei hoher Kompressionsrate (geringer Bitrate) kann dieses Ziel nicht immer erfüllt werden. Erwähnt werden soll noch, daß für einzelne Frequenzbänder keine *subband samples* kodiert werden müßen, wenn diese keinen wahrnehmbaren Beitrag am Gesamtsignal liefern.

Sind die Quantisierungsstufen ermittelt, werden die einzelnen *subband samples* durch einen Skalierungsfaktor geteilt und quantisiert. Der Skalie-

Abbildung 2.3: Aufbau des MPEG-1 Audio-Dekoders



rungsfaktor sorgt hierbei für die Ausschöpfung des zur Verfügung stehende Wertebereichs. Er wird so gewählt, daß er der kleinste verfügbare Wert aus einer im Standard definierten Tabelle (B.1) ist, welcher größer ist als das absolute Maximum in der Gruppe der *subband samples*.

Abschließend wird der Rahmen generiert. Der Aufbau des Rahmens ist für die Schichten I und II in Abbildung 2.2 dargestellt, und wird im folgenden Abschnitt im Zusammenhang mit der Dekodierung noch näher erläutert.

2.1.2 Die Dekodierung

In Abbildung 2.3 ist der prinzipielle Aufbau des Dekoders skizziert. Die Dekodierung beginnt mit der Synchronisation. Jedem Rahmen ist zu diesem Zweck eine Synchronisations-Bitfolge vorangestellt. Erleichtert wird diese Synchronisation dadurch, daß im Standard ein *Alignment* für den Beginn eines Rahmens festgelegt ist. Rahmen der Schicht I sind *word aligned* (32 Bit) und Rahmen der Schichten II, III sind *byte aligned*.

Ist der Beginn eines Rahmens gefunden, so folgt das Lesen und die Interpretation der einzelnen Teile (vgl. Abb. 2.2):

Header - enthält neben den schon erwähnten Synchronisationsbits Informationen administrativer Art, wie z.B. die verwendete Schicht, Modus (*single channel*, *stereo* usw.), Bitrate, *Sampling*-Frequenz. Dem *Header* kann optional eine Prüfsumme folgen. Dieser Teil des Rahmens ist für alle drei Schichten identisch.

Bitallocation - gibt Auskunft, mit wieviel Bits die Werte der einzelnen Frequenzbänder kodiert sind. Während bei Schicht I die Information direkt kodiert ist, dienen die bei Schicht II übertragenen Werte als Indizes in eine von vier im Standard definierten Tabellen. Um welche Tabelle es sich konkret handelt, hängt von der verwendeten Bitrate und der *Sampling*-Frequenz ab.

SCFSI - steht für *scalefactor selection information* und gilt nur für Schicht II. Bei dieser Schicht werden die 36 pro Frequenzband kodierten Werte in drei Gruppen zu je 12 Werten aufgeteilt. Zugehörig zu diesen Gruppen werden 1 bis 3 Skalierungsfaktoren übertragen. Die *SCFSI* gibt Auskunft, wieviel Faktoren konkret übertragen werden und wie diese den einzelnen Gruppen zugeordnet sind. So können z.B. zwei Faktoren übertragen werden, wobei der erste Faktor für die ersten beiden Gruppen und der zweite Faktor für die dritte Gruppe gilt. Es sind hier alle denkbaren Kombinationen möglich.

Scalefactors - enthält die Information über die für die einzelnen Frequenzbänder verwendeten Skalierungsfaktoren. Bei Schicht I wird pro Band ein Faktor übertragen. Für die Schicht II hängt die Anzahl der Faktoren pro Band von der oben erläuterten *SCFSI* ab.

Samples - sind die kodierten Werte für einzelnen Frequenzbänder, deren Länge durch die *Bitallocation* bekannt ist. Bei Schicht I werden 12 Gruppen der 32 Frequenzbänder kodiert, womit sich 384 Werte ergeben. Auch die Schicht II kodiert 12 Gruppen, wobei allerdings pro Frequenzband 3 aufeinander folgende Werte kodiert werden, so daß sich eine Anzahl von 1152 Werten ergibt.

Ancillary Data - Dieses Feld ist für den eigentlichen Dekoder nicht von Bedeutung. Es kann zur Übertragung anwendungsspezifischer Informationen dienen.

Sind die einzelnen Informationen dekodiert, werden die Werte requantisiert und mit den Skalierungfaktoren skaliert. Die rekonstruierten *subband samples* werden nun mit Hilfe des Synthese-Filters in *PCM samples* rücktransformiert. Dieses Filter ist für alle Schichten identisch. Es wird im Abschnitt 4.2 noch Gegenstand näherer Betrachtung sein.

Bis zu dieser Stelle ist noch nicht auf die verschiedenen möglichen Modi eingegangen worden. Die Modi *stereo* und *dual channel* werden identisch kodiert. Sie dienen der Unterscheidung von echten Stereo-Signalen und Zweikanal-Ton (z.B. Englisch/Deutsch). Der Aufbau eines Rahmens ist wie oben beschrieben, wobei sich die einzelnen Werte für beide Kanäle abwechseln.

Der *joint stereo mode* ist eine Variante zur Kodierung von Stereo-Signalen. Bei diesem Modus werden ab einem bestimmten Frequenzband die *subband samples* für beide Kanäle summiert und als gemeinsamer Wert übertragen.

2.2 Das Zielsystem

Implementiert wurde der Dekoder auf einem *PCI-Board*, das aus einem kompletten Rechnersystem mit dem DSP (Digitaler Signal-Prozessor) TMS320C4x von Texas Instruments besteht. Der Prozessor wird mit 40 MHz getaktet. Die Speicherausstattung bestand aus dem internen 2 *KWord* großen SRAM des Prozessors, sowie 128 *KWord* externen SRAM. Die Versorgung mit Daten erfolgt über den *Host-PC*.

2.3 Stand der Technik

2.3.1 Aktuelle Entwicklungstendenzen

Der seit einiger Zeit stark anwachsende Bereich von multimedialen Anwendungen stellt besondere Anforderungen an heutige Rechnersysteme. Es gibt derzeit zwei Tendenzen, diesen Anforderungen gerecht zu werden. Das ist zum einen die Erweiterung der Prozessoren um spezielle Befehle zur Signalverarbeitung, wie z.B. die *MMX*-Erweiterung von Intel. Auf der anderen Seite kommt spezielle *Hardware*, wie z.B. Video-Beschleuniger, zum Einsatz.

Die Anwendung reiner *Software*-Lösungen wird trotz der Unterstützung durch angepasste Befehlssätze problematisch bleiben. Zum einen besitzen Multimedia-Anwendungen in der Regel einen hohen Bedarf an Rechenleistung, so daß Rechnersysteme allein durch eine einzige derartige Anwendung voll ausgelastet sein können. Zum anderen ist die Unterstützung von Echtzeit-Anwendungen bei den derzeit etablierten Betriebssystemen nicht gegeben. Der Vorteil von *Software*-Lösungen liegt in ihrer Flexibilität und den relativ geringen Entwicklungskosten.

Spezielle *Hardware*-Lösungen haben den Vorteil, daß sie autonom bestimmte Aufgaben abarbeiten und damit den Rechner entlasten. Diese Lösungen sind z.Z. relativ teuer, wenn die Preisentwicklung auch nach unten geht. Da meist speziell für den Anwendungszweck zugeschnittene Schaltkreise verwendet werden, sind *Hardware*-Lösungen relativ unflexibel.

Ein sich andeutender dritter Weg könnte die Vorteile der beiden ersten Varianten vereinen. Der Einsatz von DSP's in Subsystemen könnte den Rechner stark von Echtzeit-Aufgaben entlasten, während gleichzeitig durch ihre Programmierbarkeit eine flexible Anpassung an neue Anwendungen möglich ist. Die hier beschriebene Arbeit ist unter diesem Punkt einzuordnen.

2.3.2 Vergleichbare Arbeiten

In [2] sind einige aktuelle DSP-Entwicklungen vorgestellt. Diese DSP's werden zum Teil mit eigenen Echtzeit-Betriebssystemen und verschiedenen C-Bibliotheken, u.a. für MPEG-1-Dekodierung, angeboten. Beispiele sind der TriMedia von Philips oder der Mpact von Chromatic Research. Diese DSP's gehören einer gegenüber dem TMS320C4x wesentlich leistungsfähigeren neuen Prozessorgeneration an.

Ein Projekt mit vielen Parallelen zu der hier beschriebenen Arbeit wird in [1] vorgestellt. Die Autoren berichten über die Implementation eines echtzeitfähigen MPEG-1-Koders/Dekoders auf Basis des TMS320C30 von Texas Instruments. Der TMS320C4x ist eine erweiterte Version des dort verwendeten DSP's und enthält dessen Befehlssatz als Untermenge. Auch in dieser Arbeit stand das Ziel, echtzeitfähig zu sein. Der beschriebene Dekoder arbeitet allerdings nur mit Schicht I und ist unvollständig implementiert. Desweiteren wurde die Berechnung des Synthese-Filters algorithmisch nicht optimal gelöst. Leider standen die Quelltexte zu diesem Projekt nicht zur Verfügung.

Kapitel 3

Entwurf

Durch die sehr genaue Definition des Dekoders durch den Standard, ist die allgemeine Struktur der *Software*-Komponenten relativ klar vorgegeben. Zu treffende Entscheidungen sind deshalb klar auf Aspekte der konkreten Implementierung ausgerichtet, welche Gegenstand von Kapitel 4 sind. An dieser Stelle werden deshalb nur einige allgemeine Punkte erläutert.

3.1 Entwicklungsetappen

Der Dekoder wurde nicht sofort in einer optimalen Variante programmiert. Begonnen wurde die Arbeit mit einem Dekoder für die Schicht I, welcher lediglich den *single channel mode* unterstützte. Diese erste Variante wurde in C programmiert und bot Gelegenheit, sich mit der Problematik vertraut zu machen. Hier erfolgten auch entscheidende Optimierungen auf algorithmischer Ebene. Ein Vorteil dieser C-Implementation war die Unabhängigkeit vom Zielsystem, wodurch anfangs eine Programmierung am *PC* möglich war. Es folgte eine Portierung auf das *DSP-Board*, welche kaum Aufwand erforderte.

In der nächsten Etappe wurden schrittweise die einzelnen Funktionen des Dekoders in Assembler neu programmiert und eingebunden. Letzlich entstand eine vollständige Assembler-Variante des Dekoders. Dieser wurde noch um die fehlende Unterstützung für die anderen Modi (*dual channel, stereo, joint stereo*) erweitert. Die bisher gewonnenen Erfahrungen erlaubten es in einem weiteren Schritt den Dekoder für die Schicht II sofort in Assembler zu programmieren.

3.2 C versus Assembler

Wie alle Hochsprachen besitzt C einige Eigenschaften, die für die Programmierung zeitkritischer Probleme hinderlich sind. Die Effizienz der erzeugten Programme hängt stark von der Leistungsfähigkeit des *Compiler* ab. Der für den TMS320C4x verfügbare *Compiler* besitzt einige Schwachpunkte. So werden beispielsweise keine *delayed branches* verwendet, welche das Leeren der *instruction pipeline* bei Programmverzweigungen vermeiden.

Weitere Nachteile sind prinzipieller Natur. So können bedingt durch das höhere Abstraktionsniveau nicht immer alle Vorteile einer bestimmten Maschinenarchitektur ausgenutzt werden. Der TMS320C4x bietet z.B. spezielle Adressierungsmodi (*circular addressing*, *bitreversed addressing*) für die C keine sprachlichen Konstrukte bietet.

Bedingt durch die bestehenden Aufrufkonventionen besitzen Funktionsrufe in C einen gewissen *Overhead*, da bestimmte Prozessor-Register nicht verändert werden dürfen und im Bedarfsfall gerettet werden müssen. Bei der Programmierung in Assembler wurde eine andere Strategie verfolgt, welche in Abschnitt 4.1 noch erläutert wird.

Die Entscheidung, zentrale Komponenten in Assembler zu implementieren, war schon aus Effizienzgründen gegeben. Das schließlich der ganze Dekoder in Assembler programmiert wurde, war eine Folge des immer weiter erreichbaren Geschwindigkeitsgewinns. Selbst das scheinbar so unkritische Entpacken des Rahmens konnte in Assembler deutlich effizienter programmiert werden.

3.3 Schnittstelle zum Betriebssystem

Die Schnittstelle des Dekoders sollte möglichst flexibel gestaltet sein, um eine spätere Einbindung in ein DSP-Betriebssystem so flexibel wie möglich zu machen.

Der Dekoder stellt einen Satz von C-Funktionen zur Verfügung. Diese arbeiten rahmenorientiert. Die iterative Dekodierung aufeinanderfolgender Rahmen muß vom Betriebssystem organisiert werden. Die Dekodierung des *Header* wird durch eine separate Funktion bereitgestellt. Dadurch können die administrativen Daten noch vor der Dekodierung des eigentlichen Audio-Rahmens ausgewertet werden.

Fragen des Datentransports wurden offen gehalten. Der Dekodier-Funktion für den Audio-Rahmen werden die Adressen zweier Speicherbereiche für die *PCM-Samples* beider Kanäle übergeben. Die Größe und Anzahl dieser Bereiche kann aus den vorher dekodierten *Header*-Informationen ermittelt werden.

Im *single channel mode* wird nur der Bereich für den linken Kanal verwendet. Nach der Dekodierung müssen die Werte durch die konkrete Applikation weiterbehandelt werden.

Für die Bereitstellung des Bitstroms wurde ein „*call back*“-Mechanismus verwendet. Das Betriebssystem muß eine Funktion bereitstellen, welche der Dekoder aufruft, um einen neuen Puffer mit Bitstrom-Daten zu bekommen. Somit wird auch die konkrete Bereitstellung der Eingabedaten durch das Betriebssystem übernommen. Durch diese Funktion kann auch die Terminierung des Dekodierprozeßes gesteuert werden.

Eine denkbare Variante der Einbindung wäre, daß parallel zum Dekodierprozeß zwei DMA-Operationen laufen, welche die Ein- und Ausgabedaten transportieren. Wird vom Dekoder ein neuer Bitstrom-Puffer angefordert, so kann das Betriebsystem gleichzeitig den DMA-Transfer für den nächsten Puffer initiieren. Analog kann mit den Ausgangsdaten verfahren werden. Der Einfluß von konkurrierenden DMA-Prozeßen wird in Abschnitt 5.3.4 erörtert.

Kapitel 4

Ausgewählte Details der Implementation

4.1 Aufrufkonventionen für Funktionen

Für Funktionsrufe werden ausschließlich *delayed branches* (siehe [7], S. 6-9ff) verwendet, welche keine *pipeline*-Konflikte verursachen. Weiterhin wird innerhalb des Dekoders eine Aufrufkonvention angewendet, welche sich von der bei C üblichen Konvention unterscheidet.

Eine Funktion darf prinzipiell alle Register verwenden, ohne diese retten zu müssen. Es wird jedoch nach außen dokumentiert, welche Register im konkreten Fall durch die Funktion modifiziert werden. Wenn der aufrufende Kontext diese Register auch verwendet, so müssen sie vor dem Aufruf der Funktion gerettet werden. Durch geschickte Verwendung der Register ist in der Praxis das Retten in den seltensten Fällen notwendig, wie die äußerst geringe Anzahl an `PUSH/POP`-Operationen beweist. Durch diese Maßnahmen kosten Funktionsaufrufe relativ wenig Zeit.

Um den Dekoder in einem C-Programm nutzen zu können, wurde eine entsprechende Schnittstelle geschaffen.

4.2 Das Synthese-Filter

Das Synthese-Filter ist zentraler Bestandteil des Dekoders für alle drei Schichten. Seine Implementierung war deshalb einer der entscheidenden Punkte, weshalb es auch die erste Funktionseinheit war, welche in Assembler umgesetzt wurde. Das Filter ist im Standard durch den in Abbildung 4.1 dargestellten Rechenalgorithmus beschrieben. Es ist leicht zu erkennen, daß die im Schritt 3 durchgeführte Transformation sehr rechenaufwendig ist. Unter der

Abbildung 4.1: Rechenalgorithmus des Synthese-Filters

1. Eingabe von 32 neuen Werten

   S_i mit $i = 0 \ldots 31$

2. Verschiebung, für $i = 1023 \ldots 64$

   $V_i = V_{i-64}$

3. Transformation

   $V_m = \sum_{i=0}^{31} S_i \cdot \cos[\frac{(2i+1)(m+16)\pi}{64}]$

   mit $m = 0 \ldots 64$

4. Berechnung eines Zwischenvektors U

   $U_{j+64i} = V_{j+128i}$
   $U_{j+64i+32} = V_{j+128i+96}$

   mit $i = 0 \ldots 7;\ j = 0 \ldots 31$

5. Wichtung mit Vektor D

   $W_m = U_m \cdot D_m$ mit $m = 0 \ldots 511$

6. Berechnung von 32 neuen *samples*

   $X_n = \sum_{j=0}^{15} W_{n+32j}$ mit $n = 0 \ldots 31$

Annahme, das die Kosinus-Terme bereits in *look-up*-Tabellen vorliegen, sind 2048 Multiplikationen und 1984 Additionen zur Berechnung notwendig. In [3] ist eine Möglichkeit der Optimierung vorgestellt, welche im Wesentlichen übernommen wurde und folgend kurz erläutert wird.

Unter Ausnutzung von periodischen Eigenschaften kann die Transformation auch auf folgende Weise berechnet werden:

$$V_m^* = \sum_{i=0}^{31} S_i \cdot \cos \frac{(2i+1) \cdot m \cdot \pi}{64} \quad , m = 0 \ldots 31 \tag{4.1}$$

$$\begin{aligned} V_i &= V_{i+16}^* && ,\text{mit } i = 0 \ldots 15 \\ V_{i+17} &= -V_{31-i}^* && ,\text{mit } i = 0 \ldots 15 \\ V_{i+32} &= -V_{16-i}^* && ,\text{mit } i = 0 \ldots 15 \\ V_{i+48} &= -V_i^* && ,\text{mit } i = 0 \ldots 15 \\ V_{16} &= 0 \end{aligned}$$

Obwohl sich der Rechenaufwand in dieser Form schon auf fast die Hälfte verringert hat, ist eine weitere Vereinfachung möglich. Bei Formel 4.1 handelt es sich um eine Diskrete Kosinus-Transformation (DCT), deren allgemeine Form durch Formel 4.2 beschrieben ist:

$$X_i = \mu_i \cdot \sum_{k=0}^{K-1} S_k \cdot \cos \frac{(2k+1) \cdot i \cdot \pi}{2 \cdot I} \quad , i = 0 \ldots I-1 \tag{4.2}$$

Für Formel 4.1 gilt $I = K = 32$, und $\mu_i = 1$ für alle i. Für die DCT sind einige schnelle Algorithmen bekannt. In [3] wird die Lee-Dekomposition verwendet. Dabei wird ähnlich wie bei der Schnellen Fourier-Transformation eine $N \times N$-DCT in zwei $\frac{N}{2} \times \frac{N}{2}$-$DCT$*'s* zerlegt. Abbildung 4.2 zeigt eine solche Zerlegung am Beispiel einer 8×8-DCT. Die rekursive Anwendung dieser Zerlegung führt zu einer beträchtlichen Reduzierung des Rechenaufwandes. In Tabelle 4.1 ist die Anzahl der notwendigen Rechenschritte für die verschiedenen Varianten der Berechnung vergleichend dargestellt. Dabei wird angenommen, daß die Kosinus-Terme in *look-up*-Tabellen vorliegen, so daß ihre Berechnung entfällt.

Tabelle 4.1: Aufwand für verschiedene Berechnungsvarianten desTransformationsschrittes

Algorithmus	Mult	Add/Sub
ohne Optimierung	2048	1984
Rückführung auf DCT	1024	1040
DCT mit Lee-Dekomposition	80	321

Die schnelle Variante mittels Lee-Dekomposition wurde als rekursive Funktion sowohl in C, als auch in Assembler implementiert. Erstaunlich ist der große Unterschied in der Rechenzeit. Die Assembler-Version ist fast 5 mal schneller als ihr Pendant in C. Durch die Rekursivität der Funktion ist vermutlich der in C wesentlich höhere *Overhead* bei den wiederholten Funktionsaufrufen der entscheidende Schwachpunkt.

Endgültig zur Anwendung im Dekoder kam eine speziell auf die 32×32-*DCT* zugeschnittene iterative Implementation in Assembler. In dieser Variante wurden die fünf notwendigen Dekompositionsschritte sequentiell programmiert. Aufgrund des *input butterfly* ist eine Zerlegung innerhalb eines Puffers nicht möglich. Deshalb werden zwei Puffer verwendet, die in jeder Stufe abwechselnd als Quell- und Ziel-Puffer dienen. Durch Anwendung von indirekter Adressierung konnte der abschließende Schritt der Zusammenführung von geraden und ungeraden Punkten vermieden werden. Dies ist möglich, indem bei der Zerlegung durch den *input butterfly* die geraden Punkte auf gerade Adressen, sowie die ungeraden Werte entsprechend auf ungeraden Adressen im Ziel-Puffer abgelegt werden. Damit stehen die Werte von vornherein an der richtigen Stelle. Abbildung 4.3 illustriert dieses Vorgehen am Beispiel der ersten zwei Zerlegungsstufen. Die Skalierung der ungeraden Punkte mit den S_i-Werten wird bei der Zerlegung mit durchgeführt. Diese Werte werden *look-up*-Tabellen entnommen.

Die Rechenzeiten der verschiedenen Implementationen der *DCT* sind in Tabelle 4.2 dargestellt.

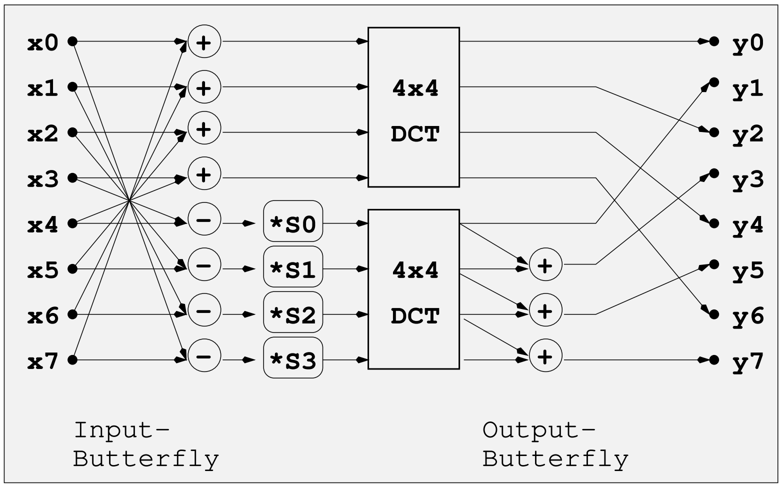
Tabelle 4.2: Rechenzeiten für die *DCT*

Implementation	Rechenzeit	
in C, rekursiv	$224\mu s$	100%
in ASM, rekursiv	$47\mu s$	21%
in ASM, iterativ	$30\mu s$	14%

Neben der Transformation konnten weitere Schritte im Synthese-Filter optimiert werden. So wurden die Schritte 4 und 5 zusammengefaßt. Um die in Schritt 2 notwendige Werteverschiebung zu vermeiden, wurde der Vektor V als Ring-Puffer organisiert. Für solche Zwecke stellt der DSP den *circular addressing mode* zur Verfügung. Bei Adreßrechnungen wird hierbei automatisch ein *wrap around* durchgeführt, wenn eine Puffergrenze erreicht wird. Die Größe des Puffers wird in das `BK`-Register des Prozessors eingetragen (siehe auch [7], S. 5-25ff).

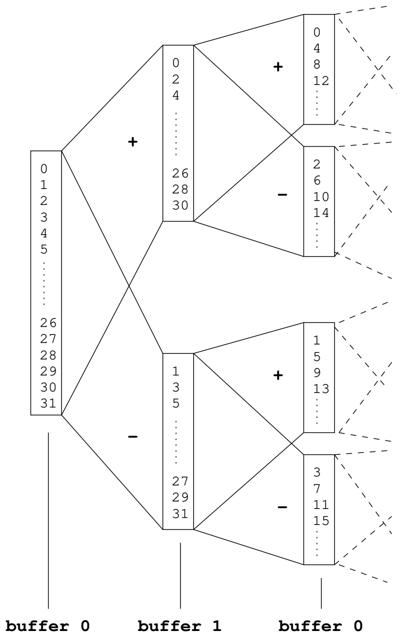
Aufgrund der besonderen Bedeutung des Synthese-Filters wurde bei der Programmierung besonders auf folgende Punkte geachtet:

Abbildung 4.2: Lee-Dekomposition am Beispiel einer 8×8-*DCT*



$$S_i = \frac{1}{2 \cdot \cos[\frac{\pi}{2N} \cdot (2i+1)]} \quad , i = 0 \dots N-1$$

Abbildung 4.3: Speicherorganisation bei der Dekomposition (dargestellt sind die ersten zwei Stufen)



- Vermeidung von *Pipeline*-Konflikten durch entsprechende Anordnung der Befehle,
- Vermeidung von konkurrierenden Speicherzugriffen,
- Verwendung paralleler Befehle.

Der zweite Punkt ist vor allem abhängig von der Organisation der einzelnen Segmente im Speicher, auf die im nächsten Abschnitt noch eingegangen wird.

4.3 Speicherorganisation

Die Leistungsfähigkeit des TMS320C4x hängt stark von der Organisation des Speichers ab. Der Prozessor besitzt intern eine sehr aufgefächerte Bus-Struktur, die es ermöglicht, gleichzeitig zwei Datenworte, einen Befehl und ein DMA-Datenwort zu transferieren. Parallele Befehle sind auf diese Möglichkeit angewiesen. Die geschickte Verteilung von Daten- und Kode-Bereichen auf den Speicher ist essentiell. Während der interne Speicher zwei Zugriffe pro Zyklus zuläßt, ist extern nur ein Zugriff möglich. Eine eingehendere Erklärung zu diesem Thema ist in [7], Abschnitte 10.2.3 und 10.3, zu finden.

Die Daten- und Kode-Bereiche des Dekoders wurden so gruppiert, daß eine effiziente Speicherverteilung vorgenommen werden kann. In Abschnitt 5.3.3 wird diese Frage im Zusammenhang mit der Rechengeschwindigkeit noch einmal aufgegriffen.

4.4 *Degrouping*

Wie bereits in Abschnitt 2.1.2 erwähnt, werden bei Schicht II in jeder Gruppe 3 aufeinanderfolgende Werte pro Frequenzband kodiert. Für einige Quantisierungsstufen werden diese Werte zu einem Kodewort zusammengefaßt (*grouping*). Die Zerlegung im Dekoder wird mit folgendem Algorithmus realisiert:

```
for  (i = 0; i < 3; i++) {
     s[i] = c % nlevel;
     c = c DIV nlevel;
}
```

Dabei sind `c` das gepackte Kodewort, `s[i]` die drei zerlegten Werte, und `nlevel` die Anzahl der Quantisierungsstufen.

Der Prozessor bietet keine direkten Befehle, um diese Operationen auszuführen. Da die `nlevel`-Werte keine Zweierpotenzen sind, ist die Anwendung logischer Operationen nicht möglich. Die Nutzung von *look-up*-Tabellen wurde verworfen, da ihre Größe indiskutabel erschien.

In [7], Abschnitt 12.3.5.1, ist ein iteratives Verfahren auf Basis wiederholter Subtraktion für den TMS320C4x beschrieben, welches sowohl die Ganzzahl-Division, als auch die Modulo-Operation berechnet. Das beschriebene Verfahren wurde für das *degrouping* optimiert und als Makro implementiert. Das Makro benötigt zwischen 21 und 25 Zyklen. Da *degrouping* nur bei wenigen Werten durchgeführt werden muß, ist diese Dauer akzeptabel.

Kapitel 5

Leistungsbewertung

Die entscheidende Forderung an den Dekoder ist seine Echzeitfähigkeit. In diesem Kapitel wird es deshalb ausschließlich um die Frage der Gewährleistung dieser Forderung gehen. Dazu muß die Echtzeitanforderung für den Dekoder genau definiert werden. Weiterhin wird deren Einhaltung analysiert. Dabei wird auf verschiedene Faktoren eingegangen, welche den Dekoder in seiner Rechenzeit beeinflussen.

5.1 Meßmittel

Die Untersuchung von Zeitanforderungen erfordert ein möglichst genaues Meßmittel. Deshalb wurde zu diesem Zweck einer der zwei beim TMS320C4x verfügbaren Zähler verwendet. Als Taktquelle dient dem Zähler der um den Faktor 2 vorgeteilte Zyklustakt `H1`. Dadurch ist eine auf die Dauer von ca. zwei Befehlen genaue Messung möglich! Zur Messung wird das Zählregister rückgesetzt und der Zähler gestartet. Am Ende der Messung wird der Zähler gestoppt und das Zählregister ausgelesen. Aus dem Zählerwert und der Taktrate von `H1` kann dann die Zeit ausgerechnet werden. Einzelheiten zu den Zählereinheiten des TMS320C4x findet man in [7], Abschnitt 9.13.

5.2 Definition der Echtzeitanforderung

Die obere Grenze für die Dauer der Dekodierung eines Rahmens entspricht der Zeitdauer der in diesem Rahmen kodierten *PCM-samples*. Diese Grenze ist von der eingesetzten Schicht und der *Sampling*-Frequenz abhängig. In Tabelle 5.1 sind die bestehenden Zeitgrenzen für die verschiedenen Konstellationen aufgelistet.

Tabelle 5.1: Bestehende Zeitgrenzen

Schicht	*Samples* pro Rahmen	*Sampling*-Frequenz in kHz		
		32.0	44.1	48.0
I	384	12.0 ms	8.707 ms	8.0 ms
II	1152	36.0 ms	26.123 ms	24.0 ms

In der realen Anwendung muß die Dekodierzeit unter diesen theoretischen Grenzen liegen, da der Prozessor ja wenigstens noch den Datentransfer organisieren muß, und um einen Sicherheitsbereich zu haben.

5.3 Echtzeitanforderung erfüllt?

Alle im weiteren Verlauf angegebenen Werte beziehen sich auf die Zeit, welche der Dekoder braucht, um die Synchronisation durchzuführen und anschließend *Header* und Audio-Daten zu dekodieren. Die Bereitstellung der Eingangsdaten und die Verarbeitung der Ausgangsdaten blieb unberücksichtigt.

Eine Obergrenze für die Rechenzeit anzugeben ist praktisch unmöglich, da diese von verschiedenen Faktoren abhängt. Diese Faktoren werden in den nächsten Abschnitten näher untersucht. Dabei wird oft ein durchschnittlicher Auslastungsgrad für den Prozessor angegeben. Diese Last wird aus dem Quotienten der durchschnittlichen Rechenzeit und der Obergrenze, nach welcher die Dekodierung des nächsten Rahmens beginnen muß, gebildet.

Allgemein kann festgestellt werden, daß die bestehenden Zeitlimits bei allen Messungen eingehalten wurden.

5.3.1 Einfluß der Natur des Audio-Signals

Da der Kodiermechanismus in starkem Maße von der Natur des Audio-Signals abhängt, schwankt die Rechenzeit sehr stark. Um eine statistische Aussage machen zu können, wurde die Dekodierzeit für verschiedene Audio-Quellen untersucht. Verwendet wurden verschiedene Musikstücke, welche mit einer *Sampling*-Frequenz von 44.1 KHz in Stereo aufgenommen wurden. Zur Kodierung wurde die für die jeweilige Schicht höchste Bitrate verwendet und der *stereo mode* verwendet. Diese Kodieroptionen stellen die höchste Anforderung an den Dekoder. Es wurde eine optimale Speicherbelegung gewählt, auf deren Einfluß im nächsten Abschnitt gesondert eingegangen wird. In Tabelle 5.2 sind die über alle Dateien gemessenen Minimum-, Maximum- und

Durchschnittswerte zusammengefaßt. Weiterhin ist die durchschnittliche Prozessorlast angegeben.

Tabelle 5.2: Statistik der Rechenzeiten pro Rahmen

Schicht	Rechenzeit in ms			durchschnittl.
	minimal	durchschnittlich	maximal	Prozessorauslastung
I	4.389	4.464	4.599	52 %
II	12.655	13.101	13.878	51 %

5.3.2 Einfluß der Bitrate

Die Länge eines Rahmens hängt von der Bitrate ab. Um den Einfluß dieses Faktors zu untersuchen, wurden MPEG-1-Dateien mit unterschiedlicher Bitrate von einer identischen Audio-Quelle kodiert. Es wurde die Schicht I im *single channel mode* verwendet. Die durchschnittliche Prozessorlast lag bei 23 Prozent bei der niedrigsten Bitrate von 32 kBit/s und 28 Prozent bei der höchsten Bitrate von 448 kBit/s. Ihr Einfluß ist demnach relativ gering.

5.3.3 Einfluß der Speicherbelegung

Bereits in Abschnitt 4.3 wurde die Bedeutung der Speicherbelegung näher erläutert. An dieser Stelle sollen die dort gemachten Aussagen mit Zahlen belegt werden. Hierzu wurde ein Bitstrom, Schicht II im *stereo mode*, mehrfach dekodiert. Dabei wurde die Zuordnung der Speichersegmente zu internem und externem Speicher variiert. Die Durchschnittswerte zwischen der günstigsten und der schlechtesten Variante lagen dabei um 8.6 ms auseinander. Das enspricht einem Zuwachs der durchschnittlichen Prozessorlast von 52 Prozent auf 85 Prozent!

5.3.4 Einfluß konkurrierender DMA-Prozeße

In einer realen Echtzeitanforderung müssen Ein- und Ausgangsdaten des Dekoders transferiert werden. Sinnvoll ist dazu die Anwendung von DMA-Transfers. In einem Versuch wurden parallel zum Dekoder zwei DMA-Operationen durchgeführt. Der Umfang der dabei transferierten Daten entsprach dem Umfang der Ein- und Ausgabewerte des Dekoders. Der Versuch wurde sowohl mit der günstigsten, als auch mit der ungünstigsten Speicherbelegung durchgeführt. Eine signifikante Änderung der Rechenzeit war nicht feststellbar.

Kapitel 6

Ergebnisse und Zusammenfassung

Im Rahmen dieser Arbeit entstand ein echtzeitfähiger Dekoder für das MPEG-1-Audio-Format nach dem Standard ISO/IEC 11172-3. Der Dekoder erfüllt alle von diesem Standard geforderten Kriterien für die Schichten I und II. Diese Arbeit zeigt auf, daß eine sehr stark auf die zugrundeliegende *Hardware* abgestimmte Programmierung in Assembler der Schlüssel für die Erfüllung zetikritischer Rechenaufgaben ist. Es zeigte sich, daß die konkrete Rechenzeit des Dekoders von vielfältige Faktoren abhängt, so daß allgemeine Aussagen nur statistischer Natur sein können. Dies ist ein sehr kritischer Punkt für den Einsatz in einem echtzeitfähigen *Multitasking*-Betriebssystem.

Die Einarbeitung in die Problematik und die Programmierung in Assembler erwiesen sich als sehr zeitaufwendig. War anfangs noch die Realisierung eines Koders geplant, mußte dies jedoch verworfen werden, da diese Aufgabe den zur Verfügung stehenden Zeitrahmen gesprengt hätte. Der Aufbau eines Koders stellt die nächste logische Aufgabe dar. Als besonders anspruchsvoll dürfte sich dabei die Implementierung des Psycho-Akustischen Modells erweisen (siehe Abschnitt 2.1.1).

Literaturverzeichnis

[1] Anand K. Anandakumar und Charles D. Murphy: *A Real-Time MPEG-1 Audio Codec Using A TMS320C30.* `http://www.seas.upenn.edu/~anandaku/`, 1996.

[2] Bertuch, Manfred: *Vielarmige Gestalten.* c't, Seiten 256–258, März 1997.

[3] *Using MMX Instructions to Implement a Synthesis Sub-Band Filter for MPEG Audio Decoding.* `http://developer.intel.com/drg/mmx/AppNotes/AP533.HTM`.

[4] ISO/IEC: *IS 11172-3, Information technologie - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part3: Audio*, 1. Auflage, 1993.

[5] Pan, Davis: *A Tutorial on MPEG/Audio Compression.* `http://www.bok.net/~pan/`, 1995.

[6] Pan, Davis Yen: *Digital Audio Compression.* Digital Technical Journal, 5(2), 1993. `ftp://ftp.digital.com/pub/Digital/info/DTJ/v5n2/Digital_Audio_Compression_01oct1993DTJA03P8.ps`.

[7] Texas Instruments Inc.: *TMS320C4x User's Guide*, 1993.