Diplomarbeit

zu dem Thema

Echtzeitfähiger MPEG-Audio-*Encoder* 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:	11. Juni 1998

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Kapitel 1

Einleitung

Im letzten Jahrzehnt hat sich der *Computer* vom teuren Spezialwerkzeug zum Konsumgut entwickelt. Diese Enwicklung wurde unterstützt durch die Eroberung von Anwendungsgebieten, welche lange Zeit die Domäne der Kommunikationstechnik und der Unterhaltungselektronik waren. Heute kann man mit dem *Computer* - neben seinen klassischen Aufgaben - telefonieren, Briefe versenden, spielen, Videos ansehen und Musik hören. Dabei stellt gerade die zeitgerechte Verarbeitung von Video- und Audiodaten große Anforderungen an die Rechentechnik und das Betriebssystem. Die Architektur von Mehrzweck-Prozessoren ist für diese Art von Aufgaben nicht optimal. In den letzten Jahren hat man zwar versucht, dieses Manko durch Erweiterungen in der Achitektur, wie z.B. *MMX* des Pentium-Prozessors von Intel, zu mildern. Trotzdem muß sich dieses Konzept den anderen Anforderungen der Mehrzweck-Prozessoren unterordnen. Ein besserer Weg ist es, das System durch Spezialprozessoren zu erweitern. Digitale Signalprozessoren (DSP) sind solche Spezialprozessoren, welche auf Aufgaben der digitalen Signalverarbeitung optimiert sind. Erste Produkte, die diesen Ansatz verfolgen, kommen derzeit auf den Markt [1].

Eine wichtige Aufgabe bei der Verarbeitung von Video- und Audiosignalen ist die Kompession und Dekompression. Diese Techniken sind wichtig, um die enormen Datenmengen dieser Signale effizient speichern und übertragen zu können. Das Ziel dieser Arbeit war deshalb die Erstellung eines echtzeitfähigen Programms für die Kompression von Audiosignalen nach dem internationalen Standard ISO/IEC 11172-3 (MPEG-1/Audio) auf dem

DSP TMS320C44 von Texas Instruments. Das vorliegende Dokument beschreibt, welche Probleme dabei auftraten und wie sie gelöst wurden. Der Einleitung folgt im Kapitel 2 eine Einführung in die zu implementierende Kodiertechnik. Kapitel 3 setzt sich mit den wichtigsten Algorithmen und deren Optimierung auseinander. Einige ausgewählte Aspekte der programmtechnischen Realisierung sind Inhalt des Kapitels 4. Die Frage, ob eine echtzeitfähige Lösung gefunden werden konnte, wird im Kapitel 5 beantwortet. Dort wird auch auf die Qualität der kodierten Signale eingegangen. Abgeschlossen wird die Arbeit durch das Kapitel 6, welches die Ergebnisse noch einmal zusammenfaßt.

An dieser Stelle möchte ich meinen Betreuern Herrn Dipl.-Inform. Robert Baumgartl und Herrn Prof. Dr. Hermann Härtig für die gewährte Unterstützung danken.

Kapitel 2

Einführung in Aufbau und Funktionsweise eines *Encoders* nach ISO/IEC 11172-3

Der internationale Standard ISO/IEC 11172 beschreibt das Format von Bitströmen und ihre Interpretation zur bitratenreduzierten Kodierung von digitalen Videosignalen und zugehörigen Audiosignalen. Die durch ihn beschriebenen Kodiertechniken sind auch unter der Bezeichnung MPEG-1 bekannt. Der Standard umfaßt vier Teile:

1. *Systems*
2. *Video*
3. *Audio*
4. *Compliance testing.*

Für die vorliegende Arbeit von Interesse ist der dritte Teil (ISO/IEC 11172-3), welcher die Kodierung von Audiosignalen beschreibt. Dieser erlaubt durch verschiedene Modi die Kodierung von Mono-, Stereo- und Zweikanalsignalen. Es werden drei verschiedene Möglichkeiten der Kodierung definiert, welche im Standard als Schichten (engl. *layers*) bezeichnet werden. Alle Schichten verwenden eine verlustbehaftete Kodierung, d.h. eine bitgenaue Rekonstruktion des Quellsignals aus dem kodierten Signal ist nicht möglich. Die eingesetzten Verfahren beruhen auf der Reduktion von Redundanz und Irrelevanz. Die Schichten unterscheiden sich in der für die Kodierung und Dekodierung notwendigen Rechenkomplexität, sowie in dem bei vergleichbarer Signalqualität erreichbaren Kompressionsfaktor. Schicht 1 ist hierbei das einfachste Verfahren, welches am stärksten verlustbehaftet ist. Schicht 2 ist etwas komplexer als Schicht 1 und erzielt eine bessere Qualität. Am wenigsten verlustbehaftet ist die Schicht 3 auf Kosten einer wesentlich höheren Komplexität. Da mit der zur Verfügung stehenden *Hardware* die Realisierung eines in Echtzeit arbeitenden *Encoders* für Schicht 3 ausgeschlossen erscheint, wird auf diese nicht näher eingegangen.

In den folgenden Abschnitten werden die Funktionseinheiten des *Encoders* vorgestellt und das zugrundeliegende Kodierprinzip für die Schichten 1 und 2 einführend erläutert. Soweit nicht anders ausgeführt, gelten die Beschreibungen für beide Schichten. Abbildung 2.1 illustriert zur Übersicht den Gesamtaufbau des *Encoders*. Die Pfeile zwischen den Funktionseinheiten verdeutlichen den Datenfluß.

Weiterführende Informationen zu diesem Thema sind in [3, 4, 7, 8] zu finden. Einen guten Einblick in die Entwicklung und Gütebewertung von Kompressionsalgorithmen für Audiosignale gibt [3]. Für allgemeine Fragen der digitalen Signalverarbeitung sei auf [5] verwiesen.

2.1 Der Eingangspuffer

Der *Encoder* verarbeitet digitale Abtastwerte (*PCM-Samples*), welche durch Pulskode-Modulation (engl. *pulse code modulation*, *PCM*) mit linearer Quantisierung aus beliebigen Audiosignalquellen gewonnen werden können (siehe [5], Abschnitte 1.9 und 3.3). Die Bitgenauigkeit sollte wenigstens 16 Bits betragen, um die Quantisierungverzerrungen (Signal-Rausch-Verhältnis) gering zu halten.

Die Arbeitsweise des *Encoders* ist rahmenorientiert. Es werden jeweils 384 Abtastwerte bei Schicht 1 sowie 1152 Abtastwerte bei Schicht 2 als zu kodierende Einheiten betrachtet. Der Eingangspuffer sammelt die eingehenden Abtastwerte und gruppiert sie für die nachfolgenden Funktionseinheiten. Desweiteren realisiert er eine zeitliche Verzögerung zwischen den Daten für die Analysefilterbank und den Daten für das psycho-akustische Modell. Die

Abbildung 2.1 Gesamtaufbau des *Encoders*

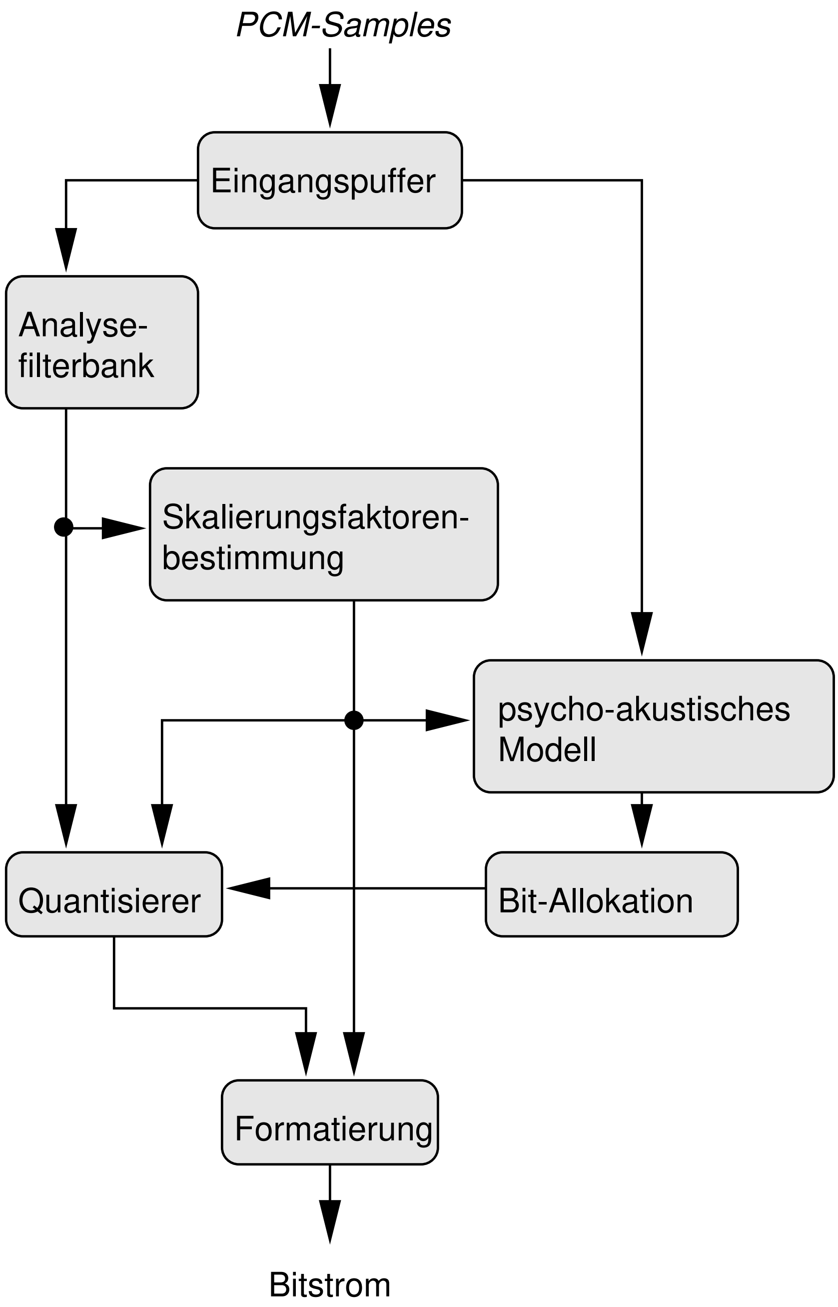
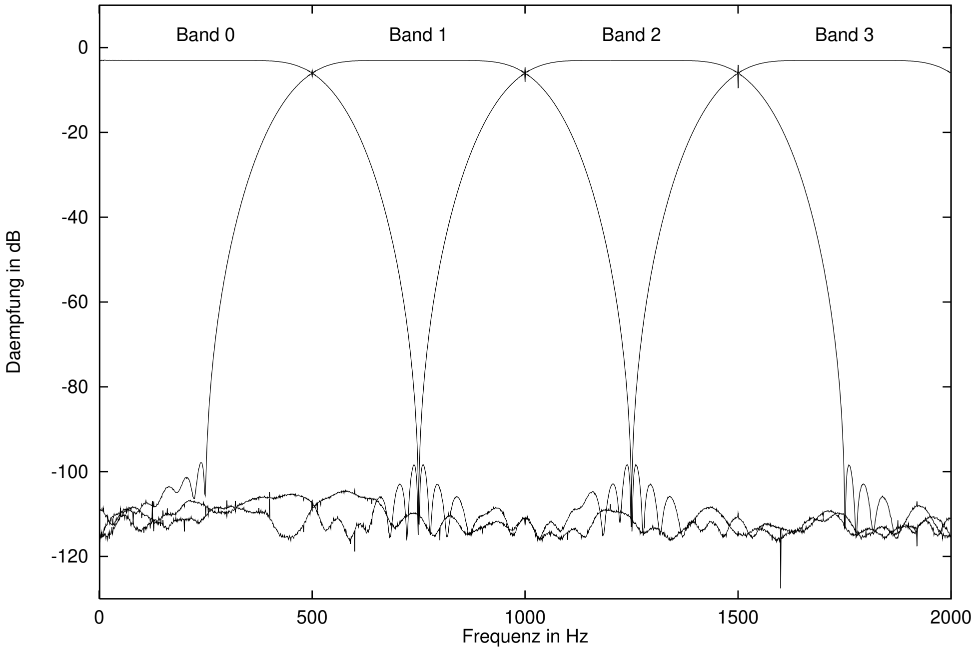


Abbildung 2.2 Übertragungskennlinien der ersten vier Bandpaß-Filter der Analysefilterbank bei einer Abtastfrequenz von 32 kHz



Notwendigkeit dieser Verzögerung wird im Abschnitt 2.2 erläutert.

2.2 Die Analysefilterbank

Der zu kodierende Strom von Abtastwerten repräsentiert ein bandbreitenbegrenztes Zeitsignal, dessen obere Grenzfrequenz die halbe Abtastfrequenz nicht überschreiten darf. Diese Forderung ergibt sich aus dem Abtasttheorem ([5], Abschn. 3.1). Die Analysefilterbank zerlegt das Zeitsignal in 32 äquidistante Frequenzbänder, welche insgesamt den Frequenzbereich vom Gleichanteil bis zur halben Abtastfrequenz abdecken. Realisiert wird diese Zerlegung mittels digitaler Bandpaß-Filter. Diese sind, ähnlich analogen Filtern, keine idealen Filter ([5], Abschn. 4.5). Aus diesem Grund kommt es zur Überschneidung der Frequenzbereiche benachbarter Filter. In Abbildung 2.2 sind die Übertragungskennlinien der ersten vier Filter bei einer Abtastfrequenz von 32 kHz dargestellt. Aus der Abbildung ist auch ersichtlich, daß die resultierende Übertragungskennlinie der gesamten Filterbank keine konstante Dämpfung aufweist. Diese Eigenschaft bedingt eine irreversible Änderung des Ausgangssignals. In Verbindung mit der Synthesefilterbank des *Decoders* wird diese Dämpfungsschwankung in [7] mit weniger als 0.07 dB angegeben und kann daher als vernachlässigbar angesehen werden.

Die Analysefilterbank führt neben der Zerlegung des Zeitsignals eine Reduzierung der Abtastrate (engl. *sub-sampling*) durch. Für 32 eingehende Abtastwerte wird pro Frequenzband ein *Sample* ausgegeben, welches auch als *Sub-band sample* bezeichnet wird. Somit erzeugt die Analysefilterbank pro 32 Abtastwerte einen Satz von 32 *Sub-band samples*.

Der Rechenalgorithmus der Analysefilterbank bedingt eine zeitliche Verzögerung, welche 256 Abtastwerten entspricht. Aus diesem Grund werden die Daten für das psycho-akustische Modell im Eingangspuffer (Abschn. 2.1) entsprechend verzögert, damit diese mit den Ausgabedaten der Filterbank korrespondieren.

Warum eine Zerlegung in Frequenzbänder sinnvoll ist, wird in Verbindung mit den psycho-akustischen Eigenschaften des menschlichen Gehörs deutlich, welche im Abschnitt 2.4 erläutert werden.

2.3 Bestimmung der Skalierungsfaktoren

Der im Abschnitt 2.6 näher beschriebene Quantisierer verwendet eine lineare Quantisierung. Wie in [5], Abschn. 3.2.2 beschrieben, besitzen lineare Quantisierer bezüglich des Signalmaximalwertes kein konstantes Signal-Rausch-Verhältnis (auch Rauschabstand, engl. *signal-to-noise ratio*, *SNR*). Dieses Verhältnis ist am größten, wenn der Signalmaximalwert dem Maximalwert des Quantisierungsbereichs entspricht, d.h. die Abtastwerte des Signals den vollen Wertebereich des Quantisierers ausschöpfen.

Unter Beachtung dieses Effekts ist es sinnvoll, die *Sub-band samples* der einzelnen Bänder pro Rahmen so zu skalieren, daß der zur Verfügung stehende Wertebereich möglichst voll ausgeschöpft wird. Der Standard definiert zu diesem Zweck eine Tabelle mit 63 Skalierungsfaktoren. Für jedes Frequenzband wird ein Faktor aus dieser Tabelle ermittelt. Hierzu wird der kleinste Wert aus der Tabelle bestimmt, welcher größer ist als der betragsmäßig größte Wert der *Sub-band samples* des jeweiligen Bandes oder diesem gleicht. Dividiert man die *Sub-band samples* durch diesen Faktor, so entspricht das einer Transformation in das Intervall $[-1, +1]$, welches im Idealfall vollständig ausgeschöpft wird.

Abweichend zur obigen Darstellung werden bei Schicht 2 pro Band drei Faktoren ermittelt. Die 36 ($\frac{1152}{32}$) *Sub-band samples* eines Bandes werden hierzu in drei Gruppen zu je 12 Werten geteilt. Für jede dieser Gruppen wird wie beschrieben ein Skalierungsfaktor bestimmt. Je nachdem, inwieweit diese Faktoren voneinander abweichen, werden nicht alle drei Faktoren verwendet. Wenn z.B. die Faktoren der ersten beiden Gruppen eine geringe Abweichung voneinander haben, so wird der größere Faktor für beide Gruppen verwendet. Die Größe der Abweichungen und die möglichen Zusammenlegungen sind im Standard genau definiert. Ziel dieser Technik ist die Reduzierung von zu kodierenden Informationen, da im besten Fall nur ein Skalierungsfaktor gegenüber maximal drei Werten kodiert werden muß.

Die Skalierungsfaktoren werden dem im Abschnitt 2.6 beschriebenen Quantisierer bereitgestellt. Die zu den Faktoren gehörenden Tabellenindizes werden bei der späteren Formatierung (Abschnitt 2.7) verwendet.

2.4 Das psycho-akustische Modell

Die wesentliche Anforderung an das Kodierverfahren besteht in der Transparenz der Datenreduktion für die subjektive Wahrnehmung des Menschen. Zur Gewährleistung dieser Transparenz bedient man sich bestimmter Wahrnehmungsschwächen des menschlichen Gehörs.

So besitzt das menschliche Gehör eine untere Wahrnehmungsschwelle, welche als Ruhehörschwelle (engl. *threshold in quiet*) bezeichnet wird. In Abbildung 2.3 ist der frequenzabhängige Verlauf dieser Hörschwelle dargestellt. Die in Dezibel (dB) angegebenen Verhältniswerte beziehen sich auf den in der Schallmeßtechnik festgelegten Bezugsschalldruck $\tilde{p}_0 = 2 \cdot 10^{-5} Pa$ ([2], Stichwort Hörschwelle). Signalanteile mit einem Schalldruck unter dieser Schwelle werden nicht wahrgenommen und sind aus diesem Grund für das menschliche Gehör irrelevant. Es ist anzumerken, daß die individuelle Ruhehörschwelle eines Menschen von Faktoren wie der Tageszeit, dem Alter und der täglichen Lärmbelastung abhängt. Die abgebildete und im *Encoder* verwendete Ruhehörschwelle stellt somit eine Verallgemeinerung dar.

Eine weitere Eigenschaft des menschlichen Gehörs ist die Maskierung (Überdeckung) von schwächeren Signalkomponenten in der spektralen Umgebung starker Signalkomponenten. Man kann diese Eigenschaft auch als Anhebung der unteren Wahrnehmungsschwelle in der spektralen Umgebung

Abbildung 2.3 Ruhehörschwelle des menschlichen Gehörs

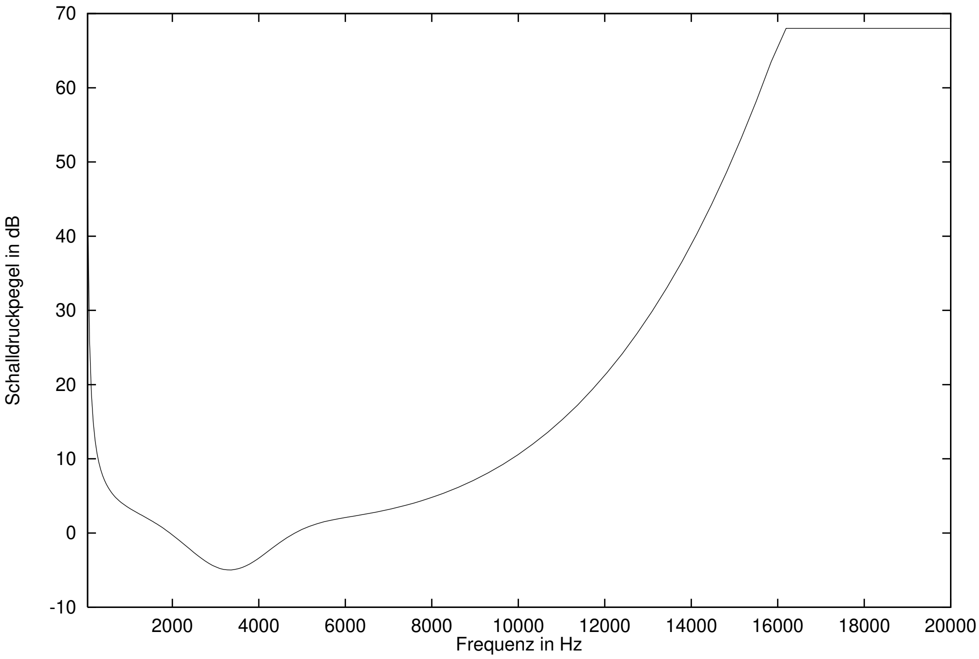
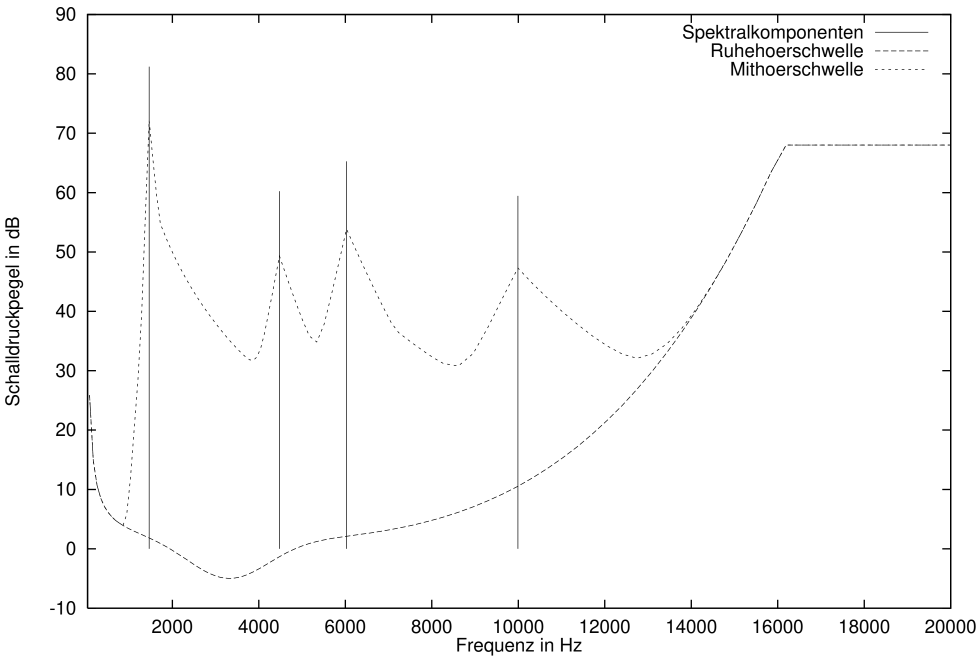


Abbildung 2.4 Beispiel für den Maskierungseffekt



des Signals verstehen. In Abbildung 2.4 ist die resultierende Wahrnehmungsschwelle beispielhaft für ein konkretes Signalspektrum dargestellt. Diese wird auch als Mithörschwelle bezeichnet ([3], S. 22).

Wegen der Frequenzabhängigkeit der genannten Eigenschaften wird deutlich, warum eine Zerlegung des Audiosignals in Frequenzbänder sinnvoll ist. So kann z.B. ermittelt werden, ob der Schalldruck in einem Band unterhalb der Ruhehörschwelle liegt. Ist dies der Fall, so trägt dieses Band für das menschliche Hören keine relevante Information. Eine für den Kodierprozeß weit wichtigere Ausnutzung der Mithörschwelle wird noch im Abschnitt 2.5 erläutert.

Das psycho-akustische Modell des *Encoders* realisiert einen Algorithmus zur näherungsweisen Berechnung der Mithörschwelle. Durch den Standard wird kein verbindlicher Algorithmus festgelegt. In seinem informativen Teil werden lediglich zwei mögliche Ansätze vorgestellt. Beide Modelle ermitteln für jedes Band den Signalschalldruck und den kleinsten Wert der Mithörschwelle. Der Quotient beider Werte ergibt das Signal-Maskierungs-Verhältnis (engl. *signal-to-mask ratio*, *SMR*). Die 32 *SMR*-Werte dienen der nachfolgend beschriebenen Bit-Allokation.

2.5 Die Bit-Allokation

Für einen zu kodierenden Rahmen steht eine beschränkte Anzahl von Bits zur Verfügung. Diese ist von der Abtastfrequenz und der angestrebten Bitrate des kodierten Datenstroms abhängig.

Die Bit-Allokation legt fest, mit wievielen Bits die *Sub-band samples* der einzelnen Frequenzbänder im Rahmen kodiert werden, wobei die zur Verfügung stehende Gesamtanzahl von Bits nicht überschritten werden darf. Die Kodierung der *Sub-band samples* stellt eine erneute Quantisierung dar, welche im Abschnitt 2.6 noch genauer beschrieben wird. Durch die dabei entstehenden Quantisierungsfehler (Quantisierungsrauschen) kommt es zu Informationsverlust. Dieser kann auch als Abnahme des Signal-Rausch-Verhältnisses (*signal-to-noise ratio*, *SNR*) dargestellt werden ([5], Abschn. 3.2). Mit steigender Anzahl der Bits vergrößert sich auch dieses Verhältnis. Aus dem Verhältnis des *SNR* für eine bestimmte Anzahl von Bits und des *SMR* aus dem psycho-akustischen Modell ergibt sich das Verhältnis der Mithörschwelle zum Quantisierungsrauschen (*mask-to-noise ratio*, *MNR*).

Das *MNR* ist eine Aussage dafür, ob das Quantisierungsrauschen in einem Band oberhalb der Mithörschwelle liegt oder nicht.

Mit Hilfe der *MNR* verteilt die Bit-Allokation die zur Verfügung stehenden Bits so, daß das Quantisierungsrauschen in den einzelnen Bändern möglichst unter der Mithörschwelle liegt. Ist diese Forderung aufgrund der beschränkten Anzahl von Bits nicht zu erfüllen, so soll die Summe des hörbaren Quantisierungsrauschens der einzelnen Bänder minimal werden. Auf diese Weise wird versucht, die geforderte Transparenz der Datenreduktion zu erreichen.

2.6 Der Quantisierer

Der Quantisierer transformiert die *Sub-band samples* in den Wertebereich, welcher durch die von der Bit-Allokation bestimmte Bit-Anzahl festgelegt ist. Mit Hilfe der Skalierungsfaktoren werden die Werte in das Intervall $[-1, +1]$ transformiert, wie dies bereits im Abschnitt 2.3 beschrieben wurde. Es erfolgt eine weitere Transformation dieser Werte X nach der Formel:

$$Y = A \cdot X + B.$$

A und B sind Konstanten, welche für die jeweilige Bit-Anzahl N aus einer definierten Tabelle entnommen werden. Der additive Wert B vermeidet die Quantisierung kleiner Werte um den Nullpunkt in verschiedene Stufen. Den quantifizierten Wert erhält man durch Entnahme der N höchstwertigen Bits von Y.

Bei Schicht 2 werden jeweils drei aufeinanderfolgende Werte eines Bandes zu einem Kodewort zusammengefaßt, wenn diese mit 3, 5 oder 9 Bits quantifiziert wurden. Die entsprechenden Formeln lauten:

$$\begin{aligned} v_3 &= 9 \cdot z + 3 \cdot y + x, & v_3 &\in (0 \ldots 26) \\ v_5 &= 25 \cdot z + 5 \cdot y + x, & v_5 &\in (0 \ldots 124) \\ v_9 &= 81 \cdot z + 9 \cdot y + x, & v_9 &\in (0 \ldots 728). \end{aligned}$$

Durch diese Gruppierung (engl. *grouping*) erhält man pro Gruppe eine Einsparung von einem bzw. zwei Bits.

2.7 Die Formatierung

Der Aufbau eines Rahmens ist im Standard [4] für die verschiedenen Schichten genau definiert. Die Formatierung stellt entsprechend der vorgegebenen Syntax den Rahmen zusammen. Jeder Rahmen wird mit einem *Header* eingeleitet. Dieser besteht aus Synchronisations- und Verwaltungsinformationen, wie z.B. Bitrate, Abtastfrequenz und Modus. Optional kann dem *Header* eine Prüfsumme folgen. Es folgen die Werte der Bit-Allokation, welche die Information enthalten, ob bzw. mit wievielen Bits die *Sub-band samples* eines Bandes kodiert sind. Alle folgenden Informationen werden nur für Bänder mit einer Bit-Allokation größer als Null kodiert. Für die Schicht 2 wird nachfolgend kodiert, welche Kombinationen an Skalierungsfaktoren kodiert sind (siehe Abschnitt 2.3). Anschließend werden für beide Schichten die Indizes der Skalierungsfaktoren kodiert, gefolgt von den quantisierten *Sub-band samples*. Der Standard ermöglicht die Kodierung von anwenderspezifischen Informationen (*ancillary data*), welche den Rahmen abschließend auffüllen können.

2.8 Unterschiede der verschiedenen Modi

Die Ausführungen der vorangegangenen Abschnitte bezogen sich auf die Kodierung eines Audiokanals, wie sie bei Monosignalen Anwendung findet. Der entsprechende Modus wird als *single channel mode* bezeichnet.

Wie bereits in der Kapiteleinleitung erwähnt, ist auch die Kodierung von Stereo- und Zweikanalsignalen möglich. Zu deren Kodierung stehen der *stereo mode* und der *dual channel mode* zur Verfügung. Beide Modi sind, bis auf die Kodierung des Modus im *Header*, identisch. Eingangspuffer, Analysefilterbank, die Bestimmung der Skalierungsfaktoren und das psycho-akustische Modell sind bei diesen Modi für beide Kanäle getrennt ausgeführt. Die Bit-Allokation erfolgt wie im Abschnitt 2.5 beschrieben, wobei aber die Bit-Anzahl für die *Sub-band samples* beider Kanäle anhand der *MNR*-Werte bestimmt wird. Für die Formatierung gilt die Beschreibung des letzten Abschnitts, wobei sich die beiden Kanäle jedoch wertweise abwechseln.

Ein weiterer Modus für Stereosignale, der *joint stereo mode*, nutzt eine bisher noch nicht genannte Eigenschaft des menschlichen Gehörs aus. Die räumliche Wahrnehmung des Menschen hängt für Signale ab etwa 2 kHz mehr von der Hüllkurve der Signale beider Kanäle ab, als vom genauen Frequenzspektrum [7, 8]. Die Berechnung der Analysefilterbank, der Skalierungsfaktoren und des psycho-akustischen Modells erfolgt auch beim *joint stereo mode* für beide Kanäle getrennt. Im weiteren Kodierprozeß werden ab einem bestimmten Frequenzband die korrespondierenden *Sub-band samples* beider Kanäle zu einem Durchschnittswert zusammengefaßt. Für die Bit-Allokation und die Quantisierung werden die entsprechenden Bänder als einkanalig aufgefaßt. Auch bei der Formatierung werden die Werte der Bit-Allokation, sowie die *Sub-band samples* dieser Bänder nur einkanalig kodiert. Lediglich die Indizes der Skalierungsfaktoren werden weiterhin für alle Bänder zweikanalig kodiert. Da die Skalierung der zusammengefaßten Werte weiterhin getrennt erfolgt, bleibt die Hüllkurve der ursprünglichen Signale beider Kanäle im wesentlichen erhalten. Durch die Zusammenlegung von Werten stehen mehr Bits für die Quantisierung der *Sub-band samples* zur Verfügung. Deshalb kann bei gleicher Bitrate ein gegenüber dem *stereo mode* geringeres Quantisierungsrauschen erzielt werden. Dieser Vorteil wird mit dem Verlust von Stereo-Information erkauft, kann aber dennoch zu subjektiv besseren Ergebnissen führen.

Kapitel 3

Design und Algorithmen

3.1 Ein schneller Algorithmus für die Analysefilterbank

Hinsichtlich der Rechenkomplexität zählt die Analysefilterbank, neben dem psycho-akustischen Modell, zu den anspruchsvollsten Komponenten des *Encoders*. Ihre Rechenvorschrift ist im Anhang C des Standards [4] beschrieben und im Algorithmus 1 widergegeben.

Einen Ansatzpunkt zur algorithmischen Optimierung bietet die Matrix-Vektor-Multiplikation im Schritt 5. In [8] ist beschrieben, wie der Rechenaufwand in diesem Schritt nahezu halbiert werden kann. Hierzu wird der Vektor Y auf folgende Weise umgeformt:

$$\begin{aligned} Y_i^* &= Y_{16} & ,i = 0; \\ &= Y_{i+16} + Y_{16-i} & ,i = 1\ldots16; \\ &= Y_{i+16} - Y_{80-i} & ,i = 17\ldots31. \end{aligned} \tag{3.1}$$

Die Berechnung der *Sub-band samples* ergibt sich mit

$$S_i = \sum_{k=0}^{32} M_{ik}^* \cdot Y_k^* \quad ,i = 0\ldots31; \tag{3.2}$$

wobei gilt

$$M_{ik}^* = \cos\frac{(2i+1)k\pi}{64}.$$

Gleichung 3.2 entspricht einer Form der unnormalisierten inversen diskreten Kosinus-Transformation (engl. *inverse discrete cosine transform*, *iDCT*),

Algorithmus 1 Berechnung der Analysefilterbank nach [4]

1. Verschieben der Werte im Puffer X der Länge 512

$$X_i = X_{i-32} \quad , i = 511 \ldots 32$$

2. Einfügen von 32 neuen Abtastwerten in X

$$X_i = next\_sample \quad , i = 31 \ldots 0$$

3. Skalierung mit dem festgelegten Vektor C

$$Z_i = C_i \cdot X_i \quad , i = 0 \ldots 511$$

4. Bildung von Teilsummen

$$Y_i = \sum_{j=0}^{7} Z_{i+64j} \quad , i = 0 \ldots 63$$

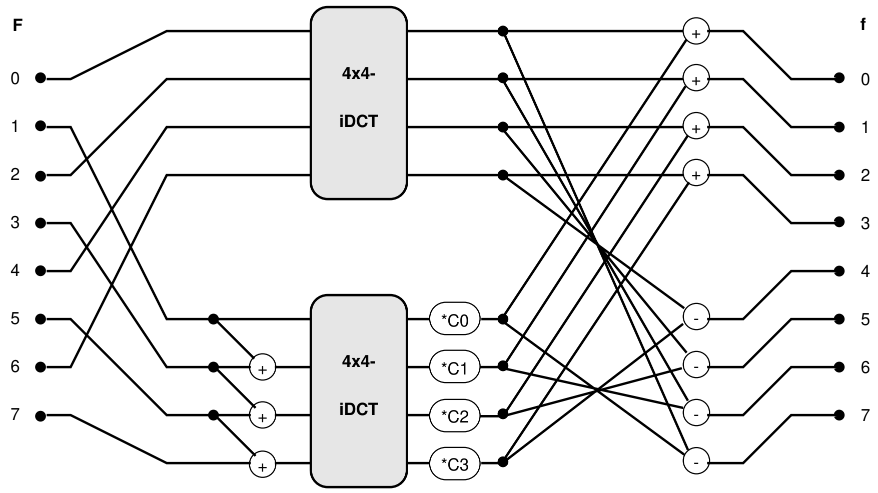
5. Berechnung der *Sub-band samples* S_i

$$S_i = \sum_{k=0}^{63} M_{ik} \cdot Y_k \quad , i = 0 \ldots 31$$

mit den Koeffizienten

$$M_{ik} = \cos \frac{(2i+1)(k-16)\pi}{64}$$

Abbildung 3.1 Rückführung einer 8-Punkt-*iDCT* auf zwei 4-Punkt-*iDCTs*



$$c_n = \frac{1}{2\cos\frac{(2n+1)\pi}{16}} \quad ,n = 0\ldots 4$$

welche allgemein durch die Formel

$$f_i = \sum_{k=0}^{N-1} F_k \cdot \cos\frac{(2i+1)k\pi}{2N} \quad ,i = 0\ldots N-1 \tag{3.3}$$

beschrieben wird. Für diese Transformation existieren einige schnelle Algorithmen. So ist z.B. eine Berechnung über die Schnelle Fourier-Transformation (engl. *fast Fourier transform*, *FFT*) möglich. Noch besser geeignet ist der Algorithmus von *B. G. Lee* [9]. Er basiert auf der Zerlegung einer N-Punkt-*iDCT* auf zwei $\frac{N}{2}$-Punkt-*iDCTs*. Abbildung 3.1 zeigt als Beispiel die Zerlegung einer 8-Punkt-*iDCT* auf zwei 4-Punkt-*iDCTs*. Das Verfahren kann rekursiv angewendet werden, so daß eine N-Punkt-*iDCT* auf $\frac{N}{2}$ 2-Punkt-*iDCTs* rückgeführt wird. Algorithmus 2 beschreibt den rekursiven Ablauf zur schnellen Berechnung der *iDCT* nach *B. G. Lee*.

Legt man einem Komplexitätsvergleich der verschiedenen Algorithmen die Anzahl der Multiplikationen und Additionen bzw. Subtraktionen zugrunde, so ergeben sich die in Tabelle 3.1 abgebildeten Werte. Alle im voraus berechenbaren Werte, wie Matrix- und Skalierungskoeffizienten, bleiben bei diesem Vergleich unberücksichtigt. Aufgrund der wesentlich geringeren Komplexität des *Lee*-Algorithmus erscheint dieser für eine schnelle Berechnung der Matrix-Vektor-Multiplikation am besten geeignet. Der Vergleich

Algorithmus 2 Rekursive Funktion $f = \mathrm{idct}(F, N)$ zur schnellen Berechnung der $iDCT$ nach *Lee*

Bedingungen: $N = 2^n \ \wedge \ n \in \mathbb{N} \ \wedge \ n > 0$
Berechnet: $f_i = \sum_{k=0}^{N-1} F_k \cdot \cos \frac{(2i+1)k\pi}{2N} \quad , i = 0 \ldots N-1$
  if $N \equiv 2$ **then** {Endebedingung}
    $f_0 = F_0 + F_1 \cdot \cos \frac{\pi}{4}$
    $f_1 = F_0 + F_1 \cdot \cos \frac{3\pi}{4} = F_0 - F_1 \cdot \cos \frac{\pi}{4}$
  else {Zerlegung in Teilvektoren F^e und F^o der Länge $\frac{N}{2}$}
    for $i = 0$ to $\frac{N}{2} - 1$ **do**
      $F_i^e = F_{2i}$
      $F_i^o = F_{2i+1}$
    end for
    for $i = \frac{N}{2} - 1$ to 1 **do**
      $F_i^o = F_i^o + F_{i-1}^o$
    end for
    $f^e = \mathrm{idct}(F^e, \frac{N}{2})$ { rekursive Berechnung der $iDCT$ für F^e }
    $f^o = \mathrm{idct}(F^o, \frac{N}{2})$ { rekursive Berechnung der $iDCT$ für F^o }
    for $i = 0$ to $\frac{N}{2} - 1$ **do**
      $f_i^o = f_i^o \cdot \frac{1}{2\cos\frac{(2i+1)\pi}{2N}}$
    end for
    for $i = 0$ to $\frac{N}{2} - 1$ **do** { Berechnung des Ergebnisvektors f }
      $f_i = f_i^e + f_i^o$
      $f_{N-i-1} = f_i^e - f_i^o$
    end for
  end if

Tabelle 3.1 Komplexität verschiedener Algorithmen zur Berechnung des Schrittes 5 der Analysefilterbank

Algorithmus	Komplexität	relative Komplexität
nach Standard	4064	199 %
Rückführung auf *iDCT*	2047	100 %
iDCT, Alg. nach *Lee*	320	16 %

verschiedener Implementationen im Abschnitt 4.2.2 wird zeigen, inwieweit die in Tabelle 3.1 wiedergegebenen Abschätzungen mit den realen Rechenzeiten korrelieren.

3.2 Ein Suchalgorithmus für die Skalierungsfaktoren

Die Bedeutung der Skalierungsfaktoren und ihre Bestimmung wurden bereits eingehend im Abschnitt 2.3 erläutert. Während die Ermittlung des betragsmäßig größten Wertes der *Sub-band samples* keine algorithmischen Alternativen bietet, gibt es verschiedene Algorithmen zur Suche des passenden Tabelleneintrags. Die einfachste Variante ist die lineare Suche, welche aber viele Nachteile besitzt. Ihre Laufzeit ist stark datenabhängig. So ist im günstigsten Fall ein Suchschritt ausreichend, während der ungünstigste Fall 62 Schritte benötigt. Die durchschnittliche Komplexität beträgt bei 63 Tabelleneinträgen 31. Dieser Algorithmus ist daher nicht optimal.

Die im Standard [4] definierte Tabelle B.1 für die Skalierungsfaktoren ist eine absteigende Wertefolge. Deshalb kann als alternativer Algorithmus die Intervallhalbierung (Bisektion) angewendet werden. Mit fünf Intervallhalbierungen wird das Suchintervall soweit eingeschränkt, daß die obere Intervallgrenze mit dem gesuchten Wert übereinstimmt. Die Vorteile dieses Verfahrens liegen in der geringen Komplexität und in der Datenunabhängigkeit.

3.3 Algorithmen zur Bit-Allokation

Im Abschnitt 2.5 wurde die Aufgabe der Bit-Allokation bereits einführend erläutert. Schwerpunkt dieses Abschnitts ist ihre algorithmische Realisierung.

Der Standard [4] beschreibt das im Algorithmus 3 wiedergegebene Verfahren, um die *Sub-band samples* nach den im Abschnitt 2.5 erläuterten Kriterien optimal zu kodieren. Die Anzahl der verfügbaren Bits kann aus der Länge des Rahmens berechnet werden, welche sich nach der Formel

$$N = K \cdot \frac{\text{Bitrate}}{\text{Abtastfrequenz}} \tag{3.4}$$

ergibt. Die Konstante K beträgt 12 für Schicht 1 und 144 für Schicht 2. Ergibt N keine ganze Zahl, so wird nach einem bestimmten Verfahren auf- oder abgerundet (siehe [4], S. 22). N gibt die Länge des Rahmens in *Slots* wieder, welche als Einheiten von 32 Bits bei Schicht 1 bzw. 8 Bits bei Schicht 2 definiert sind. Von der berechneten Bit-Anzahl wird eine feste Menge für den *Header*, die optionale Prüfsumme und optionale anwenderspezifische Daten benötigt (siehe Abschnitt 2.7). Die Kodierung der Bit-Allokationswerte braucht ebenfalls eine konstante Anzahl von Bits und muß berücksichtigt werden. Die verbleibenden Bits stehen der Kodierung von Skalierungsfaktoren und *Sub-band samples* zur Verfügung. Algorithmus 3 verbraucht in jeder Iteration eine bestimmte Anzahl von Bits. Sollen z.B. bei Schicht 1 die *Sub-band samples* eines Bandes mit jeweils einem Bit mehr kodiert werden, so werden insgesamt 12 weitere Bits benötigt, da in einem Rahmen pro Band zwölf *Sub-band samples* kodiert werden. Auf diese Weise wird in jeder Iteration eine kleine Anzahl von Bits verbraucht.

Offensichtlich hängt die Anzahl der Iterationen in starkem Maße von der Anzahl verfügbarer Bits ab, welche nach Formel 3.4 proportional zur Bitrate ist. Dieser Zusammenhang wurde auch im Laufzeitverhalten der Implementation deutlich. Abbildung 3.2 zeigt die durchschnittliche Laufzeit der Bit-Allokation bei Schicht 1 in Abhängigkeit von der Bitrate. Die als Version 1 gekennzeichnete Kurve entspricht der Implementation des Algorithmus 3.

Da die Laufzeit für hohe Bitraten nicht akzeptabel war, wurde nach Alternativen gesucht. Eine Optimierung bestand darin, den Schritt 2 des Algorithmus 3 zu erweitern. Neben dem Band k mit dem kleinsten *MNR*-Wert wird auch das Band k^* mit dem zweitkleinsten *MNR*-Wert gesucht. Die Schritte 3 bis 5 können nun solange für das Band k ausgeführt werden, bis dessen aktualisierter *MNR*-Wert größer ist als der des Bandes k^* oder die Anzahl verfügbarer Bits erschöpft ist. Sind noch Bits verfügbar, so wird mit den Schritten 3 bis 6 für das Band k^*, welches nun den kleinsten *MNR*-

Algorithmus 3 Bit-Allokation nach [4]

1. Initialisierung

    - Setze die Anzahl der Quantisierungsbits b_i für alle Bänder i auf Null.
    - Berechne das Maskierungs-Rausch-Verhältnis (MNR) für alle Bänder nach der Formel

    $$MNR_i = SNR_{b_i} - SMR_i.$$

    - Berechne die Anzahl der zur Verfügung stehenden Bits.

2. Suche das Band k mit dem kleinsten MNR.

3. Erhöhe die Anzahl der Bits b_k des Bandes k auf den nächsthöheren Wert.

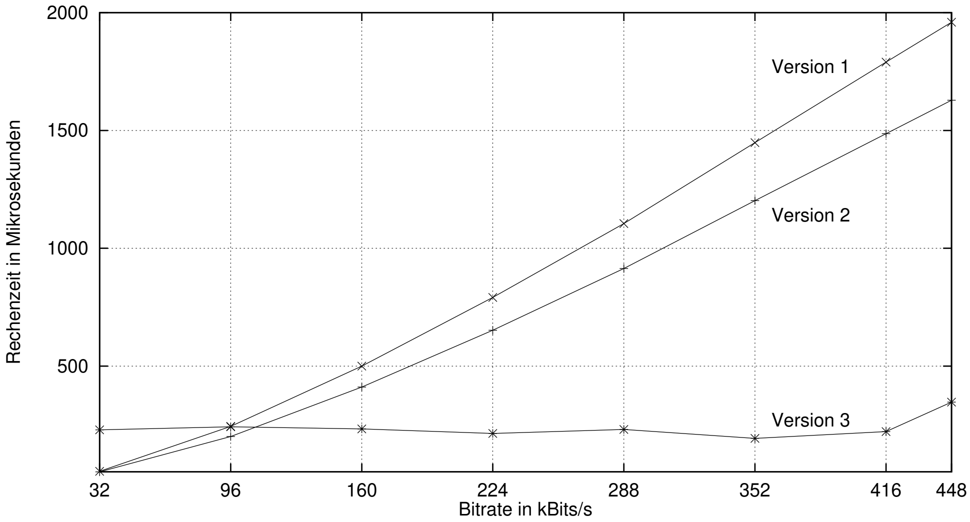
4. Aktualisiere das MNR des Bandes k

    $$MNR_k = SNR_{b_k} - SMR_k.$$

5. Verringere die Anzahl der verfügbaren Bits um die durch Schritt 3 verbrauchte Menge.

6. Gehe zu Schritt 2, wenn noch genug Bits zur Verfügung stehen, um die Quantisierungsstufe in einem Band zu erhöhen.

Abbildung 3.2 Durchschnittliche Laufzeit verschiedener Algorithmen der Bit-Allokation (Schicht 1, *single channel mode*) in Abhängigkeit von der Bitrate



Wert besitzt, wie gewohnt fortgefahren. Diese Optimierung verrringert die Anzahl der Ausführungen von Schritt 2. Dieser wird durch die zusätzliche Suche nach dem zweitkleinsten *MNR* nur unwesentlich komplexer. Das Laufzeitverhalten dieser Optimierung ist in Abbildung 3.2 als Version 2 gekennzeichnet. Wenn auch die Laufzeit im Vergleich zu Version 1 etwas günstiger ist, so besteht dennoch eine starke Abhängigkeit von der Bitrate.

Wie die bisher gemachten Feststellungen zeigen, ist ein völlig anderes Herangehen notwendig, um ein gleichmäßigeres Laufzeitverhalten zu erreichen. Im folgenden soll nun ein Algorithmus für die Schicht 1 vorgestellt werden, welcher dem gewünschten Ziel nahe kommt.

Wie bereits im Abschnitt 2.5 erklärt wurde, beschreibt das *MNR* das Verhältnis der Mithörschwelle zum Quantisierungsrauschen. Eine optimale Kodierung liegt vor, wenn dieses Verhältnis für alle Bänder 1 (0 dB) ist. Es ergibt sich die minimale Bit-Anzahl, bei der das Quantisierungsrauschen gerade noch nicht wahrnehmbar ist. Das ist der Ausgangspunkt des neuen Algorithmus. In einem ersten Schritt wird die Anzahl der Quantisierungsbits in den einzelnen Bändern so festgelegt, daß die resultierenden *MNR* gera-

Algorithmus 4 Alternative Bit-Allokation

1. Weise jedem Band so viele Quantisierungsbits zu, daß das resultierende Maskierungs-Rausch-Verhältnis *MNR* gleich 0 dB ist oder knapp darüber liegt.

2. Berechne die benötigte Bit-Anzahl.

3. Ist die benötigte Bit-Anzahl kleiner als die verfügbare Anzahl, so gehe zu Schritt 6.

4. Reduziere in jedem Band mit einer Allokation größer Null die Anzahl der Quantisierungsbits um ein Bit und aktualisiere die Anzahl benötigter Bits.

5. Werden noch immer mehr Bits benötigt, als verfügbar sind, so gehe zu Schritt 4.

6. Wenn noch Bits verfügbar sind, dann fahre fort mit Algorithmus 3.

---

de 0 dB ergeben oder knapp darüber liegen. Nun ist zu prüfen, ob die für diese Festlegung benötigte Bit-Anzahl zur Verfügung steht. Sind genug Bits vorhanden, so können eventuell noch verbleibende Bits mit Hilfe von Algorithmus 3 zugewiesen werden. Schwieriger ist es, wenn mehr Bits benötigt werden als verfügbar sind. In diesem Fall muß die benötigte Bit-Anzahl geeignet reduziert werden.

Algorithmus 3 bewirkt, daß sich die *MNR*-Werte der einzelnen Bänder langsam auf ein einheitliches Niveau nähern. Ist dies erreicht, so steigt das Niveau insgesamt. Durch den ersten Schritt des neuen Algorithmus wurde bereits ein einheitliches Niveau erreicht, welches aber offensichtlich zu hoch ist. Geht man davon aus, daß die Reduzierung um ein Quantisierungsbit das *MNR* in allen Bändern um etwa den gleichen Betrag verringert, so bleibt ein einheitliches Niveau erhalten und die benötigte Bit-Anzahl verringert sich. Dieser Schritt wird wiederholt, bis die Anzahl der benötigten Bits kleiner oder gleich der Anzahl der verfügbaren Bits ist. Sind nach der Reduzierung noch Bits verfügbar, so werden diese mit Hilfe von Algorithmus 3 vergeben. Die alternative Bit-Allokation ist als Algorithmus 4 abgebildet. Das Laufzeitverhalten der entsprechenden Implementation beschreibt die als Version

3 bezeichnete Kurve in Abbildung 3.2.

Es wurde bereits erwähnt, daß dieser Algorithmus für die Schicht 1 geeignet ist. Eine Anwendung bei Schicht 2 ist nicht möglich. Während bei Schicht 1 die möglichen Quantisierungsstufen für alle Bänder gleich sind, hängen diese bei Schicht 2 von der Nummer des Bandes ab. Daraus folgt, daß eine Änderung der Quantisierungsbits nicht in allen Bändern die etwa gleiche Änderung des *MNR* bedingt.

Es ist anzumerken, daß Algorithmus 4 nicht in jedem Fall zum exakt gleichen Resultat führt wie Algorithmus 3. Diese Tatsache begründet sich darin, daß die Änderung des *MNR* bei Abziehen eines Quantisierungsbits nicht konstant ist, wie weiter oben angenommen wurde. Wird z.B. in einem Band die Anzahl von 5 auf 4 Bits verkleinert, so verringert sich das *MNR* um 6.31 dB. Wird gleichzeitig in einem anderen Band einer Verringerung von 3 auf 2 Bits vorgenommen, so resultiert daraus eine Verringerung um 9 dB. In den meisten Fällen sorgt der als letzter Schritt ausgeführte Algorithmus 3 für ein identisches Resultat. Da die als Ausgangswerte dienenden *SMR*-Werte des psycho-akustischen Modells bereits Schätzwerte sind, ist eine geringe Abweichung akzeptabel.

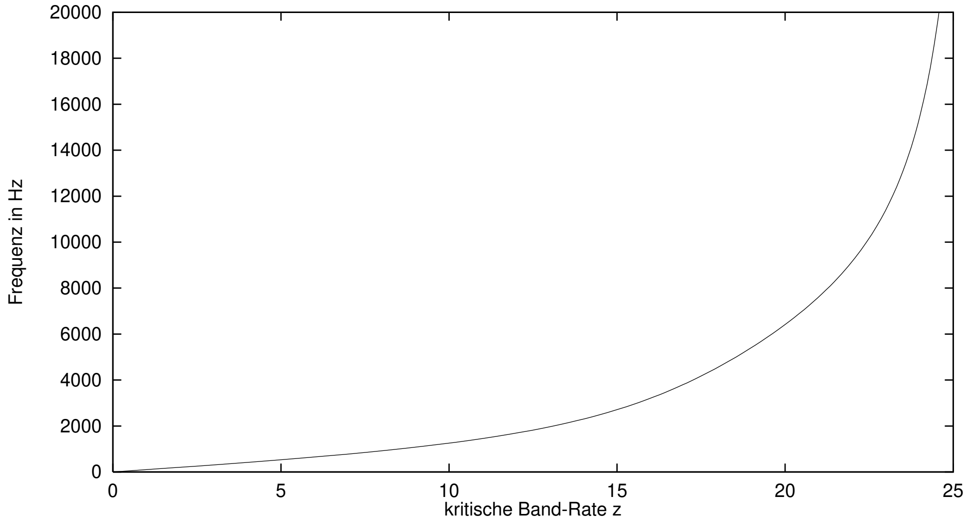
3.4 Realisierung des psycho-akustischen Modells

Der Anhang D des Standards [4] beschreibt zwei mögliche psycho-akustische Modelle. In diesem Abschnitt wird diskutiert, ob diese Modelle für eine echtzeitfähige Implementation geeignet sind oder entsprechend modifiziert werden können. Auf eine genaue Beschreibung der beiden Modelle wird verzichtet und auf den Standard [4] verwiesen. Die prinzipielle Funktion des psycho-akustischen Modells wurde im Abschnitt 2.4 beschrieben. Neben den Modellen des Standards wird auch ein Realisierungsversuch unter Verwendung eines künstlich-neuronalen Netzes ausgewertet.

3.4.1 Kritische Bänder

Die Modelle des Standards verwenden für einige Berechnungen eine Einteilung des Frequenzspektrums in kritische Bänder (*critical bands*). Sie führen dafür die kritische Band-Rate z (*critical band rate*) ein, welche in einem nichtlinearen Zusammenhang zur Frequenz steht (siehe Abb. 3.3). Ein kritisches Band entspricht einer ganzen Einheit in der kritischen Band-Rate.

Abbildung 3.3 Funktionaler Zusammenhang zwischen Frequenz und kritischer Band-Rate



So entspricht z.B. das kritische Band 0 dem Bereich von $z = 0$ bis $z = 1$. Der Sinn dieser Darstellung ist die Linearisierung von Berechnungen. So ist z.B. die Breite des Maskierungsbereichs starker Signalkomponenten von der Frequenzlage abhängig. Bei niedrigen Frequenzen ist dieser Bereich wesentlich schmaler als bei hohen Frequenzen. Für die rechnerische Nachbildung der Maskierung kann unter Verwendung der kritischen Band-Rate trotzdem eine einheitliche Funktion verwendet werden.

3.4.2 Das psycho-akustische Modell 1

Unter Verwendung des psycho-akustischen Modells 1 gelang es, eine echtzeitfähige Implementation zu entwickeln. Im Standard sind die Rechenschritte im Dämpfungsmaß Dezibel (dB) beschrieben, welches für Leistungsverhältnisse als

$$a = 10 \cdot \lg \frac{P_1}{P_2} \tag{3.5}$$

berechnet wird (siehe [5], S. 34). Ein optimiertes, in Assembler geschriebenes Programm benötigt für die Berechnung des Dämpfungsmaßes etwa 40 Befehlszyklen, während ein Programm zur Rücktransformation in die lineare Darstellung etwa 30 Zyklen benötigt. Aufgrund ihrer Häufigkeit wird sehr viel Rechenzeit für diese Transformationen verbraucht. So ist z.B. bei Schicht 1 im ersten Schritt des Modells das Dämpfungsmaß von 256 Werten zu berechnen. Bei einem Prozessortakt von 50 MHz benötigt der TMS320C44 für diese Berechnung etwa 0.41 ms. Grundlage einer echtzeitfähigen Implementation war deshalb eine Berechnung des gesamten Modells in der linearen Wertedarstellung, um die für die Transformationen benötigte Rechenzeit einzusparen. Im folgenden wird auf die Rechenschritte des Modells einzeln eingegangen. Um in Formeln einfach zwischen dem Dämpfungsmaß und der linearen Domäne zu unterscheiden, erhalten Werte im Dämpfungsmaß den hochgestellten Index dB, wobei gilt

$$X^{dB} = 10 \cdot \lg X \ .$$

Schritt 1: Diskrete Fourier-Transformation (*FFT*)

Die sehr grobe Frequenzaufteilung der Analysefilterbank ist für das psychoakustische Modell nicht ausreichend. Deshalb wird im ersten Schritt mittels einer diskreten Fourier-Transformation ein genaueres Frequenzspektrum berechnet. Die diskrete Fourier-Transformation ist eine Standardberechnung der digitalen Signalverarbeitung und kann mit Hilfe des als "Schnelle Fourier-Transformation" (*fast Fourier transform*, *FFT*) bekannten Algorithmus effizient berechnet werden (siehe [5], Abschnitte 4.2 und 4.3). Für den TMS320 C44 existieren mehrere Implementationen zur Berechnung der *FFT*. Verwendung fand das in [10] als Beispiel 12-42 (S. 12-80 ff.) abgedruckte Programm. Für die bei Schicht 1 verwendete 512-Punkt-*FFT* benötigt ein TMS320C44 mit 50 MHz Taktfrequenz 0.345 Millisekunden. Die bei Schicht 2 verwendete 1024-Punkt-*FFT* wird in 0.776 Millisekunden berechnet (Werte nach Tabelle 12-3 in [10]). Für die weiteren Schritte ist nur die erste Hälfte des durch die *FFT* berechneten diskreten Frequenzspektrums relevant, welche vom Gleichanteil bis zur halben Abtastfrequenz reicht. Somit ergeben sich 256 Ausgangswerte bei Schicht 1 bzw. 512 Ausgangswerte bei Schicht 2. Gemäß der Modellbeschreibung ist das Spektrum so zu normalisieren, daß der größte Wert 96 dB entspricht. Der Normalisierungsfaktor ergibt sich mit

$$n^{dB} = 96 - X^{dB}_{max}$$

und wird zu allen Werten des Spektrums addiert, wobei X^{db}_{max} der Maximalwert im Spektrum ist. Unter Anwendung der Logarithmengesetze kann diese Formel in die lineare Darstellung überführt werden und lautet dann

$$n = \frac{10^{\frac{96}{10}}}{X_{max}} .$$

In der linearen Darstellung erfolgt die Normalisierung durch Multiplikation mit dem Faktor n.

Schritt 2: Berechnung des Schalldrucks

Für die Ermittlung des Schalldrucks der einzelnen Frequenzbänder werden zwei alternative Formeln angegeben, von denen die zweite Berechnungsvariante gewählt wurde. Diese lautet

$$L_{sb}^{dB}(n) = max(X_{spl}^{dB}(n),\ 20 \cdot \lg(scf_{max}(n) \cdot 32768) - 10) \text{ , mit } n = 0 \ldots 31.$$

Der Wert $scf_{max}(n)$ ist bei Schicht 1 der Skalierungsfaktor des Bandes n und bei Schicht 2 der größte der drei Skalierungsfaktoren des Bandes n (siehe Abschnitt 2.3). $X_{spl}^{dB}(n)$ wird aus dem Frequenzspektrum nach der Formel

$$X_{spl}^{dB}(n) = 10 \cdot \lg(\sum_k 10^{\frac{X^{dB}(k)}{10}})$$

berechnet, wobei die Laufvariable k über die zum Band n korrespondierenden Spektralwerte läuft. Diese Formel vereinfacht sich in der linearen Darstellung zu

$$X_{spl}(n) = \sum_k X(k) .$$

In der Implementation wurde auf die Berücksichtigung der Skalierungsfaktoren verzichtet, so daß gilt $L_{sb}(n) = X_{spl}(n)$.

Schritt 3: Ruhehörschwelle

Für die verschiedenen Schichten und Abtastraten definiert der Standard Tabellen mit Werten der Ruhehörschwelle (siehe Abschnitt 2.4). Um in der linearen Domäne zu verbleiben, wurden die Tabellen entsprechend übertragen. Ist die Zielbitrate größer oder gleich 96 kBits/s, so sind die Tabellenwerte mit einem Offset von -12 dB zu addieren. In der linearen Domäne entspricht das einer Multiplikation mit dem Wert $10^{\frac{-12}{10}}$.

Schritt 4: Ermittlung von Ton- und Rausch-Komponenten

Für die weiteren Berechnungen ist eine Unterscheidung zwischen deterministischen Ton-Komponenten (*tonal components*) und stochastischen Rausch-Komponenten (*non-tonal components*) vorzunehmen. In einem ersten Schritt werden Spektrallinien als lokale Maxima des Spektrums markiert, wenn sie die folgende Bedingung erfüllen:

$$X^{dB}(k) > X^{dB}(k-1) \ \wedge \ X^{dB}(k) \geq X^{dB}(k+1) \ .$$

Die Laufvariable k läuft dabei über das gesamte Spektrum. Die Ungleichungen behalten in der linearen Domäne ihre Gültigkeit, da der dekadische Logarithmus eine stetige, monoton steigende Funktion ist. Dieser Teilschritt bleibt daher unverändert.

In einem weiteren Teilschritt wird untersucht, ob die gefundenen Maxima der folgenden Bedingung genügen:

$$X^{dB}(k) - X^{dB}(k+j) \geq 7 \ dB \ , \text{ mit } j = \pm 2 \ldots \pm J.$$

Der Index k kennzeichnet das zu untersuchende Maximum, während J für die verschiedenen Schichten und verschiedenen Abschnitten des Spektrums unterschiedlich festgelegt wird (siehe [4], S. 112). Stellt man die Ungleichung etwas um, so kann diese Bedingung in der linearen Domäne wie folgt ausgedrückt werden:

$$10^{\frac{-7}{10}} \cdot X(k) \geq X(k+j) \ .$$

Ist die Bedingung erfüllt, so wird eine neuer Eintrag

$$X_{tm}^{db}(k) = 10 \cdot \lg(10^{\frac{X^{dB}(k-1)}{10}} + 10^{\frac{X^{dB}(k)}{10}} + 10^{\frac{X^{dB}(k+1)}{10}})$$

an der Stelle k in eine Liste der Ton-Komponenten eingetragen. Weiterhin werden alle Spektrallinien im Bereich $k-J$ bis $k+J$ auf $-\infty$ dB gesetzt, was in der linearen Domäne einem Nullsetzen entspricht. Die Berechnung der Ton-Komponente vereinfacht sich zu

$$X_{tm}(k) = X(k-1) + X(k) + X(k+1) \ .$$

Ist das gesamte Spektrum nach Ton-Komponenten analysiert worden, so werden die verbleibenden, nicht Null gesetzten Spektralkomponenten zur

Berechnung der Rausch-Komponenten verwendet. Hierzu werden jeweils die Komponenten summiert, welche in einem kritischen Band liegen. Diese Summenwerte X_{nm} werden einer Spektralliniennummer k zugeordnet, deren kritische Band-Rate z etwa der Mitte des kritischen Bandes entspricht und in eine Liste der Rausch-Komponenten eingetragen.

Schritt 5: Reduzierung der Ton- und Rausch-Komponenten

Für die weiteren Berechnungen sind nur Komponenten relevant, welche oberhalb der Ruhehörschwelle liegen. In der Beschreibung von Schritt 3 wurde bereits auf die Ruhehörschwelle eingegangen. In diesem Schritt werden alle Komponenten aus den Listen entfernt, welche die folgende Ungleichung nicht erfüllen:

$$X_{tm}^{dB}(k) \geq LT_q^{dB}(k) \ \ \text{bzw.} \ \ X_{nm}^{dB}(k) \geq LT_q^{dB}(k) \ .$$

Der Ausdruck $LT_q^{dB}(k)$ ist der Wert der Ruhehörschwelle bei der zur Spektrallinie k korrespondierenden Frequenz. Diese Ungleichungen gelten unverändert in der linearen Domäne.

Schritt 6: Die Maskierungsschwelle

Die Maskierungsschwelle bildet den im Abschnitt 2.4 beschriebenen Maskierungseffekt mathematisch nach. Sie berechnet sich für Ton-Komponenten

$$LT_{tm}^{dB}(j,i) = X_{tm}^{dB}(j) + \mathrm{av}_{tm}^{dB}(j) + \mathrm{vf}^{dB}(j,i)$$

und für Rausch-Komponenten

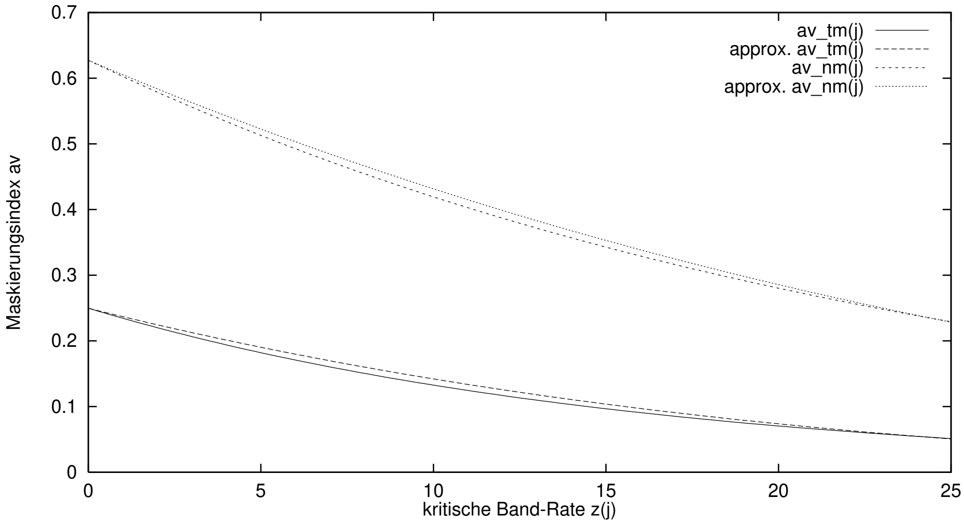
$$LT_{nm}^{dB}(j,i) = X_{nm}^{dB}(j) + \mathrm{av}_{nm}^{dB}(j) + \mathrm{vf}^{dB}(j,i) \ .$$

Der Wert $X_{tm}^{dB}(j)$ bzw. $X_{nm}^{dB}(j)$ ist die maskierenden Komponente mit dem Index j. Der Index i entspricht der Spektrallinie, für die die Maskierung errechnet wird. Auf die Funktionen av und vf wird später eingangen. Die Maskierungsschwelle kann mit Hilfe der Logarithmengesetze einfach in die lineare Domäne übertragen werden:

$$LT_{tm}(j,i) = X_{tm}(j) \cdot \mathrm{av}_{tm}(j) \cdot \mathrm{vf}\,(j,i) \ \ \text{bzw.}$$

$$LT_{nm}(j,i) = X_{nm}(j) \cdot \mathrm{av}_{nm}(j) \cdot \mathrm{vf}\,(j,i) \ .$$

Abbildung 3.4 Verlauf der av-Funktionen und ihrer Approximationen



Man erkennt, daß auch die Teilfunktionen av und vf in die lineare Repräsentation übertragen werden müssen.

Die Maskierungsindex (*masking index*) genannte Funktion av ist definiert als

$$\mathrm{av}_{tm}^{dB}(j) = -1.525 - 0.275 \cdot z(j) - 4.5$$

für Ton-Komponenten und

$$\mathrm{av}_{nm}^{dB}(j) = -1.525 - 0.175 \cdot z(j) - 0.5$$

für Rausch-Komponenten. Der Wert $z(j)$ entspricht der kritischen Band-Rate der Komponente j. Diese Funktionen sind in der linearen Domäne nur aufwendig zu berechnen. Deshalb wurde auf eine genaue Übertragung verzichtet und eine Approximation mit einfacheren Funktionen durchgeführt. Die Approximationsfunktionen in der linearen Domäne lauten

$$\mathrm{av}_{tm}(j) = 7.4857 \cdot 10^{-9} \cdot (z(j) - 76)^4 \quad \text{bzw.}$$

$$\mathrm{av}_{nm}(j) = 3.9867 \cdot 10^{-9} \cdot (z(j) - 112)^4 \ .$$

Der Verlauf der Funktionen und ihrer Approximationen ist in Abbildung 3.4 dargestellt.

Auch die Berechnung der Maskierungfunktion vf ist in der linearen Domäne nur mit großem Rechenaufwand möglich. Der Standard definiert die Funktion wie folgt:

$$\mathrm{vf}^{dB} = \begin{cases} 17 \cdot (dz+1) - (0.4 \cdot X^{dB}(j) + 6) & , -3 \leq dz < -1 \\ (0.4 \cdot X^{dB}(j) + 6) \cdot dz & , -1 \leq dz < 0 \\ -17 \cdot dz & , 0 \leq dz < 1 \\ (dz-1) \cdot (17 - 0.15 \cdot X^{dB}(j)) - 17 & , 1 \leq dz < 8 \end{cases}$$

mit $dz = z(i) - z(j)$. Man erkennt, daß es sich um eine zweidimensionale Funktion handelt, welche von der maskierenden Komponente $X^{dB}(j)$ und dem Distanzwert dz abhängt. Zur Approximation der Funktion wurde eine Tabelle mit stückweise linearen Approximationsfunktionen verwendet. Aus dem linearen Wert der maskierenden Komponente $X(j)$ wird ein Index x berechnet, welcher zur Auswahl einer Approximationsfunktion aus der Tabelle dient. Die verwendete Formel lautet:

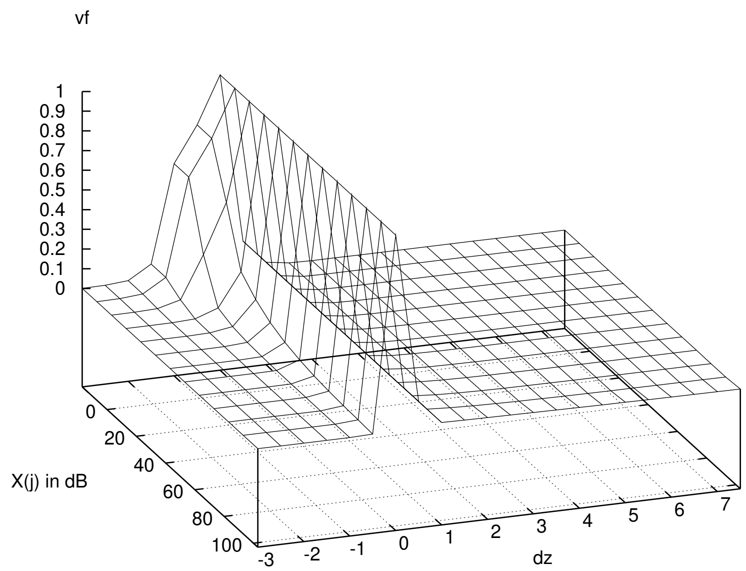
$$x = \mathrm{int}(0.3 \cdot \log_2 X(j) + 1.7) \ .$$

Die Funktion int soll die Umwandlung einer reellen Zahl in eine ganze Zahl durch Abschneiden der Kommastellen durchführen. Diese Formel realisiert eine Indexierung, welche in etwa Zehnerpotenzschritten bzw. Schritten von 10 dB entspricht, da gilt $lg\ x \approx 0.3 \cdot \log_2 x$. Der additive Wert 1.7 sorgt für eine Abbildung des kleinstmöglichen Wertes $X_{min} = -17$ dB auf den Index $x = 0$. X_{min} entspricht dem Minimalwert der Ruhehörschwelle, da im Schritt 5 alle Komponenten unterhalb dieser entfernt werden. Die durch den Index x ausgewählte Approximationsfunktion hängt nun ausschließlich von dz ab. Mit Hilfe eines weiteren Indexes $y = \mathrm{int}(4 \cdot (dz+3))$ wird der Anstieg a und die Verschiebung m eines Geradensegmentes der stückweise linearen Approximationsfunktion bestimmt. Aus der gewählten Indexberechnung für y ergibt sich, daß die Approximationsfunktion aus 44 Segmenten besteht, um den Definitionsbereich ($-3 \leq dz < 8$) abzudecken. Der endgültige Funktionswert berechnet sich nach der Formel

$$\mathrm{vf}\,(X(j), dz) = m_{xy} \cdot dz + a_{xy} \ .$$

Abbildung 3.5 skizziert die approximierte Funktion vf.

Abbildung 3.5 Approximierte Funktion vf



Schritt 7: Berechnung der Mithörschwelle LT_g

Die Berechnung der Mithörschwelle ist im Standard durch die Formel

$$LT_g^{dB}(i) = 10 \cdot \lg(10^{\frac{LT_q^{dB}(i)}{10}} + \sum_j 10^{\frac{LT_{tm}^{dB}(j,i)}{10}} + \sum_j 10^{\frac{LT_{nm}^{dB}(j,i)}{10}}) \ .$$

beschrieben. Der Index i läuft über alle Spektrallinien. $LT_q^{dB}(i)$ entspricht dem Wert der Ruhehörschwelle der Spektrallinie i. Der Laufparameter j der beiden Summen läuft über alle maskierenden Komponenten aus der Liste der Ton-Komponenten bzw. Rausch-Komponenten, welche die Bedingung erfüllen $-3 \leq z(i) - z(j) < 8$. $LT_{tm}^{dB}(j,i)$ und $LT_{nm}^{dB}(j,i)$ sind die im Schritt 6 beschriebenen Funktionen zur Berechnung der Maskierungsschwelle. Die Übertragung in die lineare Domäne vereinfacht die Berechnung der Mithörschwelle zu

$$LT_g(i) = LT_q(i) + \sum_j LT_{tm}(j,i) + \sum_j LT_{nm}(j,i) \ ,$$

wobei die Funktionen $LT_{tm}(j,i)$ und $LT_{nm}(j,i)$, wie im Schritt 6 beschrieben, approximiert werden.

Schritt 8: Berechnung der minimalen Mithörschwelle

Für jedes der 32 Frequenzbänder wird aus der Mithörschwelle LT_g der minimale Wert ermittelt:

$$LT_{min}^{dB}(n) = \min[LT_g^{dB}(i)] \ .$$

Der Index i läuft über alle im Frequenzband n liegenden Spektrallinien. Diese Minimumsuche bleibt in der linearen Domäne unverändert.

Schritt 9: Berechnung des Signal-Maskierungs-Verhältnisses *SMR*

Das SMR wird für jedes Frequenzband n aus dem im Schritt 2 ermittelten Schalldruck $L_{sb}(n)$ und der im vorhergehenden Schritt berechneten zugehörigen minimalen Mithörschwelle $LT_{min}(n)$ berechnet:

$$SMR_{sb}^{dB}(n) = L_{sb}^{dB}(n) - LT_{min}^{dB}(n) \ .$$

In der linearen Domäne lautet die ensprechende Formel

$$SMR_{sb}(n) = \frac{L_{sb}(n)}{LT_{min}(n)} \ .$$

3.4.3 Das psycho-akustische Modell 2

Das zweite im Standard beschriebene psycho-akustische Modell ist als Grundlage für eine echtzeitfähige Implementation nicht geeignet. Die folgende Abschätzung soll diese Aussage näher begründen. Tabelle 3.2 gibt die geschätzte Anzahl an Befehlszyklen an, welche der TMS320C44 zur Berechnung verschiedener Operationen benötigt. Die Werte einfacher Funktionen, wie Division und Quadratwurzel, wurden aus den Beispielprogrammen des Kapitels 12 in [10] entnommen. Die Werte der trigonometrischen Funktionen wurden durch Zählung der notwendigen Operationen und Wichtung mit ihrer Zyklendauer in den entsprechenden Funktionsquelltexten der C-Standardbibliothek (`rts.src`) für den TMS320C44 ermittelt und stellen eine idealisierte Mindestanzahl an Zyklen dar.

Ausgangspunkt ist auch in diesem Modell ein Frequenzspektrum, welches für alle Schichten einheitlich mit Hilfe einer 1024-Punkt-FFT gewonnen wird. Bei Verwendung des Programmbeispiels 12-42 aus [10] werden etwa 19404 Befehlszyklen zur Berechnung der FFT gebraucht (nach Tabelle 12-2 in [10]). Für die weitere Berechnung sind die 512 relevanten komplexen

Tabelle 3.2 Idealisierte Befehlszyklenanzahl zur Berechnung verschiedener Operationen durch den TMS320C44

Operation	Befehlszyklen
Multiplikation	1
Division	9
Addition, Subtraktion	1
Quadratwurzel	12
`float`, `int`	1
`abs`, `fabs`	1
`atan2`	52
`sin`	20
`cos`	22

Ergebniswerte der *FFT* in die Polarkoordinatendarstellung zu überführen. Der Radius r und der Phasenwinkel φ berechnen sich mit

$$r = \sqrt{a^2 + b^2}$$

$$\varphi = \arctan \frac{b}{a} \; ,$$

wobei a der Realteil und b der Imaginärteil ist. Wichtet man die notwendigen Operationen mit den Werten der Tabelle 3.2, so werden für diese Berechnungen pro komplexer Zahl 67 Befehlszyklen benötigt.

Der nächste Schritt des Modells berechnet aus den Werten der zwei vorhergehenden *FFTs* einen Satz von Vorhersagewerten:

$$r^{pred} = 2 \cdot r(t-1) - r(t-2)$$

$$\varphi^{pred} = 2 \cdot \varphi(t-1) - \varphi(t-2) \; .$$

Diese Berechnungen benötigen 4 Befehlszyklen pro Wert, so daß sich für alle 512 zu berechnenden Werte 2048 weitere Zyklen ergeben.

Im folgenden Schritt sind Unvorhersagbarkeitswerte c nach der Formel

$$c = \frac{\sqrt{(a - r^{pred} \cos \varphi^{pred})^2 + (b - r^{pred} \sin \varphi^{pred})^2}}{r + |r^{pred}|}$$

zu berechnen. Pro Wert werden für die Berechnung dieser Formel mindestens 70 Befehlszyklen gebraucht.

Rechnet man die mindestens benötigten Befehlszyklen der aufgezählten Schritte zusammen, so ergeben sich

$$19404 + 512 \cdot (67 + 4 + 70) = 91596 \text{ Befehlszyklen.}$$

Die Befehlszykluszeit des TMS320C44 beträgt 40 ns bei einem Systemtakt von 50 MHz. Die Abarbeitung der errechneten Befehlszyklen dauert unter dieser Voraussetzung etwa 3.7 ms. Für die Schicht 1 entspricht diese Zeitdauer etwa 45 Prozent der für die Kodierung eines Rahmens zur Verfügung stehenden Zeit. Die abgeschätzte Rechenzeit schließt bereits eine Anwendung des Modells für alle zweikanaligen Modi (*stereo mode*, *joint stereo mode*, *dual channel mode*) in dieser Schicht aus, da die verbleibende Rechenzeit nicht für die übrigen Komponenten des *Encoders* ausreicht. Auch für die Schicht 2 ist die ermittelte Zeit bereits kritisch, da für diese Schicht eine zweimalige Berechnung pro Rahmen und Kanal erforderlich ist. Beachtet man weiterhin, daß die ermittelte Rechenzeit auf einer idealisierten Abschätzung beruht und nur ein Teil der Rechenschritte des Modells betrachtet wurden, so wäre bei einer konkreten Implementation eine wesentlich höhere Rechenzeit zu erwarten. Aufgrund dieser Einschätzung wurde das psycho-akustische Modell 2 als für eine echtzeitfähige Implementation ungeeignet angesehen.

3.4.4 Verwendung eines künstlich-neuronalen Netzes

Das psycho-akustische Modell kann als Abbildungsfunktion eines diskreten Frequenzspektrums auf einen Vektor mit 32 *SMR*-Werten betrachtet werden. *Multilayer-Perzeptrons* (*MLPs*), eine Klasse der künstlich-neuronalen Netze, sind prinzipiell geeignet, solche Abbildungen zu approximieren. Die Untersuchung auf Verwendbarkeit eines *MLPs* ist durch die folgenden potentiellen Vorteile motiviert:

1. eine einfache programmtechnische Realisierung,
2. die effiziente Berechnung durch den DSP,
3. eine konstante Rechenzeit.

Die folgenden Ausführungen setzten die Kenntnis über Aufbau und Funktionsweise von *MLPs* voraus. Hier sei auf die Standardliteratur zu den künstlich-neuronalen Netzen verwiesen. Das *Multilayer-Perzeptron* wird z.B. ausführlich in [6], Kapitel 5 vorgestellt.

Um die aufgezählten Vorteile näher zu begründen, soll zunächst die Propagierungsfunktion net_v für Neuronen des *MLPs* betrachtet werden. Diese lautet nach Paragraph v der Definition 5.1 aus [6]

$$net_v = \sum_{u \in U_{i-1}} W(u,v) \cdot o_u + \theta_v \ .$$

Im wesentlichen berechnet die Formel ein skalares Vektorprodukt zwischen einem Spaltenvektor der Gewichtsmatrix W und dem Vektor der Ausgabewerte o der vorhergehenden Schicht U_{i-1}. Mit der Multiplizierer-Addierer-Einheit des TMS320C44, sowie dessen *Hardware*-Unterstützung für Schleifen kann diese Berechnung sehr effizient implementiert werden. Das Beispiel 12-3 in [10] zeigt eine solche Implementation. Der das Skalarprodukt berechnende Programmkode besteht aus nur 5 Assemblerbefehlen! Die Berechnung benötigt bei einer idealen Speicheraufteilung $N + 5$ Befehlszyklen, wobei N die Länge der Vektoren ist.

Problematischer ist die Berechnung der Aktivierungsfunktion. Der Lernalgorithmus erfordert eine sigmoide, differenzierbare Aktivierungsfunktion, wie z.B. die logistische Funktion

$$f(x) = \frac{1}{1 + e^{-\beta x}}, \quad \beta > 0 \ .$$

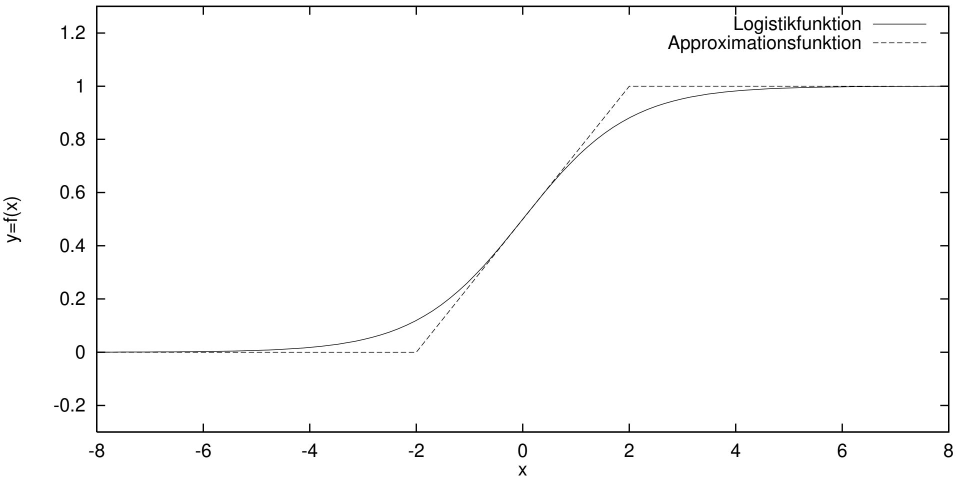
Die Berechnung dieser Funktion ist sehr aufwendig, so daß in der Anwendungsphase eine geeignete, einfach berechenbare Approximation einzusetzen ist. Hierfür könnte beispielsweise die stückweise lineare Funktion

$$f(x) = \begin{cases} 0 & \text{, wenn } x < -\frac{2}{\beta} \\ 1 & \text{, wenn } x > \frac{2}{\beta} \\ \frac{\beta}{4} \cdot x + 0.5 & \text{, sonst} \end{cases}$$

verwendet werden. Die Berechnung dieser Funktion benötigt nur fünf Befehlszyklen. Abbildung 3.6 zeigt den Verlauf der Logistikfunktion mit $\beta = 1$ und ihrer Approximationsfunktion.

Die bisherigen Ausführungen zeigen, daß ein *MLP* für den TMS320C44 einfach und effizient zu implementieren ist. Der Vorteil der konstanten Rechenzeit liegt in der Rechenvorschrift des *MLPs* begründet, welche keine Datenabhängigkeit enthält.

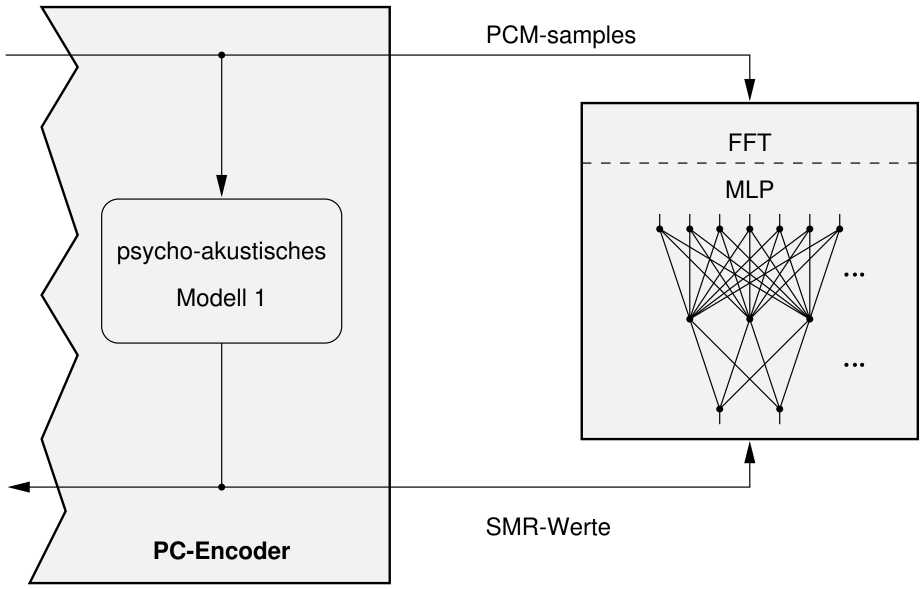
Abbildung 3.6 Logistikfunktion mit $\beta = 1$ und die entsprechende Approximationsfunktion



Untersucht wurde ein dreistufiges *MLP*, d.h. ein Netz mit einer verborgenen Schicht (*hidden layer*). Als Eingabe dient ein diskretes Frequenzspektrum, welches mit einer *FFT* berechnet wird. Da die Eingabewerte im Intervall $[0, 1]$ liegen müssen, werden die Spektralwerte so skaliert, daß der betragsmäßig größte Wert des Spektrums einer Eingabe von 1 entspricht. Die Ausgabewerte des *MLPs* liegen ebenfalls im Intervall $[0, 1]$, so daß auch sie in geeigneter Weise skaliert werden müssen.

Um das *MLP* auf die zu realisierende Abbildung anzulernen, wurde der *Backpropagation*-Algorithmus mit Momentumterm verwendet (siehe [6], Abschnitt 5.3). Das Netz wurde an das psycho-akustische Modell 1 eines bestehenden *Software-Encoders* parallel angekoppelt, wie dies in Abbildung 3.7 dargestellt ist. Während der *Encoder* Audiodaten kodiert, dienen die Eingabe- und Ergebniswerte des psycho-akustischen Modells als Lernmuster für das *MLP*. In [6], Abschnitt 5.3 wird erwähnt, daß ein Epochenlernen in der Regel die Lerngeschwindigkeit verbessert. Dabei werden dem Netz alle zu lernenden Muster präsentiert und die Gewichtsänderungen aufsummiert. Die Gewichte der Neuronen werden erst nach Ablauf einer Epoche, d.h. wenn alle Lernmuster dem Netz präsentiert wurden, mit den aufsummierten Gewichtsänderungen aktualisiert. Da die durch das psycho-akustische Modell generierten Lernmuster unbegrenzt sind, gibt es eigentlich keine Epoche.

Abbildung 3.7 Anlernen des *MLPs*



Um dennoch Epochenlernen verwenden zu können, wurde die Länge einer Epoche willkürlich festgelegt.

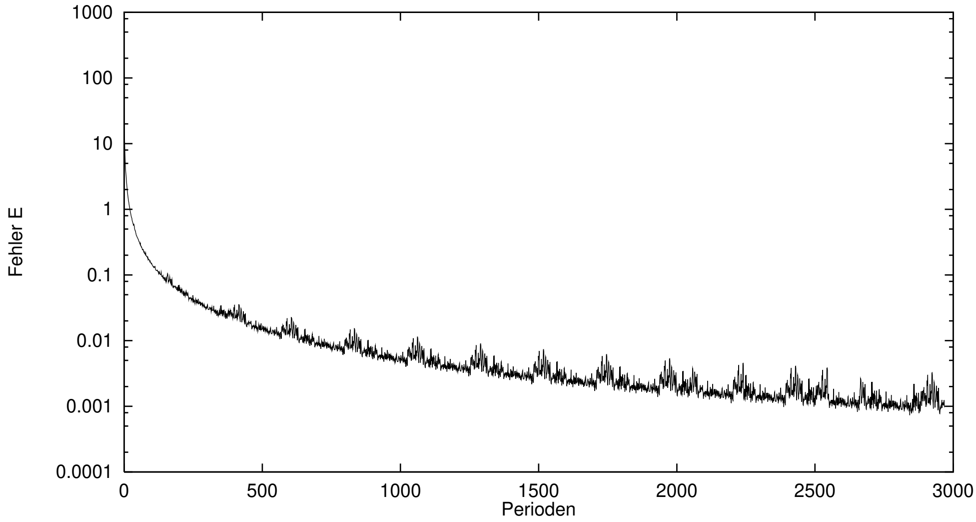
Für die Festlegung der Anzahl der Neuronen der versteckten Schicht gibt es keinen Richtwert. Wünschenswert ist eine möglichst geringe Anzahl, um den Rechenaufwand gering zu halten. Zu wenige versteckte Neuronen führen andererseits dazu, daß die gewünschte Abbildung nicht angelernt werden kann. Deshalb wurde mit verschiedenen Netzen experimentiert, wobei die Anzahl der versteckten Neuronen zwischen 64 und 256 lag.

Abbildung 3.8 zeigt den Fehlerverlauf beim Anlernen eines *MLP* mit 256 versteckten Neuronen. Der Fehler berechnet sich nach der Formel

$$E = \frac{1}{2} \sum_{p \in L} \sum_{v \in U_o} (t_v^{(p)} - a_v^{(p)})^2 \ .$$

Die Menge L beschreibt die Lernmuster p einer Epoche und die Menge U_o repräsentiert die Neuronen der Ausgabeschicht. Man erkennt, daß das Netz anfangs gut lernt, der Fehler dann jedoch anfängt zu schwanken. Während der erzielte Restfehler für viele Anwendungen künstlich-neuronaler Netze akzeptabel wäre, ist er für die Realisierung des psycho-akustischen Modells zu hoch. Die Ausgangswerte liegen in einem Bereich von -1.7 dB bis etwa 70 dB , was in der linearen Domäne einem Umfang von etwa 9 Zehnerpoten-

Abbildung 3.8 Fehlerverlauf beim Anlernen eines *MLPs* mit 256 versteckten Neuronen



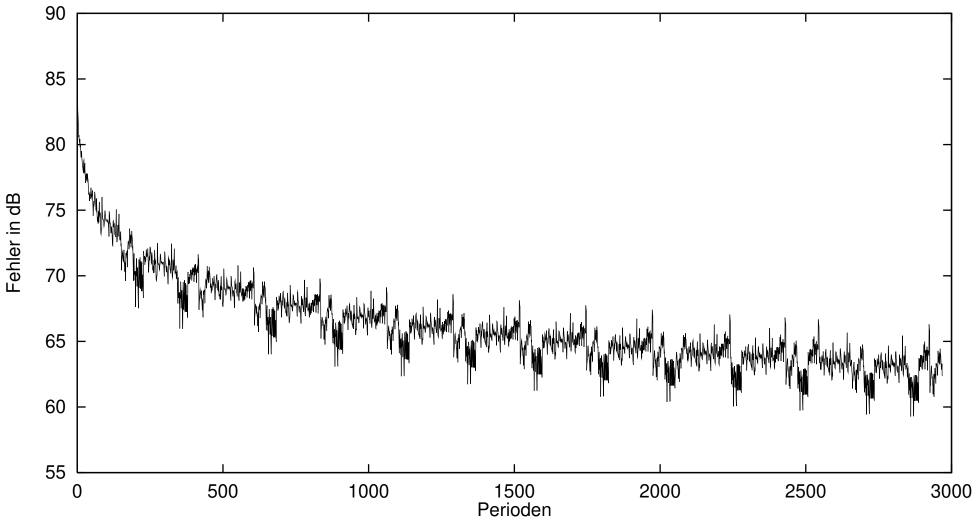
zen entspricht. Durch die für das Netz notwendige Skalierung in das Intervall $[0, 1]$ liegt der Wertebereich beim *MLP* zwischen 1 und etwa 10^{-9}. Nimmt man an, daß ein Restfehler von 10^{-3} erreichbar ist, so ist dieser immer noch 6 Zehnerpotenzen zu hoch. Noch deutlicher wird das Problem, wenn man die durchschnittliche Abweichung der Werte des Netzes von den Sollwerten als Verhältnis in Dezibel darstellt:

$$E^* = \frac{1}{N \cdot M} \cdot \sum_{p \in L} \sum_{v \in U_o} 10 \cdot |\lg \frac{t_v^{(p)}}{a_v^{(p)}}| \; .$$

N ist die Anzahl der Neuronen der Ausgangsschicht und M ist die Länge der Lernperiode. Diese Formel gibt eine von der Größe der Werte unabhängige Aussage über den Fehler des Netzes. Abbildung 3.9 zeigt die Entwicklung dieses Fehlers im Verlauf des Lernens.

Alle untersuchten *MLPs* zeigten ein ähnliches Lernverhalten. Neben der Anzahl der Neuronen der inneren Schicht, wurden auch Parameter, wie die Lernrate η und das Moment β, sowie die Periodenlänge variiert, ohne eine nennenswerte Verbesserung des Lernverhaltens zu erreichen. Wegen der hohen Ungenauigkeit des *MLPs* wurde die Realisierung des psycho-akustischen Modells mit einem solchen Netz letzlich verworfen.

Abbildung 3.9 Verlauf des durchschnittlichen Verhältnisfehlers des *MLP*



Kapitel 4

Aspekte der Implementation

4.1 Zielsystem und Enwicklungsumgebung

Die technische Grundlage für die zu entwickelnde *Software* bildet ein Rechnersystem mit dem digitalen Signal-Prozessor TMS320C44 von Texas Instruments. Das System ist als *PCI*-Einsteckkarte für einen PC realisiert. Der Systemtakt beträgt 50 MHz.

Der TMS320C44 ist ein Gleitkomma-Prozessor mit 11 Mehrzweckregistern. Ihre Breite beträgt 32 Bits für Festkommazahlen und 40 Bits für Gleitkommazahlen. Weiterhin sind 7 32-Bit-Adressregister sowie verschiedene Register für spezielle Aufgaben vorhanden. Der Prozessor verfügt über einen 128 Worte großen Befehls-*Cache*. Zwei Speicherblöcke mit einer Größe von jeweils 1024×32 Bits sind im Prozessor integriert. Jeder Speicherblock besitzt zwei *Ports*, wodurch in einem Befehlszyklus 2 parallele Zugriffe möglich sind. Weiterhin sind zwei Zähler/Zeitgeber und ein *DMA*-Koprozessor mit sechs Kanälen integriert. Die stark aufgefächerte *BUS*-Struktur innerhalb des Prozessors ermöglicht es, einen Befehlslesezugriff, zwei Datenzugriffe und einen *DMA*-Zugriff parallel auszuführen, sofern keine Speicherzugriffskonflikte bestehen. Zwei identische, programmierbare Speicherschnittstellen ermöglichen den Anschluß von externem Speicher und E/A-Bausteinen (*memory mapped io*). Eine ausführliche Beschreibung des TMS320C44 gibt [10].

Neben dem internen Speicher des Prozessors sind 128 KWorte statischer *RAM* verfügbar, welcher einen Zugriff ohne Wartezyklen ermöglicht. Ein Lesezugriff des Prozessors erfordert einen Befehlszyklus, während ein Schreibzugriff zwei Befehlszyklen benötigt.

Die *PCI*-Schnittstelle realisiert den Datenaustausch zwischen dem DSP-System und dem *Host*-PC. Weiterhin ist der Anschluß eines Wandlermoduls für die A/D- und D/A-Umsetzung geplant.

Zum Testen der *Software* kam das frei verfügbare Programm `gdb` (*GNU-debugger*) zum Einsatz. Die für die Unterstützung des TMS320C44 notwendige Ergänzung (*Patch*) ist auf dem *FTP-Server* `ongaonga.chch.cri.nz` verfügbar (Stand: Mai 1998). Diese Ergänzung beinhaltet auch einen Simulator, so daß eine vom Zielsystem unabhängige Programmentwicklung möglich ist. Der Simulator bietet eine sehr gute Unterstützung für das *Profiling* von Programmen. Er ermöglicht eine Analyse der Programmabarbeitung bis auf die Ebene der Prozessor-*Pipeline*.

4.2 Die Analysefilterbank

Die Analysefilterbank wird für die Kodierung eines Rahmens 12 Mal bei Schicht 1 bzw. 36 Mal bei Schicht 2 im *single channel mode* berechnet. Für alle anderen Modi verdoppeln sich diese Werte nochmals. Aufgrund dieser hohen Anzahl von Aufrufen hat ihre Rechenzeit großen Einfluß auf die gesamte Rechenzeit des *Encoders*. In diesem Abschnitt wird auf einige Aspekte der Implementation näher eingegangen. Der Algorithmus zur Berechnung der Filterbank ist auf Seite 26 wiedergegeben.

4.2.1 Organisation des Puffers X

Wie aus den ersten beiden Schritten zur Berechnung der Analysefilterbank ersichtlich ist, dient der Puffer X als *FIFO*-Speicher für die letzten 512 Abtastwerte. Um die Werteverschiebung im ersten Schritt zu vermeiden, ist es sinnvoll, diesen als Ring-Puffer auf einem linearen Feld zu implementieren. Der erste Schritt reduziert sich dabei auf die Aktualisierung des Startindex s nach der Formel

$$s = (s + 480) \bmod 512.$$

Der Nachteil dieser Implementation ist, daß für den Zugriff auf einen Wert mit dem Index i dessen Feldindex i^* zuerst berechnet werden muß. Diese Berechnung erfolgt nach der Formel

$$i^* = (s + i) \bmod 512.$$

Der TMS320C44 unterstützt durch einen speziellen Addressierungsmodus, den *circular addressing mode*, die Implementation von Ring-Puffern. In diesem Modus berechnet die Adreßrecheneinheit des Prozessors die notwendige Modulo-Operation. Der Speicherzugriff erfolgt hierbei genauso schnell wie in allen anderen Adressierungsarten. Der *circular addressing mode* ist ausführlich in [10], Abschnitt 5.3 beschrieben.

4.2.2 Implementationsvarianten der *iDCT*

Im Abschnitt 3.1 wurde bereits erläutert, wie sich die Rechenkomplexität der Matrix-Vektor-Multiplikation im Schritt 5 reduzieren läßt, indem man sie auf eine inverse diskrete Kosinus-Transformation (*iDCT*) zurückführt. Weiterhin wurde an gleicher Stelle ein Algorithmus zu deren schneller Berechnung vorgestellt. Im *Encoder* wurden verschiedene Varianten zur Berechnung der *iDCT* implementiert.

Für eine erste Implementation wurde die Matrix-Vektor-Multiplikation nach Formel 3.2 (Seite 25) verwendet. Der Vorteil dieser Variante liegt in der sehr einfachen Realisierung. Aufgrund der Multiplizierer-Addierer-Einheit des TMS320C44 und dessen *Hardware*-Unterstützung für Programmschleifen läßt sich die Matrix-Vektor-Multiplikation effizient berechnen. Trotz der genannten Vorteile ist die Rechenzeit dieser Variante aufgrund der hohen Rechenkomplexität sehr hoch. Ein weiterer Nachteil ist der relativ hohe Speicherbedarf für die Matrixkoeffizienten.

Im Abschnitt 3.1 wurde ein Algorithmus (Alg. 2, Seite 28) zur schnellen Berechnung der *iDCT* vorgestellt, welcher gegenüber der Matrix-Vektor-Multiplikation eine wesentlich kleinere Rechenkomplexität besitzt. Aufgrund seiner rekursiven Definition läßt er sich elegant als rekursive Funktion implementieren. Der Nachteil rekursiver Funktionen besteht in dem bei jedem Wiederaufruf auftretenden zeitlichen *Overhead*, sowie in der größeren Belastung des *Stack* mit Rückkehradressen und lokalen Daten. Diese Nachteile konnten aber bei der Implementation fast vollständig ausgeräumt werden. In Abbildung 4.1 ist zum besseren Verständnis der prinzipielle Aufbau der Funktion als C-Quelltext dargestellt. Es sei aber angemerkt, daß der im folgenden beschriebene rekursive Aufrufmechanismus nur in Assembler effizient zu implementieren ist.

Abbildung 4.1 Prinzipieller Ablauf der rekursiven Implementation der *iDCT*

```
void idct(register float* f, register unsigned int n)
{
  unsigned int ne = n;
  register unsigned int bitstack = 0;
  M1: /* rekursiver Eintrittspunkt */
  bitstack <<= 1; /* Retten des letzten Schalter-Bits */
  if (n == 2) {
     /* berechne 2-Punkt-iDCT */
      ...
  }
  else {
     /* Vektorzerlegung */
      ...
     n >>= 1; /* Neuberechnung von n für die Teilvektoren */
     goto M1; /* rekursiver Aufruf für den ersten Teilvektor */
     M2: /* Rueckkehradresse, erster Teilvektor bearbeitet */
     f += n; /* Startadresse für den zweiten Teilvektor */
     bitstack |= 1; /* Schalter-Bit setzen */
     goto M1; /* rekursiver Aufruf für den zweiten Teilvektor */
     M3: /* Rueckkehradresse, zweiter Teilvektor bearbeitet */
     f -= n; /* Wiederherstellen von f */
     n <<= 1; /* Wiederherstellen von n */
     /* Berechnung des Ergebnisvektors aus den Teilvektoren */
      ...
  }
  bitstack >>= 1; /* letztes Schalter-Bit wiederholen */
  if (n < ne) {
     if (bitstack & 1) /* Schalter-Bit gesetzt? */
        goto M3; /* zweiter Teilvektor bearbeitet */
     else
        goto M2; /* erster Teilvektor bearbeitet */
  }
}
```

Die Funktion kehrt, außer in der obersten Rekursionsebene, immer zu einer von zwei Adressen (Marke `M2` bzw. `M3`) zurück, da der Aufruf entweder für den ersten oder für den zweiten Teilvektor erfolgte. Aus diesem Grund ist es ausreichend, wenn man mit einem Schalter (*flag*) vermerkt, für welchen der beiden Teilvektoren der Aufruf erfolgte. Damit dieses Verfahren über mehrere Rekursionsebenen funktioniert, wurde ein Prozessorregister als Bit-*Stack* (Variable `bitstack`) verwendet. Das niederwertigste Bit des Registers dient als der beschriebene Schalter in der aktuellen Rekursionsebene. Bei Eintritt in eine neue Rekursionsebene wird der Registerinhalt um ein Bit nach links verschoben, so daß die Schalter-Bits der höheren Rekursionsebenen nicht überschrieben werden. Bei Verlassen der aktuellen Ebene wird diese Verschiebung rückgängig gemacht. Durch diesen Mechanismus ist das Retten von Rückkehradressen auf den *Stack* unnötig. Deshalb kann der Wiederaufruf und die Rückkehr mit normalen Sprungbefehlen erfolgen. In der Implementation wurden hierzu ausschließlich verzögerte Sprünge (*delayed branches*) verwendet, welche eine Ausführungszeit von nur einem Zyklus besitzen. Der sonst übliche Zeitaufwand für Sprünge bzw. Funktionsaufrufe reduziert sich deshalb auf ein geringes Maß. Eine Beschreibung der verzögerten Sprungbefehle des TMS320C44 findet der interessierte Leser in [10], Abschnitt 6.2. Der beschriebene Aufrufmechanismus kann auf ähnliche rekursive Algorithmen übertragen werden, wobei die maximale Rekursionstiefe durch die Breite des Bit-*Stack*-Registers begrenzt wird.

Als Argumente benötigt die Funktion die Länge `n` des Vektors, sowie dessen Startadresse `f`. Diese werden in Prozessorregistern übergeben. Vor jedem Wiederaufruf werden die Argumente neu berechnet. Auf das Sichern der alten Werte kann verzichtet werden, da man sie nach dem Funktionsaufruf zurückrechnen kann. Diese Berechnungen sind einfache Befehle, welche nicht aufwendiger sind als das Sichern und Wiederherstellen.

Der Vorteil der rekursiven Implementation besteht in dem geringen Verbrauch an Programmspeicher. Die Funktion paßt vollständig in den Programm-*Cache*, welcher auch durch die Wiederaufrufe genutzt wird.

Trotz der bereits effizienten rekursiven Implementation wurde als dritte Variante der Algorithmus nach *Lee* iterativ implementiert. Hierbei wurden die für die 32-Punkt-$iDCT$ notwendigen fünf Rekursionsebenen einzeln programmiert. Bei der rekursiven Variante werden die Teilvektoren einer Ebe-

Tabelle 4.1 Zeiten der verschiedenen Implementationen zur Berechnung der *iDCT*

Implementation	Zeit in μs	relative Zeit
Matrix-Vektor-Mult.	90	100 %
Lee-Algorithmus, rekursiv	42	47 %
Lee-Algorithmus, iterativ	29	32 %

ne in separaten Funktionsaufrufen bearbeitet. Im Gegensatz dazu werden gleiche Bearbeitungsschritte der Teilvektoren einer Ebene in der iterativen Variante innerhalb einer Schleife gemeinsam abgearbeitet, wodurch nur ein eimaliger Vorbereitungsaufwand entsteht.

Diese Variante braucht mehr Programmspeicher als die vorhergehenden, berechnet aber die *iDCT* am schnellsten.

In Tabelle 4.1 sind die Zeiten zur Berechnung der *iDCT* der verschiedenen Varianten gegenübergestellt. Nimmt man die Komplexität der Matrix-Vektor-Multiplikation der Formel 3.2 und die Rechenzeit ihrer Implementation jeweils als Bezugspunkt, so ist ein großer Unterschied zwischen der relativen Komplexität von 16 Prozent des *Lee*-Algorithmus (vergl. Tabelle 3.1) und der relativen Zeit von 32 Prozent der effizientesten Implementation zu bemerken. Diese Diskrepanz läßt sich einerseits durch die sehr effiziente Berechnung von Matrix-Vektor-Multiplikationen mit dem TMS320C44 erklären, andererseits erfordert der *Lee*-Algorithmus einen wesentlich höheren Verwaltungsaufwand.

4.3 Speicheraufteilung

Verschiedene parallele Befehle des TMS320C44, wie z.B. der Multiplizier-Addier-Befehl, erfordern in einem Zyklus zwei Speicherzugriffe zum Holen der Operanden. Weiterhin wird normalerweise in jedem Zyklus ein neues Befehlswort geholt, so daß sich eine maximale Anzahl von drei Speicherzugriffen pro Zyklus ergibt. Zusätzlich kann noch ein Speicherzugriff durch den *DMA*-Koprozessor auftreten. Wie bereits im Abschnitt 4.1 erwähnt, besitzt der TMS320C44 eine aufgefächerte interne *BUS*-Struktur, um alle diese Zugriffe parallel ausführen zu können. Einen Engpaß bilden aber die angesprochenen Speichereinheiten. Während der auf dem Prozessor integrierte Speicher zwei Zugriffe pro Zyklus zuläßt, kann auf den externen Speicher maximal einmal pro Zyklus zugegriffen werden. Treten mehr Speicherzugriffe auf, so entsteht ein Speicherzugriffskonflikt, welchen der Prozessor durch die Verteilung der Zugriffe über mehrere Zyklen auflöst. Die genauen Bedingungen, welche zu einem Zugriffskonflikt führen, sind in [10], Kap. 10 beschrieben.

Für eine effiziente Auslastung des Prozessors ist aus den genannten Gründen eine überlegte Anordnung der einzelnen Speichersegmente erforderlich. Diese Aussage läßt sich gut am Beispiel der *FFT* veranschaulichen. Die Anzahl der Befehlszyklen zur Bechnung einer 512-Punkt-*FFT* wurde für zwei unterschiedliche Speicherbelegungen ermittelt. Der Programmkode lag bei beiden Messungen im externen Speicher und der Befehls-*Cache* war aktiviert. Der Datenbereich (Segment `.fftbuf`) wurde im ersten Versuch im internen Speicher und im zweiten Versuch im externen Speicher angeordnet. Die erste Messung ergab 10076 Befehlszyklen. Der zweite Versuch benötigte 21704 Zyklen, womit für das gleiche Programm mehr als doppelt so viel Zeit gebraucht wurde. Für die Analysefilterbank ergab sich in einer ähnlichen Untersuchung ein Unterschied von 30 Prozent in der Rechenzeit.

Das Beispiel zeigt den großen Einfluß der Speicheranordnung auf die Rechenzeit. Datenbereiche, auf die häufig parallele Zugriffe erfolgen, wurden in eigenständige Segmente gelegt. Diese können dadurch beim *Linking* des Programms gezielt in die integrierten Speicherbereiche des Prozessors gelegt werden.

Weitere Segmente ergaben sich aus der Notwendigkeit, Datenbereiche an bestimmten Adressen auszurichten (*Alignment*). Die speziellen Adressierungsmodi *circular addressing* und *bit-reversed addressing* des TMS320C44 erfordern diese Ausrichtung (siehe [10], Abschnitte 5.3 und 5.4). Eine Beschreibung der einzelnen Segmente der *Encoder-Software* ist im Anhang B zu finden.

Kapitel 5

Leistungbewertung

Mit den in den beiden vorangegangenen Kapiteln erläuterten Algorithmen und Techniken wurde das Ziel verfolgt, die Rechenzeit für den Kodierprozeß zu reduzieren, um die bestehenden zeitlichen Restriktionen für die Echtzeitverarbeitung erfüllen zu können. In diesem Kapitel werden die Ergebnisse dieser Bemühungen ausgewertet. Neben den Rechenzeiten wird auch kurz auf die Qualität der kodierten Audiosignale eingegangen.

5.1 Meßmittel

Alle Zeitmessungen erfolgten mit einem im TMS320C44 integrierten Zeitgeber (siehe [10], Abschn. 9.13). Dieser Zeitgeber ermöglicht eine auf zwei Befehlszyklen genaue Messung. Alle Werte wurden mit dem im Abschnitt 4.1 beschriebenen Simulator erfaßt, welcher auch den Zeitgeber simuliert. Mit einer einmaligen vergleichenden Messung am realen System wurde die zeitliche Korrektheit des Simulators überprüft.

5.2 Definition der Echtzeitanforderung

Die Dauer des in einem Rahmen kodierten Abschnitts des Audiosignals entspricht dem Produkt aus Anzahl der kodierten Abtastwerte und Abtastfrequenz. Diese Dauer ist die obere Grenze für die zur Verarbeitung eines Rahmens zur Verfügung stehende Zeit. Tabelle 5.1 zeigt die für die verschiedenen Schichten und Abtastfrequenzen bestehenden Zeitgrenzen.

Neben der Rechenzeit des *Encoders* ist auch die Transferzeit der Eingangs-

Tabelle 5.1 Bestehende Zeitgrenzen

Schicht	Abtastwerte pro Rahmen	Abtastfrequenz in kHz		
		32.0	44.1	48.0
1	384	12.0 ms	8.707 ms	8.0 ms
2	1152	36.0 ms	26.123 ms	24.0 ms

und Ausgangsdaten zu berücksichtigen. Um eine Echtzeitverarbeitung gewährleisten zu können, darf die Summe dieser beiden Zeiten die bestehende Grenze nicht überschreiten.

5.3 Rechenzeiten des *Encoders*

5.3.1 Testbedingungen

Für alle in den folgenden Abschnitten erläuterten Messungen wurde die gleiche Speicheraufteilung verwendet, auf deren Einfluß bereits im Abschnitt 4.3 hingewiesen wurde. Im Anhang B ist die Bedeutung der einzelnen Speichersegmente näher erläutert. An gleicher Stelle ist auch die für die Messungen verwendete Speicheraufteilung abgedruckt.

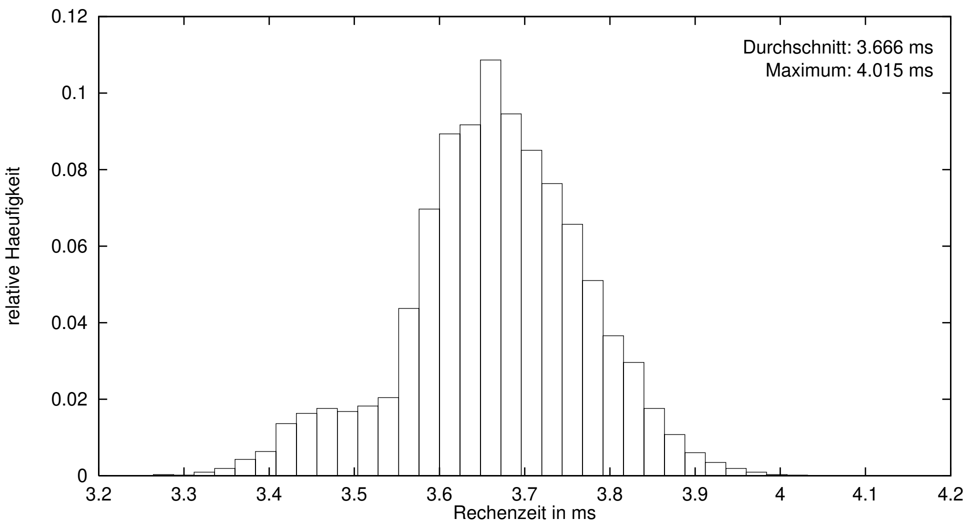
Es wurden nur die Zeiten für die Kodierung gemessen. Transferzeiten blieben unberücksichtigt, da diese von der jeweiligen Anwendung abhängen. Während der Messungen gab es keine *Interrupts* und keine konkurrierenden *DMA*-Transfers.

5.3.2 Die Schicht 1

Einige Komponenten des *Encoders*, wie z.B. das psycho-akustische Modell, arbeiten mit datenabhängigen Algorithmen. Aus diesem Grund ist auch die Kodierzeit eines Rahmens datenabhängig. Die genaue Bestimmung einer oberen Grenze für die Kodierzeit ist praktisch unmöglich. Um dennoch eine möglichst sichere Aussage über die Einhaltung der Echtzeitbedingungen zu machen, wurden Audiosignale unterschiedlichster Natur kodiert. Diese beinhalteten

- einen männlichen Sprecher
- Pop-Musik (weibliche Sängerin)

Abbildung 5.1 Häufigkeitsverteilung der Rahmenkodierzeiten für die Schicht 1 im *single channel mode*



- Chor und Orchester
- *Hard Rock*
- Rauschen
- ein synthetisches 4000 Hz Sinussignal.

Die Quelldaten lagen als 16-Bit-Monosignale mit einer Abtastfrequenz von 44.1 kHz vor. Für diese Signale wurden die Rahmenkodierzeiten erfaßt, wobei die Zielbitrate 96 kBits/s betrug. Aus den Meßwerten wurde eine Häufigkeitsverteilung der Kodierzeiten berechnet. Zu diesem Zweck wurde der Zeitbereich von 3 ms bis 4.2 ms in 50 Intervalle aufgeteilt und die Anzahl der in die einzelnen Intervalle fallenden Rahmenkodierzeiten gezählt. Diese Werte wurden schließlich mit der Gesamtanzahl der Meßwerte normiert. Die Häufigkeitsverteilung ist in Abbildung 5.1 dargestellt. Die maximal erreichte Kodierzeit betrug 4.015 ms. Das entspricht 46 % der zur Verfügung stehenden Zeit. Damit stehen noch 54 % der Zeit für den Transfer der Daten und anderen Aufgaben zur Verfügung.

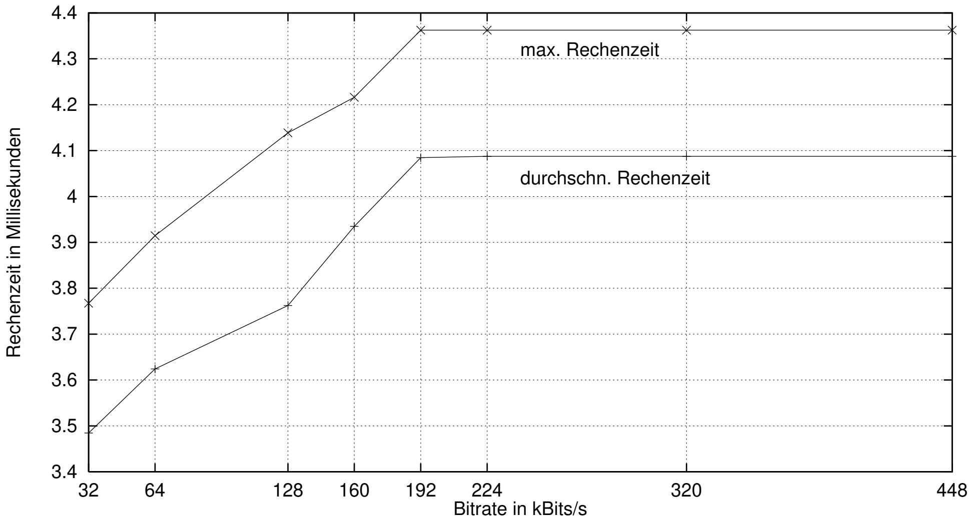
Die Rechenzeit des *Encoders* für die zweikanaligen Modi läßt sich nur abschätzen, da diese noch nicht vollständig implementiert sind. Im Abschnitt

2.8 ist erläutert worden, daß in diesen Modi einige Komponenten des *Encoders* für jeden Kanal getrennt durchlaufen werden. Dazu zählen auch die rechenintensivsten Komponenten, wie die Analysefilterbank und das psychoakustische Modell. Deshalb läßt sich mit großer Sicherheit abschätzen, daß die zweikanaligen Modi etwa die doppelte Laufzeit des *single channel mode* benötigen werden. Unter dieser Annahme beträgt die maximale Kodierzeit für diese Modi etwa 92 % der verfügbaren Zeit. Eine Echtzeitverarbeitung kann auch in diesen Modi gewährleistet werden, wenn die verbleibenden 8 % der Zeit für den Datentransfer ausreichend sind.

Einige Komponenten des *Encoders* sind in ihrer Laufzeit von der Zielbitrate abhängig. Im Abschnitt 3.3 wurde dieses Problem bereits anhand der Bit-Allokation deutlich. Für diese konnte allerdings ein Algorithmus gefunden werden, welcher nahezu keine Abhängigkeit der Laufzeit von der Zielbitrate aufweist. Auch der Quantisierer und die Formatierung hängen indirekt von der Zielbitrate ab. Aus diesem Grund wurde eine Audioquelle mehrmals für unterschiedliche Zielbitraten kodiert, um deren Einfluß auf die Kodierzeit zu untersuchen. Abbildung 5.2 zeigt die gemessenen Durchschnitts- und Maximalwerte der Rahmenkodierzeit in Abhängigkeit von der Bitrate. Man erkennt, daß die Rechenzeit ab 192 kBits/s nicht weiter ansteigt. Dieser Effekt kann damit erklärt werden, daß bei kleineren Bitraten aufgrund der beschränkten Anzahl verfügbarer Bits nicht für alle Frequenzbänder Bits allokiert werden können. Deshalb werden für diese Bänder auch keine *Subband samples* quantisiert und kodiert. Mit steigender Bitrate werden mehr Bänder kodiert, was zu einem Anstieg der zu quantisierenden und kodierenden *Sub-band samples* führt. Ab einer Bitrate von 192 kBits/s steigt die Anzahl der kodierten Bänder nicht mehr an, weshalb die Rechenzeit bei höheren Bitraten nahezu konstant bleibt.

Ab der Bitrate von 192 kBits/s beträgt die maximale Kodierzeit konstant 4.363 ms. Das sind 50.1 % der verfügbaren Zeit. Geht man von einer Verdopplung der Kodierzeit bei den zweikanaligen Modi aus, dann muß festgestellt werden, daß die Echtzeitbedingung verletzt wird. Der größte Durchschnittswert der Messungen betrug 4.085 ms, also etwa 47 % der verfügbaren Zeit. Die durchschnittliche Kodierzeit bei den zweikanaligen Modi kann damit auf 95 % geschätzt werden. Unter der Voraussetzung, daß die Daten in geeignetem Umfang gepuffert werden, sollte dennoch eine Echtzeitverarbeitung möglich sein. Eine endgültige Aussage kann erst nach der Implemen-

Abbildung 5.2 Durchschnittliche und maximale Rahmenkodierzeit in Abhängigkeit von der Bitrate bei Schicht 1 (*single channel mode*, 44.1 kHz Abtastfrequenz)



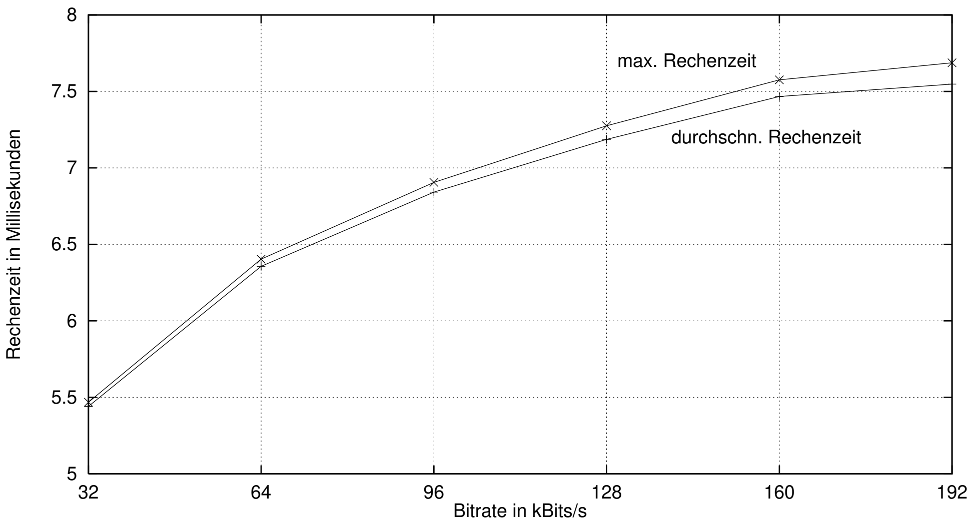
tation aller Modi getroffen werden.

Neben der Bitrate hat auch die Abtastfrequenz einen Einfluß auf die Rahmenkodierzeit. Ursache ist auch hier die Anzahl der verfügbaren Bits, welche mit abnehmender Abtastfrequenz zunimmt. Das führt aufgrund des oben beschriebenen Effekts zu einem Anstieg der Kodierzeit mit kleiner werdender Abtastfrequenz. Da sich aber gleichzeitig die zur Verfügung stehende Rechenzeit vergrößert (siehe Tab. 5.1), ist dieser Anstieg unkritisch. Auf eine Darstellung der Meßwerte wird aus diesem Grund verzichtet.

5.3.3 Die Schicht 2

Die im vorangegangenen Abschnitt erläuterten Messungen wurden in gleicher Weise für die Schicht 2 durchgeführt. Dabei ist anzumerken, daß das psycho-akustische Modell für diese Schicht derzeit nur in einer stark vereinfachten Form implementiert ist. Im Gegensatz zu der vollständigen Implementation ist die Laufzeit dieser Variante nicht datenabhängig. Die Meßergebnisse stellen aus diesem Grund vorläufige Werte dar, um eine Abschätzung über die Echtzeitfähigkeit machen zu können. Aus dem gleichen Grund wird auch auf die Abbildung einer Häufigkeitsverteilung der Rahmenkodierzeit

Abbildung 5.3 Durchschnittliche und maximale Rahmenkodierzeit in Abhängigkeit von der Bitrate bei Schicht 2 (*single channel mode*, 44.1 kHz Abtastfrequenz)



verzichtet.

Für die Ermittlung einer maximalen Kodierzeit wurden die gleichen Audioquellen wie bei der Schicht 1 verwendet. Die Zielbitrate war auch bei dieser Messung auf 96 kBits/s festgelegt. Der ermittelte Maximalwert der Rahmenkodierzeit betrug 6.905 ms. Bei einer Abtastfrequenz von 44.1 kHz entspricht dieser Wert 26.4 % der zur Verfügung stehenden Zeit. Man erkennt, daß die Einhaltung der Echtzeitbedingungen gegenüber der Schicht 1 viel unkritischer ist. Für die bei einer vollständigen Implementation des psycho-akustischen Modells etwas höher liegende Rechenzeit ist noch eine große Reserve vorhanden.

Auch für die Schicht 2 läßt sich derzeit die Rechenzeit bei den zweikanaligen Modi nur durch Verdopplung der Meßwerte abschätzen. Mit einer Kodierzeit von etwa 53 % der verfügbaren Zeit sind auch in diesen Modi genügend Zeitreserven vorhanden.

Interessant ist auch bei dieser Schicht der Einfluß der Bitrate auf die Kodierzeit. Abbildung 5.3 stellt diesen anhand der Durchschnitts- und Maximalwerte grafisch dar. Der bei der Bitrate von 192 kBits/s gemessene Maximalwert von 7.688 ms entspricht 29.4 % der verfügbaren Zeit. Höhere Bitraten sind bei der Schicht 2 im *single channel mode* nicht erlaubt.

5.4 Abschätzung der Datentransferzeit

Der Transfer der Eingangs- und Ausgangsdaten ist von der jeweiligen Anwendung abhängig. So können z.B. die Abtastwerte mit einem *DMA*-Transfer in Blöcken vom *Host-PC* übertragen werden. Eine andere Möglichkeit ist der direkte Anschluß eines A/D-Wandlers an das System, welcher *interrupt*-gesteuert die einzelnen Abtastwerte bereitstellt. Die zweite Möglichkeit wird wegen des häufigen Aufrufs der *Interrupt*-Funktion mehr Prozessorzeit erfordern.

Unabhängig von der Anwendung läßt sich abschätzen, wieviel Zeit bzw. Befehlszyklen im Durchschnitt für den Transfer eines Datums verbraucht werden können. Das soll am Beispiel der Werte aus dem Abschnitt 5.3.2 gezeigt werden. Dort wurde abgeschätzt, daß der *Encoder* bei einer Bitrate von 96 kBits/s für die zweikanaligen Modi etwa 92 % der verfügbaren Rechenzeit benötigt, womit maximal 8 % für den Datentransfer verbleiben. Bei einer Abtastfrequenz von 44.1 kHz sind das etwa 696 μs. In dieser Zeit müssen 384 neue Abtastwerte bereitgestellt werden. Weiterhin ist ein Rahmen zu transferieren. Die Länge des Rahmens beträgt bei 96 kBits/s Zielbitrate 27 Worte. Teilt man die verfügbare Zeit durch die Summe von Eingangs- und Ausgangsdaten, dann ergibt sich eine Zeit von 1.69 μs pro Datum. Diese Zeit entspricht 42 Befehlszyklen des Prozessors, welche im Durchschnitt für den Transfer eines Datums verbraucht werden können.

5.5 Qualität des kodierten Signals

Die qualitative Bewertung kodierter Audiosignale erfordert einen großen zeitlichen und technischen Aufwand. In [3] ist beschrieben, welche Methoden eingesetzt werden, um kodierte Signale in ihrer Qualität zu beurteilen. Unter anderem wird eine große Anzahl geschulter Testhörer sowie eine hochwertige Wiedergabetechnik eingesetzt. Im Gegensatz dazu sind die Ergebnisse dieses Abschnitts subjektive Eindrücke des Autors. Für die Wiedergabe der Audiosignale wurde eine Standard-*Sound*-Karte mit angeschlossenem Verstärker verwendet. Deshalb wurde lediglich abgeschätzt, ab welcher Bitrate die kodierten Signale keine größeren Störungen mehr aufwiesen. Ferner wurden die

Tabelle 5.2 Obere Grenzfrequenz des kodierten Audiosignals in Abhängigkeit von der Bitrate bei der Schicht 2 (44.1 kHz Abtastfrequenz)

Bitrate in kBits/s	obere Grenzfrequenz in kHz
≤ 48	5.5
56 ... 80	18.6
96 ... 192	20.7

Audioquellen mit einem alternativen *Encoder* kodiert und mit den Resultaten des DSP-*Encoders* verglichen. Die folgenden Aussagen beziehen sich auf den *single channel mode.*

Bei der Schicht 1 ist bei Bitraten bis 64 kBits/s eine deutlich hörbare Beeinträchtigung der Signalqualität zu bemerken. Erst ab 96 kBits/s wird eine akzeptable Qualität erreicht, wenn auch noch ein hörbarer Unterschied zum Originalsignal besteht. Dieser verschwindet erst ab etwa 128 kBits/s. Die mit dem alternativen *Encoder* kodierten Signale zeigten eine vergleichbare Charakteristik.

Bei der Schicht 2 wird die Bandbreite des kodierten Audiosignals, bedingt durch die Kodiertechnik, in Abhängigkeit von der Bitrate begrenzt. Dadurch wird bereits bei der Wahl der Bitrate eine erste Einschränkung der Signalqualität festgelegt. In Tabelle 5.2 ist für die verschiedenen Bitraten die obere Grenzfrequenz bei einer Abtastfrequenz von 44.1 kHz dargestellt.

Abgesehen von der Einschränkung der Bandbreite verschwinden starke kompressionsbedingte Störungen etwa ab einer Bitrate von 64 kBits/s. Auch hier ist der Höreindruck der kodierten Signale beider *Encoder* vergleichbar.

Kapitel 6

Zusammenfassung

Mit dieser Arbeit wurde gezeigt, daß die Implementierung eines echtzeitfähigen MPEG-1/Audio-*Encoders* auf dem TMS320C44 möglich ist. Es läßt sich feststellen, daß die Analyse und Optimierung von Algorithmen im Vorfeld einer Implementation einen großen Beitrag zur Reduzierung der Rechenkomplexität beitragen. Das Beispiel der Bit-Allokation zeigte aber auch, daß erst nach der Implementation ungünstige Eigenschaften eines Algorithmus anhand des Laufzeitverhaltens erkannt werden können. In diesem Beispiel konnte zumindest für die Schicht 1 ein besserer Algorithmus entworfen werden. Die Untersuchung des psycho-akustischen Modells 2 zeigte, daß bereits vor einer Implementation abgeschätzt werden kann, ob ein Algorithmus überhaupt in Echtzeit realisierbar ist. Es konnte auch festgestellt werden, daß eine Realisierung dieser Komponente mit einem künstlich-neuronalen Netz nicht möglich ist.

Neben geeigneten Algorithmen stellte die effiziente Implementierung den zweiten wichtigen Schwerpunkt dar. Dazu muß festgestellt werden, daß der TMS320C44 mit der gestellten Aufgabe an die Grenze seiner Leistungsfähigkeit stößt. Aus diesem Grund mußte die Prozessorarchitektur optimal ausgenutzt werden, weshalb das komplette Programm in Assembler geschrieben wurde. Eine leistungsfähigere Generation von Prozessoren und die weitere Entwicklung der *Compiler*-Technik erlauben hier vielleicht einmal den Einsatz einer höheren Programmiersprache, wobei von den optimierten Algorithmen dieser Arbeit in gleicher Weise profitiert werden kann.

Weitere Aufgaben sind die vollständige Implementierung der zweikanaligen

Modi und die Fertigstellung des psycho-akustischen Modells 1 für die Schicht 2. Weiterhin muß die *Encoder-Software* in das Betriebssystem des DSPs eingebunden werden, um sie nutzen zu können.

Anhang A

Liste der Quelltextdateien

Die folgende Aufzählung gibt eine Orientierung über den Inhalt der Quelltextdateien. Das Quellkode-Verzeichnis enthält zwei Unterverzeichnisse. Die Assembler-Quelltexte liegen im Unterverzeichnis `src`. Die zugehörigen *Include*-Dateien befinden sich im Unterverzeichnis `include`.

`anafilt.asm`: Analysefilterbank

`balloc.asm`: Bit-Allokation

`bstream.asm`: Bitstrom- und Rahmen-Erzeugung

`coder.asm`: Schnittstelle für `C`-Programme

`crc16.asm`: Prüfsummen-Generator

`dctmat.asm`: Koeffizientenmatrix M^*_{ik} nach Formel 3.2

`fft.asm`: Schnelle Fourier-Transformation (FFT), Beispielprogramm 12-42 aus [10]

`format.asm`: Formatierung des Rahmens

`idct32.asm`: rekursive Implementation der 32-Punkt-$iDCT$

`optmath.asm`: optimierte Funktionen für den dekadischen Logarithmus und seine Umkehrung 10^x

`pcmbuf.asm`: Verwaltung der Eingangspuffer für die Abtastwerte

`psy_one.asm`: psycho-akustisches Modell 1 `psytabs.asm`: Datentabellen für das psycho-akustische Modell 1

`quant.asm`: Quantisierer

`scalfac.asm`: Ermittlung der Skalierungsfaktoren

`tables.asm`: verschiedene Datentabellen

Anhang B

Speichersegmente

Im folgenden werden die in der *Encoder-Software* definierten Speichersegmente aufgelistet und ihre Bedeutung erläutert.

`.text` enthält den Programmkode.

`tables` enthält alle konstanten Datentabellen des *Encoders.*

`.fftbuf` speichert Ausgangs- und Ergebnisdaten der *FFT*. Das Segment muß an einer 2048-Wort-Grenze ausgerichtet sein, da der *bit-reversed addressing mode* verwendet wird (siehe [10], Abschn. 5.4). Es sollte im internen Speicher angeordnet werden, um eine hohe Einbuße an Rechenzeit (>100 %) durch Speicherzugriffskonflikte zu vermeiden.

`ana_xl` **und** `ana_xr` reservieren den Speicher für den Puffer X der Analysefilterbank jeweils für den linken und rechten Kanal (vgl. Alg. 1, S. 26). Im *single channel mode* wird nur `ana_xl` verwendet. Beide Segmente müssen an einer 1024-Wort-Grenze ausgerichtet sein wegen der Verwendung des *circular addressing modes* (siehe [10], Abschn. 5.3).

`.data` speichert die statischen Variablen des *Encoders.*

`ana_f` **und** `ana_tmp` speichern die Daten bei der Berechnung der *iDCT* innerhalb der Analysefilterbank. Eine Anordnung dieser Segmente in den internen Speicher beschleunigt die Berechnung der Analysefilterbank um ca. 30 Prozent gegenüber der Anordnung im externen Speicher. Bei Verwendung der schnellen Berechnungsvariante der *iDCT* (Makrovariable `.IDCT=2`) muß `ana_f` an einer 64-Wort-Grenze ausgerichtet sein, da diese Variante den *bit-reversed addressing mode* benutzt.

Für die im Kapitel 5 beschriebenen Messungen wurde die folgende Speicheraufteilung verwendet:

`.fftbuf`	interner Speicher - Block 0
`ana_xl`	externer Speicher
`.text`	externer Speicher
`ana_xr`	externer Speicher
`tables`	externer Speicher
`ana_f`	interner Speicher - Block 1
`.data`	interner Speicher - Block 1
`ana_tmp`	interner Speicher - Block 1
`.stack`	externer Speicher

Anhang C

Schnittstelle für C-Programme

Das `src`-Unterverzeichnis der *Encoder-Software* enthält ein `Makefile`, mit dessen Hilfe die Bibliothek `coder.lib` erstellt wird. Diese Bibliothek kann in C-Programme eingebunden werden (*linking*). Die C-Schnittstelle ist in der Datei `coder.h` deklariert, welche sich im Unterverzeichnis `include` befindet.

Die Initialisierung des *Encoders* erfolgt mit der Funktion `init_coder`, welche wie folgt deklariert ist:

```
int init_coder(unsigned int layer,
               unsigned int fs,
               unsigned int bitrate,
               unsigned int mode,
               unsigned int mext,
               unsigned int prot,
               unsigned int *pcmbuf,
               unsigned int blocks);
```

Das Argument `layer` legt die Schicht (1 oder 2) fest. Die Abtastfrequenz in Hertz wird im Argument `fs` übergegeben. Die Festlegung der Zielbitrate erfolgt im Argument `bitrate`. Die verwendete Einheit ist kBits/s. Der Modus wird durch das Argument `mode` festgelegt. Es gilt folgende Zuordnung:

Modus	`mode`
stereo mode	0
joint stereo mode	1
dual channel mode	2
single channel (mono) mode	3

Im *joint stereo mode* werden mit dem Argument `mext` die zusammengelegten Frequenzbänder festgelegt. Die gültigen Werte sind im Standard [4], S.22 festgelegt. Für alle anderen Modi ist dieses Argument bedeutungslos. Das Argument `prot` schaltet die Generierung der Prüfsumme aus (0) oder ein (1). Der Zeiger `pcmbuf` übergibt die Adresse eines Speicherbereiches, in dem die eingehenden Abtastwerte gespeichert werden. Die Größe des Puffers muß ein ganzzahliges Vielfaches der Anzahl von Abtastwerten pro Rahmen betragen, also $n \cdot 384$ bei Schicht 1 und $n \cdot 1152$ bei Schicht 2 mit $n > 1$. Die Anzahl n wird in dem Argument `blocks` übergeben. Die Funktion liefert bei erfolgreicher Initialisierung einen Wert ungleich Null zurück.

Die kontinuierliche Kodierung eines eingehenden *Sample*-Stroms erfolgt durch wiederholten Aufruf der parameterlosen Funktion `code_frame`. Pro Aufruf wird dabei ein Rahmen kodiert.

Für den Datenaustausch wird ein *Call-back*-Mechanismus verwendet. Das Programm, welches die *Encoder-Software* benutzt, muß die folgenden Funktionen implementieren:

```
extern void get_samples(unsigned int *buf, unsigned int len);
extern unsigned int *next_fbuf(unsigned int len);
```

Die Funktion `get_samples` wird durch den *Encoder* aufgerufen, wenn neue Abtastwerte benötigt werden. Der zu füllende Speicherbereich wird durch den Zeiger `buf` adressiert. Die Anzahl der benötigten Werte wird in `len` übergeben. Der *Encoder* unterstützt 16-Bit-*Samples*. Im *single channel mode* werden die Werte in den unteren 16 Bits der Speicherworte des Puffers erwartet. In den anderen Modi liegen die Werte des einen (linken) Kanals in den unteren 16 Bits und die Werte des zweiten (rechten) Kanals in den oberen 16 Bits der Speicherworte.

Durch Aufruf der Funktion `next_fbuf` fordert der *Encoder* neuen Speicher für einen Rahmen an. Das Argument `len` spezifiziert die Länge des angeforderten Speicherbereichs. Die Funktion muß einen Zeiger auf diesen Bereich zurückgeben. Der Aufruf dieser Funktion bedeutet gleichzeitig die

Rückgabe des durch den vorhergehenden Aufruf angeforderten Speichers. Dieser enthält den aktuellen Teil des kodierten Bitstroms und kann durch die Anwendung weiterverarbeitet werden.

Literaturverzeichnis

[1] Bertuch, Manfred: *Kino inklusive.* c't, (10):80–81, 1998.

[2] *brockhaus abc elektronik.* Brockhaus Verlag Leipzig, 1978.

[3] Deutsche Telekom AG, Technologiezentrum, Am Kavalleriesand 3, 64276 Darmstadt: *Highlights aus der Forschung - Quellencodierung und Qualitätsbeurteilung als Impulsgeber für Innovationen bei Sprache, Audio und Video*, 1996.

[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 - Part 3: Audio*, 1. Auflage, 1993.

[5] Lochmann, Dietmar: *Digitale Nachrichtentechnik*, Band 1. VEB Verlag Technik Berlin, 1990.

[6] Nauck, Detlef, Frank Klawonn und Rudolf Kruse: *Neuronale Netze und Fuzzy-Systeme.* Vieweg Verlag, 2. Auflage, 1996.

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

[8] 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`.

[9] P. Yip, K.R. Rao: *Discrete Cosine Transform - Algorithms, Advantages, Applications.* ACADEMIC PRESS, INC, 1990.

[10] *TMS320C4x User's Guide*, 1993.

Erklärung

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

Dresden, den 17. September 1998 Kai Stuhlemmer