

Analysis of the MPEG-1 Layer III (MP3) Algorithm Using MATLAB

Synthesis Lectures on Algorithms and Software in Engineering

Editor

Andreas Spanias, *Arizona State University*

[Analysis of the MPEG-1 Layer III \(MP3\) Algorithm Using MATLAB](#)

Jayaraman J. Thiagarajan and Andreas Spanias

2011

[Theory and Applications of Gaussian Quadrature Methods](#)

Narayan Kovvali

2011

[Algorithms and Software for Predictive and Perceptual Modeling of Speech](#)

Venkatraman Atti

2011

[Adaptive High-Resolution Sensor Waveform Design for Tracking](#)

Ioannis Kyriakides, Darryl Morrell, and Antonia Papandreou-Suppappola

2010

[MATLAB® Software for the Code Excited Linear Prediction Algorithm: The Federal Standard-1016](#)

Karthikeyan N. Ramamurthy and Andreas S. Spanias

2010

[OFDM Systems for Wireless Communications](#)

Adarsh B. Narasimhamurthy, Mahesh K. Banavar, and Cihan Tepedelenliouglu

2010

[Advances in Modern Blind Signal Separation Algorithms: Theory and Applications](#)

Kostas Kokkinakis and Philipos C. Loizou

2010

Advances in Waveform-Agile Sensing for Tracking

Sandeep Prasad Sira, Antonia Papandreou-Suppappola, and Darryl Morrell
2008

Despeckle Filtering Algorithms and Software for Ultrasound Imaging

Christos P. Loizou and Constantinos S. Pattichis
2008

Copyright © 2012 by Morgan & Claypool

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopy, recording, or any other except for brief quotations in printed reviews, without the prior permission of the publisher.

Analysis of the MPEG-1 Layer III (MP3) Algorithm Using MATLAB

Jayaraman J. Thiagarajan and Andreas Spanias

www.morganclaypool.com

ISBN: 9781608458011 paperback

ISBN: 9781608458028 ebook

DOI 10.2200/S00382ED1V01Y201110ASE009

A Publication in the Morgan & Claypool Publishers series

SYNTHESIS LECTURES ON ALGORITHMS AND SOFTWARE IN ENGINEERING

Lecture #9

Series Editor: Andreas Spanias, *Arizona State University*

Series ISSN

Synthesis Lectures on Algorithms and Software in Engineering

Print 1938-1727 Electronic 1938-1735

Although the authors believe that the concepts, algorithms, software, and data presented are accurate, they provide no warranty or implied warranty that they are free of error. No warranty or implied warranty is given that any of the book content and software is fit for a particular purpose, application or product. Theory, algorithms and software must not be used without extensive verification.

Analysis of the MPEG-1 Layer III (MP3) Algorithm Using MATLAB

Jayaraman J. Thiagarajan and Andreas Spanias
Arizona State University

*SYNTHESIS LECTURES ON ALGORITHMS AND SOFTWARE IN
ENGINEERING #9*



MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

The MPEG-1 Layer III (MP3) algorithm is one of the most successful audio The MPEG-1 Layer III (MP3) algorithm is one of the most successful audio formats for consumer audio storage and for transfer and playback of music on digital audio players. The MP3 compression standard along with the AAC (Advanced Audio Coding) algorithm are associated with the most successful music players of the last decade. This book describes the fundamentals and the MATLAB implementation details of the MP3 algorithm. Several of the tedious processes in MP3 are supported by demonstrations using MATLAB® software. The book presents the theoretical concepts and algorithms used in the MP3 standard. The implementation details and simulations with MATLAB® complement the theoretical principles. The extensive list of references enables the reader to perform a more detailed study on specific aspects of the algorithm and gain exposure to advancements in perceptual coding.

KEYWORDS

perceptual audio coders, lossy coding, MPEG standards, psychoacoustic models, MP3 algorithm, variable-bit-rate coding

Contents

	Preface	xi
	Acknowledgments	xiii
1	Introduction	1
1.1	A Brief History of Audio Coders	1
1.1.1	Recent Audio Codecs	2
1.2	A General Perceptual Audio Coding Architecture	4
1.3	Principles of Psychoacoustics	5
1.3.1	Psychoacoustic Model I	8
1.4	The MPEG-1 Layer III Algorithm	9
1.4.1	Analysis Subband Filterbank	9
1.4.2	MDCT and Hybrid Filter Bank	10
1.4.3	Psychoacoustic Analysis	10
1.4.4	Bit Allocation	10
1.5	Summary	11
2	Analysis Subband Filter Bank	13
2.1	Description	13
2.2	Characteristics of the Analysis Filter Bank	13
2.3	Filter Bank Implementation	15
2.3.1	Framing	18
2.3.2	Build Input Sample Vector	19
2.3.3	Window Vector x	20
2.3.4	Calculation of the Cosine Modulation Matrix	20
2.3.5	Calculation of the Subband Samples	21
2.4	Demonstration with Test Data	22
2.5	Summary	22
3	Psychoacoustic Model II	25
3.1	Description	25
3.2	Illustration of the Psychoacoustic Model II with MATLAB Code	25

	3.2.1 Analysis	25
	3.2.2 Computation of Energy and Unpredictability in Threshold Partitions ...	28
	3.2.3 Spreading Function and Excitation Pattern	31
	3.2.4 Tonality Index Estimation	35
	3.2.5 Calculation of Masking Thresholds for Threshold Partitions	36
	3.2.6 Pre-Echo Detection and Window Switching	40
	3.2.7 Estimation of SMR	43
	3.3 Summary	46
4	MDCT	47
	4.1 Description	47
	4.2 Illustration of the MDCT with MATLAB Code	47
	4.2.1 MDCT Calculation	47
	4.2.2 Alias Reduction	51
	4.3 Summary	53
5	Bit Allocation, Quantization and Coding	57
	5.1 Description	57
	5.2 The Loops Frame Function	58
	5.2.1 Calculation of the Scalefactor Selection Information (scfsi)	58
	5.3 Distortion Control (Outer Loop)	61
	5.4 Inner Loop (Rate Control)	65
	5.4.1 Non-Uniform Quantization	65
	5.4.2 Huffman Coding	66
	5.5 Bitstream Formatting	73
	5.5.1 Bit Reservoir	74
	5.6 Summary	80
6	Decoder	81
	6.1 Synchronization and Error Checking	81
	6.2 Decoding Side Information	83
	6.2.1 Extracting Parameters for Huffman Decoding	83
	6.2.2 Extracting Parameters for Scalefactor Decoding	84
	6.3 Scalefactor Decoding	86
	6.4 Huffman Decoding	88
	6.5 Requantization	90
	6.6 Reordering	91

6.7	Alias Reconstruction	93
6.8	The Inverse Modified Discrete Cosine Transform	93
6.9	Polyphase Synthesis Filterbank	94
6.10	Summary	97
A	Complexity Profile of the MPEG-1 Layer III Algorithm	101
	Bibliography	105
	Authors' Biographies	115

Preface

The MPEG-1 audio Layer-3, popularly referred to as the MP3, is a digital encoding format based on perceptual audio compression. This audio-specific format, designed by the Moving Pictures Expert Group, was standardized as a part of the generic MPEG-1 standard in the early 1990's. MP3 has found widespread application in consumer audio storage and in recording and playback of music on digital audio players. The MP3 compression standard along with the AAC (Advanced Audio Coding) standard is associated with perhaps the most successful music players of the last decade.

This book presents MATLAB[®] software that implements several important functions of the MPEG-1 Layer 3 encoding and decoding algorithms. We describe the fundamentals and implementation details of the algorithm along with several MATLAB[®] demonstrations. In the First chapter, we present a brief discussion on the history of audio coders and describe the architectural overview of perceptual coders. Furthermore, we describe the principles of psychoacoustics and provide details of the *psychoacoustic model I* used in the earlier MPEG audio coders. From Chapter 2 and on, we discuss the various modules of the MP3 algorithm. The theory sections in most chapters provide description of the necessary concepts required to understand the algorithm. The implementation details and simulations with MATLAB[®] functions complement these descriptions. The extensive list of references will enable the reader to perform a more detailed study on specific aspects of the algorithm and gain exposure to some of the recent advancements in perceptual coding. Finally, a detailed analysis of the computational complexity of the MP3 algorithm, for both the C and MATLAB implementations, is presented

Audio coders, in addition to being an interesting application of signal processing principles for students, present a valuable resource to both practitioners and researchers in several other sound related implementations. In particular, the functions and software provided in this book will enable practitioners and algorithm developers to understand and optimize/modify sections of the algorithm in order to achieve improved performance and design computationally efficient implementations. Furthermore, understanding the principles of the MP3 algorithm will enable the reader to understand and analyze several of the more recent perceptual coders. Students will be able to visualize and understand the various modules of MP3 by using the software and the associated simulations.

Jayaraman J. Thiagarajan and Andreas Spanias
October 2011

Acknowledgments

The authors acknowledge Ramapriya Rangachar, a Masters graduate student that worked in our lab, who was involved in developing some of the MATLAB® code for the MP3 algorithm, and ISO for developing the C code for the algorithm. We note that the MATLAB® code presented in this book is organized in modules by the authors in order to facilitate the learning of the MP3 functions. In fact, several functions have been developed directly from the original standard documentation. The authors also like to thank Ted Painter and Venkatraman Atti for providing descriptions of some of the foundations of psychoacoustics and audio coding. In addition, the first author acknowledges Karthikeyan N. Ramamurthy, Harini Sridhar and Vivek Iyer for their support. Portions of this work and associated software have been supported by the SenSIP Center and NSF grants 0817596, 1035086.

Jayaraman J. Thiagarajan and Andreas Spanias
October 2011

CHAPTER 1

Introduction

Audio coding or *audio compression* algorithms obtain reduced bit rate representations of high-fidelity audio signals and have applications in transmission, storage, streaming and broadcasting. The objective of audio coding algorithms is to represent the signal with a small number of bits while maintaining its perceptual quality such that it is indistinguishable from the original. The compact disk (CD) introduced true high-fidelity at high data rates. Conventional digital audio signals are associated with sampling frequencies of either 44.1 (CD) or 48 kHz (DAT) and the samples are encoded with pulse code modulation (PCM) at 16-bits per sample. This results in very high data rates amounting to 1.41 Mbits/s for a stereo-pair sampled at 44.1 kHz. Motivated by the need for compression algorithms for network and portable applications several codecs have been established. The focus of this book is to present a detailed analysis of several aspects of the MPEG-1 Layer-III audio coding standard. To facilitate the understanding of MP3, the theoretical concepts discussed in the following chapters are accompanied by various simulation examples. Furthermore, this book emphasizes the implementation specifics of the MP3 codec by including MATLAB code snippets and a detailed complexity analysis of the encoder and decoder functions.

1.1 A BRIEF HISTORY OF AUDIO CODERS

Digital audio compression relies on sophisticated time-frequency analysis techniques that use transform coders, filterbanks or hybrid signal-adaptive coding techniques. Audio coding relies heavily on exploiting properties of psychoacoustics [1, 2, 3, 4]. Foundations of audio coding using filter banks have been established both in the time-domain [5, 6, 7, 8, 9, 10, 11, 12, 13] and the frequency domain [14, 15]. Furthermore, Discrete Wavelet Transform (DWT) based subband coders [16, 17, 18] have also been considered for audio coding due to the flexibility in the choice of filter coefficients. The characterization of the auditory filterbank in terms of critical bands [2] has been used in audio compression and over the years several filterbanks [19, 20, 21, 22, 23] have been proposed to mimic the critical band structure. Furthermore, several quantization strategies have also been applied to transform-coders that use the discrete cosine transform (DCT) [24] and the modified DCT (MDCT) [25].

During the early nineties, several workgroups and organizations such as the ISO/IEC and the International Telecommunications Union (ITU) became actively involved in developing perceptual audio coding standards. The Masking pattern adapted Universal Subband Integrated Coding and Multiplexing (MUSICAM) algorithm [20, 26] had an initial influence on the ISO/IEC (International Organization for Standardization/International Electro-technical Commission) MPEG

2 1. INTRODUCTION

(Moving Pictures Experts Group) audio standards, i.e., MPEG-1 [27] and the MPEG-2 [28]. Furthermore, several successful commercial audio standards have been published including Sony's Adaptive TRansform Acoustic Coding (ATRAC), DTS Coherent Acoustics (DTS-CA) and Dolby's Audio Coder-3 (AC-3). Elements or entire algorithms for perceptual coding have also appeared in [21, 23], [27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 41, 42, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53]. With the emergence of surround sound systems, multi-channel encoding formats also gained interest [54, 55, 56]. The advent of ISO/IEC MPEG-4 standardization [45, 47] established new research goals for high-quality coding of general audio signals even at low bit rates. MPEG-4 audio encompasses an integrated family of algorithms with wide ranging provisions for scalable, object-based speech and audio coding at bit rates from 200 bps up to 64 kbps per channel [57, 58].

1.1.1 RECENT AUDIO CODECS

The older MPEG-1 hybrid audio coding technique (ISO/IEC 11172-3) incorporates subband filter bank decomposition, signal transforms such as the FFT and psychoacoustic analysis. MPEG-1 audio operates on 16-bit PCM input audio data and accommodates sample rates of 32, 44.1, and 48 kHz. Operating modes of this algorithm include mono, stereo, dual independent mono, and joint stereo. The target bit rates are programmable in the range of 32-192 kbits/s for mono and 64-384 kbits/s for stereo. Despite the fact that MPEG-1 Layer-III (MP3) is still an active and popular standard, several new algorithms have been shown to perform better. Advanced Audio Coding (AAC) is a standardized, lossy compression scheme that generally achieves better sound quality than MP3 at similar bit rates. It has been standardized by the ISO and IEC as part of the MPEG-2 and MPEG-4 standards. Designed as a successor to the MP3 algorithm, AAC allows more sampling frequencies (8 kHz to 96 kHz) and supports up to 48 channels.

Though perceptual audio coders such as the MP3 and AAC offer reasonably good quality at bit rates down to 80 kbps, they are associated with an algorithmic delay that exceeds 120 ms. Applications such as two-way communications or broadcasting require low end-to-end delays of the order of 20 ms. As a result, Low Delay (LD) audio coding schemes have been developed and they provide comparable perceptual quality to MP3 or AAC with a very low algorithmic delay. The MPEG-4 AAC audio coder is used as a basis to build the low delay functionality preferable in end-to-end applications such as teleconferencing and telephony. Typical bit rates of AAC-LD start at 32 kbps for a mono signal with 22 kHz sampling rate and reach 128 kbps providing excellent audio quality [59]. AAC-ELD (Enhanced Low Delay) was standardized as part of MPEG in January 2008. AAC-ELD has an algorithmic delay of 32 ms at 24 kbps down to 15 ms at 64 kbps. AAC-ELD combines the advantages of AAC-LD for low encoding/decoding purposes and Spectral Band Replication (SBR) for preserving high quality at low bit rates. Delay critical applications such as wideband audio/video conferencing, broadcasting which require high quality audio at low bit rates can benefit from this scheme [60]. The Ultra Low Delay (ULD) AAC [61] was developed at Fraunhofer and attains delays of the order of 8 ms.

The need for an interface to exchange multimedia content through the internet resulted in the development of the MPEG-7 audio standard [48]. MPEG-7 supports a broad range of applications [62] that include the following: multimedia indexing/searching, multimedia editing, broadcast media selection, and multimedia digital library sorting. Issues such as the “interoperability” and multimedia resource delivery over a wide range of networks and terminals motivated the MPEG-21 Framework [53].

As mentioned earlier, Adaptive Transform Acoustic Coding (ATRAC) is a family of audio compression algorithms developed by Sony. Though the initial versions of ATRAC were used with the MiniDisc in the early 1990s, today the recent advanced ATRAC algorithms are used in several Sony-branded audio players, the Real Audio 8 and the native audio compression format for audio rendering in PS3 [63]. The MPEG-4 parametric audio codec, called Harmonic and Individual Lines plus Noise (HILN), enables coding of general audio signals at bitrates as low as 4 kbit/s using a parametric representation [64]. The encoder assumes that the audio signals can be synthesized using only sinusoids and noise. The input signal is decomposed into components based on appropriate source models and represented by model parameters. This approach utilizes more advanced source modeling than just assuming a stationary signal for the duration of a frame.

The launch of storage formats (in 1999) such as the DVD-Audio and the Super Audio CD (SACD) provided the audio codec designers with enormous storage capacity. This motivated an effort for *lossless* coding of digital audio [46, 51, 65]. A lossless audio coding system is able to reconstruct perfectly a “bit-for-bit representation” of the original input audio from the coded bitstream. In contrast, a coding scheme incapable of perfect reconstruction from the coded representation is called *lossy*. Several commercially successful lossless codecs have been developed in the last decade. Some of the earliest lossless audio coders include the Apple Lossless Audio Codec (ALAC) [66] and the Windows Media Audio 9 (WMA 9) lossless codec [13]. ALAC is an audio codec developed by Apple Inc. for lossless data compression of digital music. Typically, it is stored within an MP4 container with the filename extension .m4a. Though this extension is also used by AAC, ALAC employs linear prediction similar to other lossless codecs. All current iPod and iPhone devices can play Apple Lossless-encoded files. The WMA 9 lossless codec was released by Microsoft in early 2003 and it supports up to 96 kHz, 24-bit, 5.1 discrete channels with full dynamic range compression control. It can compress this multichannel signal audio CD at bit rates of 470 to 940 kbit/s.

Dolby TrueHD is an advanced lossless multi-channel audio codec developed by Dolby Laboratories [67]. It is primarily intended for high-definition home-entertainment equipment such as the Blu-ray Disc and the HD DVD. Though Dolby TrueHD is based on Meridian Lossless Packing (MLP) [46], it is significantly different from DVD-Audio. This variable bit-rate codec can support up to 14 discrete sound channels in its bitstream. Another important audio codec is DTS-HD Master Audio, developed by Digital Theater System [11]. It is an optional audio format for the Blu-ray Disc format exclusively. This format aims to allow a bit-to-bit representation of the original movie’s studio master soundtrack. To accomplish this, DTS-HD MA supports variable bit rates up to 24.5 Mbit/s on a Blu-ray Disc and up to 18.0 Mbit/s for HD DVD. The DTS-HD Master Audio

4 1. INTRODUCTION

contains 2 data streams: the original DTS core stream and the residual stream which contains the difference between the original signal and the lossy compression DTS core stream [68]. The residual data is then encoded by a lossless encoder and packed together with the core. The most recent version of the Real player also supports lossless coding. The RealAudio lossless codec is designed primarily for high-quality music downloads in mono or two-channel stereo format (multichannel output is not supported). It replicates CD-quality sound in a format that takes less time for the user to download. Although the lossless audio codec is designed for high-fidelity music downloads, it can also be used for broadcasts in high-bandwidth environments.

The MPEG-4 Audio Lossless Coding, also referred as MPEG-4 ALS [12], extends the MPEG-4 Part 3 audio standard to perform lossless audio compression. It comprises of a short-term predictor, which is a quantized LPC predictor with a lossless residual, and a long term predictor modeled by 5 long-term weighted residues, each with its own delay. The long term predictor improves the compression for sounds with rich harmonics found in several musical instruments and human voice.

1.2 A GENERAL PERCEPTUAL AUDIO CODING ARCHITECTURE

It is important to note the architectural similarities that characterize most perceptual audio coders before we describe the MP3 audio codec in the following chapters. Over the last few years, researchers have proposed several efficient signal models and compression standards/methodologies for high-quality digital audio reproduction. Most of these algorithms are based on the generic architecture shown in Figure 1.1. Most coders typically segment input signals into quasi-stationary frames ranging from 2 to 50 ms in duration. This is followed by a time-frequency analysis to estimate the temporal and spectral components of each frame. Often, the time-frequency mapping is matched to the analysis properties of the human auditory system, although this is not always the case. The objective is to extract a set of time-frequency parameters that can be efficiently coded based on perceptual criteria. The time-frequency analysis module can typically comprise of time-invariant or time-varying filterbanks, harmonic analyzers and hybrid transforms.

The choice of time-frequency analysis methodology always involves a fundamental tradeoff between time and frequency resolution requirements. The time-frequency analysis module employed in the MPEG-1 codec is described in Chapter 2 and the strategies to handle the different resolution requirements are presented in Chapter 4. Perceptual distortion control is achieved by a psychoacoustic signal analysis module that estimates signal masking power based on psychoacoustic principles. The psychoacoustic model quantifies the maximum amount of distortion at each point in the time-frequency plane such that quantization of the time-frequency parameters does not introduce audible artifacts. The steps involved in the estimation of the masking thresholds using the psychoacoustic model – II are explained in Section 3.2 of this book. The quantization and encoding module can also exploit statistical redundancies through classical techniques such as DPCM or ADPCM. The redundancies in the quantized parameters can be removed using run-length and entropy coding

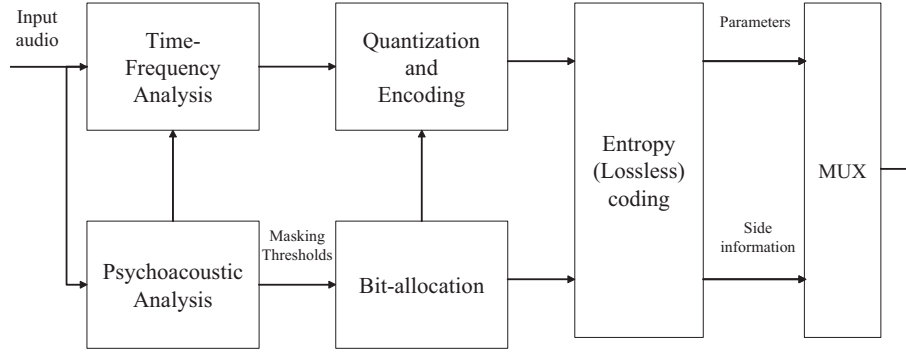


Figure 1.1: A generic block diagram of a perceptual audio encoder.

strategies [69, 70, 71]. Since the psychoacoustic module is signal dependent, most audio coding algorithms are variable rate. However, fixed channel rates can be achieved by efficient management of bit allocation using buffer feedback schemes. The coding methodology and the bit management techniques employed in the MP3 algorithm to achieve a fixed average bit rate are discussed in Sections 5.4 and 5.5, respectively.

1.3 PRINCIPLES OF PSYCHOACOUSTICS

Audio coding algorithms rely on generalized models of human hearing to optimize coding efficiency. The receiver is ultimately the human ear, and sound perception is affected by its masking properties. The field of psychoacoustics has made significant progress toward characterizing the time-frequency analysis capabilities of the inner ear. This, in turn, enabled audio coders to achieve compression by exploiting “irrelevant” information that is not detectable by even a trained listener. Irrelevant information is identified by incorporating psychoacoustic principles in quantization rules, including critical band frequency analysis and masking. The psychoacoustic model described in Chapter 3 relies on the principles discussed in this section. The *Sound Pressure Level* (SPL) is a standard metric that quantifies the intensity of an acoustic stimulus. The SPL provides the level (intensity) of sound pressure in decibels (dB) relative to an internationally defined reference level, i.e., $L_{SPL} = 20 \log_{10}(p/p_0)$, where L_{SPL} is the SPL of a stimulus, p is the sound pressure of the stimulus in Pascals (Pa - equivalent to Newton/m^2), and p_0 is the standard reference level of $20 \mu\text{Pa}$. Loosely speaking, about 150 dB SPL spans the dynamic range of the auditory system; an SPL reference of a quiet environment is around 0 dB SPL while a stimulus of 140 dB SPL approaches the threshold of pain. The absolute threshold of hearing shown in Figure 1.2 characterizes the amount of energy needed in a pure tone such that it can be detected by a listener in a noiseless environment.

The curve for the absolute threshold of hearing alone cannot be used for audio coding. Typically, music records require spectrally complex quantization rules and hence one has to modify the

6 1. INTRODUCTION

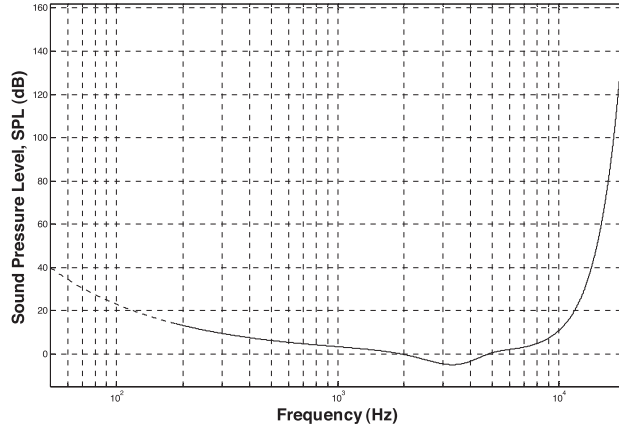


Figure 1.2: The absolute threshold of hearing in a noiseless environment.

absolute threshold in a dynamic manner. In order to estimate a time-varying threshold, one must use models for human hearing that take into account how the human ear performs spectral analysis. A frequency-to-place transformation takes place in the cochlea (inner ear), along the basilar membrane [72]. The loudness (perceived intensity) remains constant for a narrowband noise source presented at a constant SPL even as the noise bandwidth is increased up to the critical bandwidth. For any increase beyond the critical bandwidth, the loudness begins to increase. Critical bandwidth tends to remain constant (about 100 Hz) up to 500 Hz, and increases to approximately 20% of the center frequency above 500 Hz. The width of a critical band is commonly referred to as one *Bark*. The nonlinear function,

$$H_z(f) = 1.3 \arctan(0.00076f) + 3.5 \arctan\left[\left(\frac{f}{7500}\right)^2\right] \text{ (Bark)} \quad (1.1)$$

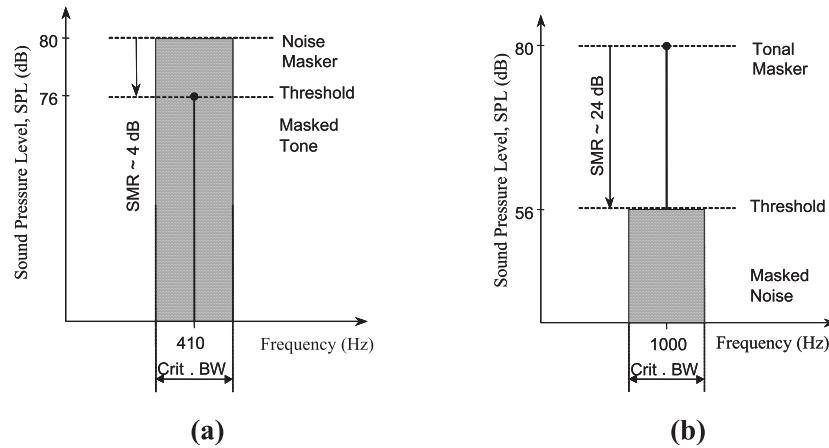
is often used to convert frequency from the Hertz to the Bark scale. Table 1.1 shows the idealized critical band filter bank in terms of band edges and center frequencies for a collection of 26 critical bandwidth auditory filters that span the audio spectrum. The frequency resolution of the auditory filter bank largely determines which portions of a signal are perceptually irrelevant. The auditory time-frequency analysis that occurs in the critical band filter bank induces simultaneous and non-simultaneous masking phenomena that are routinely used by modern audio coders to shape the coding distortion spectrum. As we will discuss in Section 5.1, the perceptual models allocate bits for signal components such that the quantization noise is shaped to exploit the *masking* thresholds for a complex sound.

Masking refers to a process where one sound is rendered inaudible because of the presence of another sound. In audio coding typically we distinguish between only three types of simultaneous

Table 1.1: Band Edges of 26 Critical Bandwidth Auditory Filters that Span the Audio Spectrum.

Band No.	Bandwidth (Hz)	Band No.	Bandwidth (Hz)
1	- 100	14	2000-2320
2	100-200	15	2320-2700
3	200-300	16	2700-3150
4	300-400	17	3150-3700
5	400-510	18	3700-4400
6	510-630	19	4400-5300
7	630-770	20	5300-6400
8	770-920	21	6400-7700
9	920-1080	22	7700-9500
10	1080-1270	23	9500-12000
11	1270-1480	24	12000-15500
12	1480-1720	25	15500-25000
13	1720-2000	26	25000-

masking, namely, *noise-masking-tone* (NMT), *tone-masking-noise* (TMN), and *noise-masking-noise* (NMN). In the NMT scenario shown in Figure 1.3 (a), a narrowband noise with 1 Bark bandwidth masks a tone within the same critical band, provided that the intensity of the masked tone is below a predictable threshold directly related to the intensity, and to a lesser extent, center frequency of the

**Figure 1.3:** (a) Noise masking tone, and (b) tone masking noise [72].

8 1. INTRODUCTION

masking noise. In the same figure, a critical band noise masker centered at 410 Hz with an intensity of 80 dB SPL masks a 410 Hz tone. In a tone-masking-noise scenario, Figure 1.3 (b), a 1 kHz pure tone at 80 dB SPL masks a critical band narrowband noise centered at 1 kHz of overall intensity 56 dB SPL. When comparing Figure 1.3 (a) with Figure 1.3 (b), it is important to notice that noise maskers have greater masking power than tonal maskers.

The simultaneous masking effects characterized above by the simplified paradigms of NMT and TMN are not bandlimited to within the boundaries of a single critical band. The effect of a masker centered within one critical band on the psychoacoustic computations in other critical bands is typically modeled by an approximately triangular spreading function that has slopes of +25 dB and -10 dB per bark (Figure 1.4). The global masking threshold or the level of “Just Noticeable Distortion” (JND) is an estimate of level at which the quantization noise becomes perceptible. The standard practice in perceptual coding involves first classifying masking signals as either noise or tone, computing appropriate thresholds, and using this information to shape the noise spectrum beneath the JND level.

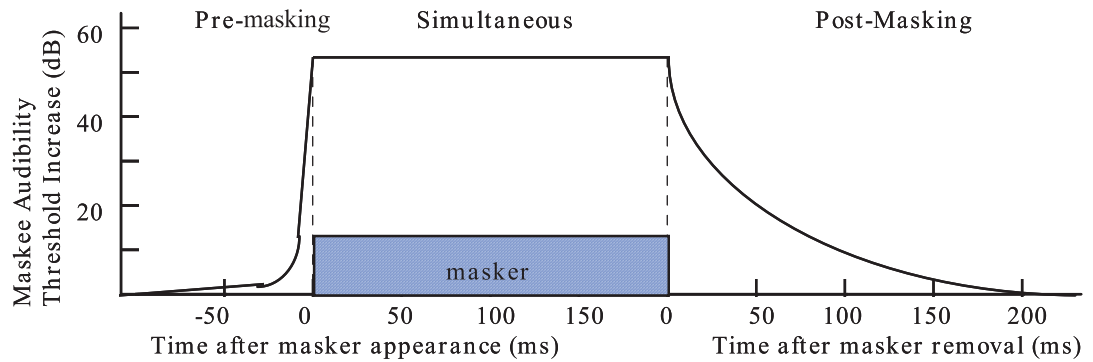


Figure 1.4: Non-simultaneous masking properties of the human ear [72].

1.3.1 PSYCHOACOUSTIC MODEL I

In this section, we briefly discuss the psychoacoustic model I used in MPEG-I audio coders. However, it is important to note that the MP3 algorithm employs a modified version of this model (psychoacoustic model II) and it will be presented in detail in Chapter 3. Researchers in audio coding have combined the notions of psychoacoustic masking with signal quantization principles to define *perceptual entropy* (PE), a measure of perceptually relevant information contained in any audio record. PE is expressed in bits per sample and represents a theoretical limit on the compressibility of a particular signal. PE measurements suggest that a wide variety of CD-quality audio source material can be transparently compressed to approximately 2.1 bits per sample. In psychoacoustics model I, the PE estimation process is accomplished as follows. The signal is windowed (Hanning)

and transformed to the frequency domain by a 2048-point Fast Fourier Transform (FFT). Masking thresholds are obtained by performing critical band analysis (with spreading), determining the noise-like or tone-like nature of the signal, applying threshold rules for the signal quality, and then accounting for the absolute hearing threshold. Finally, a determination of the number of bits required to quantize the spectrum without injecting perceptible noise is made. The PE measurement is obtained by constructing a PE histogram over many frames and then choosing a worst-case value as the actual measurement. The details of this psychoacoustic model can be found in [72]. As we will describe in Section 3.2, the ideas for performing the PE computation are embedded in the MP3 codec.

1.4 THE MPEG-1 LAYER III ALGORITHM

The architecture of the MP3 encoder, shown in Figure 1.5, operates on frames that consist of 1152 audio samples; each frame is split into two 576-sample subframes called *granules*. At the decoder,

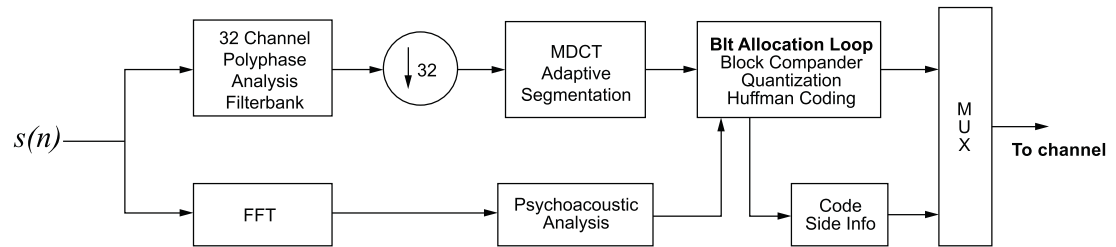


Figure 1.5: The ISO/MPEG-1 Layer III (MP3) encoder.

every granule can be decoded independently. A hybrid filter bank is used to approximate the critical band model for human hearing. This filter bank includes adaptive segmentation and consists of subband filters followed by the Modified Discrete Cosine Transform (MDCT). The filter bank and MDCT perform time-frequency analysis with adaptive resolution. The filter bank consists of thirty-two channels with fixed bandwidth followed by an MDCT. The MDCT and its software implementation can be found in Chapter 4. The MDCT operates on an adaptive frame length to allow low and high resolution spectral analysis. Sophisticated bit allocation strategies that rely upon non-uniform quantization and entropy coding are used to reduce the overall bit rate and “pack” an audio data record efficiently in a small file. Bit allocation and quantization of the spectral MDCT components are realized in a nested loop procedure that uses both non-uniform quantization and Huffman coding.

1.4.1 ANALYSIS SUBBAND FILTERBANK

As an alternative to tree-structured QMF filter banks, cosine modulation of a lowpass prototype filter is used in the MP3 algorithm to realize the parallel M -channel filter bank. Because these

10 1. INTRODUCTION

filters do not achieve perfect reconstruction, they are known as pseudo-QMF (PQMF) and are characterized by several attractive properties, i.e., linear phase channel response, constant group delay, low complexity, and fast block realization. In the PQMF bank formulation, phase distortion is eliminated by forcing the k -th analysis and synthesis filters to satisfy the mirror image condition on their L -th order impulse responses. Moreover, adjacent channel aliasing is cancelled by establishing precise relationships between the analysis and synthesis filters.

1.4.2 MDCT AND HYBRID FILTER BANK

The filter bank outputs are processed using the MDCT. The data is segmented and processed with 50% overlapping blocks. In Layer III, there are two possible block sizes for the MDCT, namely, a short block (12 samples) and a long block (36 samples). The MDCT module employs short blocks (better time resolution) for quick transients and long blocks (better frequency resolution) for slowly varying signals. The selection of long and short blocks, based on the perceptual entropy measure, is described in Section 4.2. To avoid rapid transitions, intermediate long-to-short and short-to-long transition windows are provided in the standard. For a particular block of data, all the filter bank channels can have the same MDCT block-type (short or long) or a mixed mode, where the two lower frequency subbands have long blocks while the remaining 30 upper bands have short blocks. The mixed mode provides better frequency resolution for the lower frequencies while maintaining a high time resolution for the higher frequencies.

1.4.3 PSYCHOACOUSTIC ANALYSIS

As described earlier, the signal analysis modes of the hybrid filter bank and the subsequent quantization rules are determined based on the principles of psychoacoustics. The MP3 psychoacoustic model II determines the maximum allowable quantization noise energy in each critical band such that quantization noise remains inaudible. In one of its modes, the model uses a 512-point FFT for high resolution spectral analysis (86.13 Hz resolution), then estimates individual simultaneous masking thresholds for each input frame due to the presence of tone-like and noise-like maskers in the signal spectrum. A global masking threshold is then estimated for a subset of the original 256 FFT bins by an additive combination of the tonal and individual non-tonal masking thresholds.

1.4.4 BIT ALLOCATION

The bit allocation process uses the parameters estimated in the psychoacoustic module to determine the number of bits to be allocated to each of the subbands. In the MP3 algorithm, a nested loop procedure is implemented which adjusts the quantizer step sizes until the number of bits required to encode the transform components falls within the available bit budget. This is achieved by evaluating the quality of the encoded signal in terms of quantization noise relative to the masked thresholds. Although this audio coding algorithm supports a fixed rate, the bit rates required for each frame is in general time varying. The number of bits provided to each frame is determined within the iteration loop. In Section 5.5.1, we describe a short-term buffer technique, referred to as the bit reservoir, that

enhances the coding efficiency by storing surplus bits during periods of low demand and allocating them during periods of high demand. This results in a time-varying instantaneous bit rate but a fixed average bit rate.

1.5 SUMMARY

This chapter provided a review on the history of audio coding algorithms and introduced the general architecture of perceptual audio coders. In addition, we presented a brief discussion on the basic psychoacoustic principles and the implementation of psychoacoustic model-I used in MPEG coders. A more detailed review of the modern audio coding algorithms can be found in the book by Spanias et al. [72]. From Chapter 2 and on, we will present the details of the MPEG-1 Layer III algorithm and provide relevant MATLAB program for all the pertinent functions. Chapter 2 will describe the design of the analysis subband filterbank and analyze the characteristics in terms of its relevance to the human auditory system. The estimation of the masking thresholds and JND using the psychoacoustic model II are discussed in detail in Chapter 3. The principles of the modified DCT and the application of different windows to achieve time-varying temporal and spectral resolutions are detailed in Chapter 4. Chapter 5 illustrates the bit allocation process, quantization and coding of the spectral lines, and the final chapter presents the design of the MP3 decoder for a single channel case.

Analysis Subband Filter Bank

2.1 DESCRIPTION

The MPEG-1 audio codec uses a time-frequency analysis block to extract parameters from the time-domain input signal that are suitable for quantization and encoding based on perceptual criteria. One of the most commonly used tools for time-frequency mapping is a filter bank, which divides the signal spectrum into multiple frequency subbands. In the MPEG-1 audio encoder, the input audio stream is passed through a filter bank that divides the audio input into 32 uniformly spaced frequency subbands. The time-frequency mapping provided by the filter bank is amenable to perceptual noise shaping at the encoder [73]. Furthermore, decomposition of the signal into its constituent frequency components reduces the redundancy in the signal.

The analysis subband filter bank consists of a polyphase structure and is common to all layers of the MPEG-1 audio compression [72]. The filter bank is based on a cosine-modulated lowpass prototype filter and consists of M channels with uniform bandwidth ($M=32$). This pseudo-QMF (PQMF) filter bank achieves nearly perfect reconstruction [74, 75]. These filter banks can be easily understood using an analysis-synthesis framework as shown in Figure 2.1, which illustrates a uniform 32-band critically sampled analysis-synthesis filter bank. The analysis and synthesis filters are designed suitably to reduce aliasing and imaging distortions.

2.2 CHARACTERISTICS OF THE ANALYSIS FILTER BANK

Efficient coding performance of a perceptual coder depends on the match between the properties of the designed filter bank and the characteristics of the input signal [76]. The filter bank used in the MP3 encoder is critically sampled or maximally decimated, i.e., the number of subband samples is same as the number of input samples. For every 32 input samples, the filter bank produces 32 output samples. In effect, each of the 32 subband filters subsamples its output by a factor of 32. The filters provide good time resolution with a reasonable frequency resolution. A cosine modulated pseudo-QMF M -band filter bank has a nearly perfect reconstruction and has uniform, linear phase channel responses [77]. The prototype impulse response contains 512 samples, with sidelobe suppression better than 96-dB in the stop-band and the output ripple is less than 0.07 dB [27].

Though the analysis subband filterbank has very good characteristics, the lack of sharp cutoff frequencies in the subband filters can cause the spectral component of a tone to leak into an adjacent subband. Since it is impossible to achieve perfectly separated magnitude responses with bandpass filters, there is unavoidable aliasing between the decimated subband samples. As seen in Figure 2.5, there is a significant overlap in the magnitude responses of adjacent bands. To illustrate this effect,

14 2. ANALYSIS SUBBAND FILTER BANK

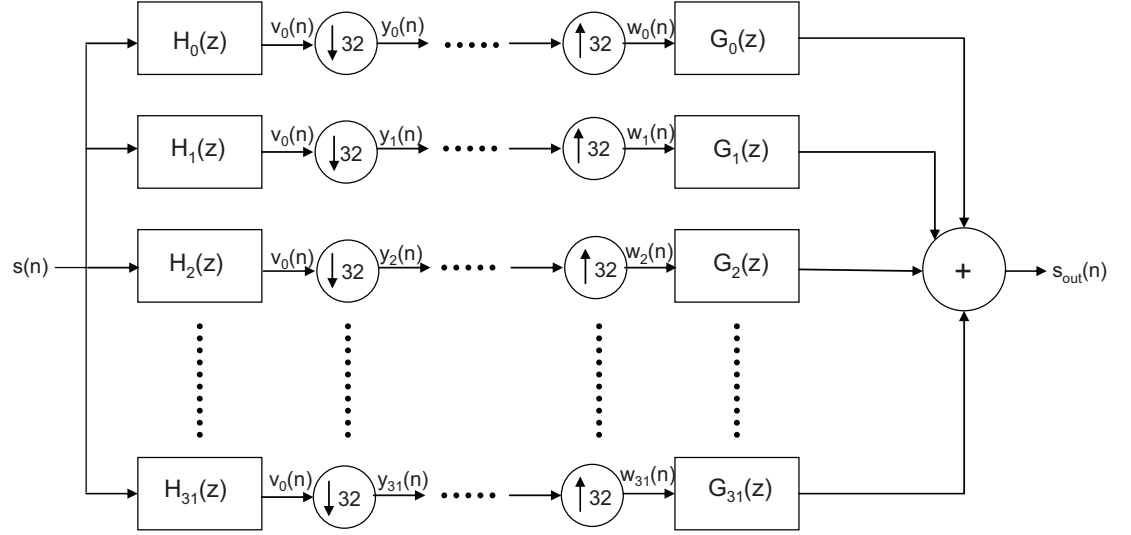


Figure 2.1: A uniform 32-band critically sampled analysis-synthesis filter bank.

consider a single tone of 1500 Hz, provided as an input to this filterbank. The bandwidth of each subband filter is around 689 Hz. Therefore, ideally, the spectral component of the tone should be seen only in subband 3. But due to the overlapping, the tone leaks into the adjacent band (subband 2) also. Figure 2.2 illustrates the input tone and the responses of the two subbands. Furthermore, the subsampling step also introduces aliasing. Hence, alias reduction is performed in the MDCT module, which will be described in Chapter 4.

The other important consideration with this analysis filter bank design is that, the equal widths of the subbands do not reflect the human auditory system's frequency dependent behavior. The width of a *critical band* as a function of frequency is a better indicator of the human auditory behavior [73]. Many psychoacoustic effects are consistent in a critical-band frequency scale. At lower frequencies, a single subband covers several critical bands and in this circumstance the number of quantization bits cannot be allocated for the individual critical bands. Filter banks that emulate the properties of the human auditory system have proven to be highly efficient in coding transient signals. However, such filter banks have been less effective on harmonically structured signals because of their low coding gain compared to filter banks with a large number of subbands. Furthermore, the filter bank and its inverse do not comprise a lossless transformation. Even in the absence of quantization, the inverse transform cannot recover the original signal perfectly. However, by efficient design the error introduced by the filter bank can be made small and inaudible.

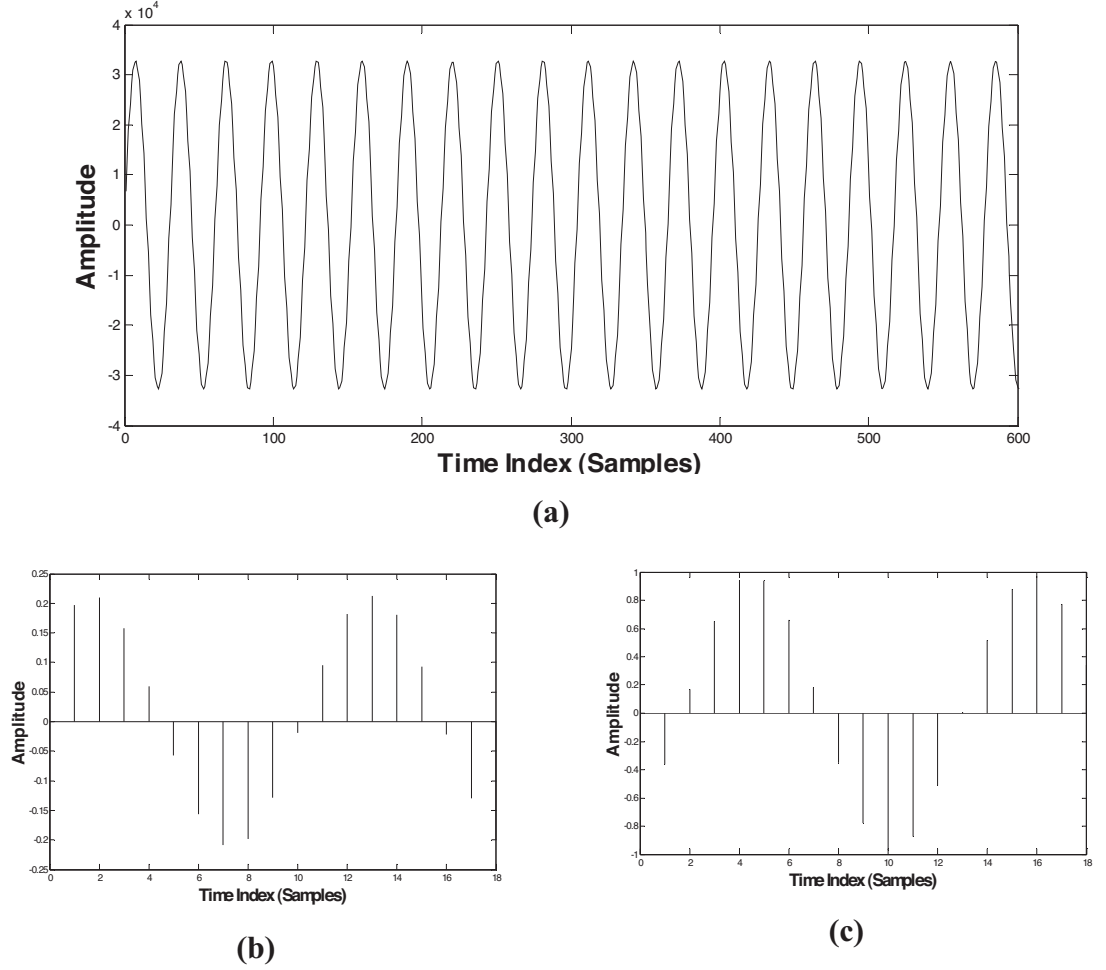


Figure 2.2: (a) Input – single tone of 1500 Hz, (b) Output of subband 2, (c) Output of subband 3.

2.3 FILTER BANK IMPLEMENTATION

The block diagram in Figure 2.3 illustrates the basic steps involved in the implementation of the pseudo-QMF polyphase filter bank. These filter banks are characterized by low-complexity implementations and can be efficiently employed using fast algorithms.

By combining the steps shown in Figure 2.3, the filter bank outputs can be derived as

$$s_i = \sum_{k=0}^{63} \sum_{j=0}^7 \mathbf{M}_{ik} (\mathbf{c}(k + 64j) \mathbf{x}(k + 64j)) \quad (2.1)$$

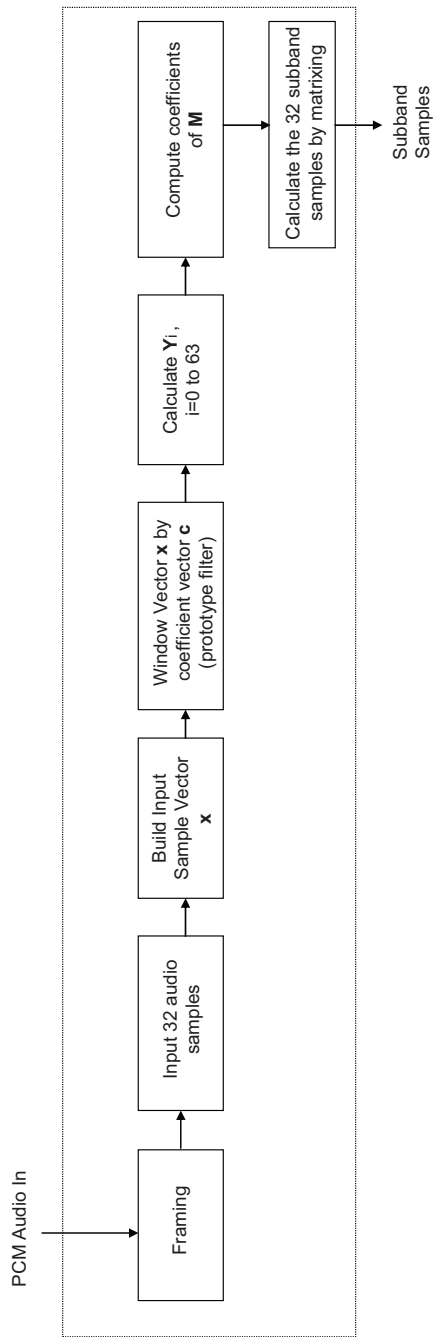


Figure 2.3: Implementation of the analysis subband filterbank.

where i is the subband index and ranges from 0 to 31, \mathbf{c} contains the analysis window coefficients, \mathbf{x} is the audio input sample vector, and \mathbf{M} is a 32×64 matrix with the analysis coefficients, computed as follows.

$$\mathbf{M}_{i,k} = \cos \left[\frac{(2i+1)(k-16)\pi}{64} \right], i = 0 \text{ to } 31, k = 0 \text{ to } 63. \quad (2.2)$$

In (2.2), the index i indicates the subband number. It can be observed that (2.1) can be easily reformulated into a familiar filter convolution expression,

$$s_i(n) = \sum_{m=0}^{511} x(n-m) h_i^c(m), \quad (2.3)$$

where $x(n)$ is an audio sample at time n , and in this form each subband of the filter bank has its own bandpass filter response, $h_i^c(n)$. Although this form is more convenient for analysis, it is not efficient for implementation. Each of the filters in the polyphase filter bank in (2.3) has a modulated impulse response, i.e.,

$$h_i^c(n) = h(n) \cos \left[\frac{(2i+1)(n-16)\pi}{64} \right], i = 0 \text{ to } 31, n = 0 \text{ to } 511. \quad (2.4)$$

The coefficients of the prototype filter, $c(n)$, relate to the impulse response, $h(n)$, as follows.

$$h(n) = \begin{cases} -c(n), & \text{if } \lfloor \frac{n}{64} \rfloor \text{ is odd} \\ c(n), & \text{else} \end{cases} \quad n = 0 \text{ to } 512. \quad (2.5)$$

The coefficients, $h(n)$, correspond to the prototype lowpass filter response of the polyphase filter bank. Figure 2.4 compares the plots of $c(n)$ with $h(n)$, while Figure 2.5 illustrates the magnitude responses of all 32 subbands of the polyphase filter bank. The $c(n)$ used in (2.1) has every odd-numbered group of 64 coefficients of $h(n)$ negated, so as to compensate for M_{ik} . The cosine term in (2.2) ranges from $k = 0$ to 63 and covers an odd number of half cycles, whereas the cosine term in (2.3) ranges from $n = 0$ to 511 and hence covers eight times the number of half cycles.

The expression for $h_i^c(n)$ in (2.4) clearly indicates that the response of each subband is a modulated version of the prototype response, with a cosine term to shift the lowpass response to the appropriate frequency band. The center frequencies of these filters are at odd multiples of $\pi/(64T)$, where T is the audio sampling period and each has a bandwidth $\pi/(32T)$.

As seen in Figure 2.5, the prototype filter response does not have a sharp cutoff. Furthermore, when the filter outputs are sub-sampled by a factor 32 (for critical sampling), a considerable amount of aliasing occurs. The steps involved in the calculation of the polyphase components and the implementation of the subband filters are described here.

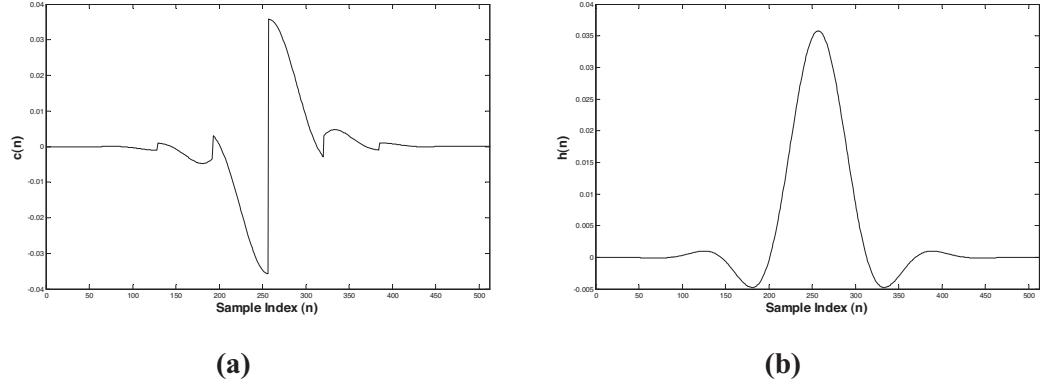


Figure 2.4: Coefficients of the prototype filter (a) $c(n)$ and (b) $h(n)$.

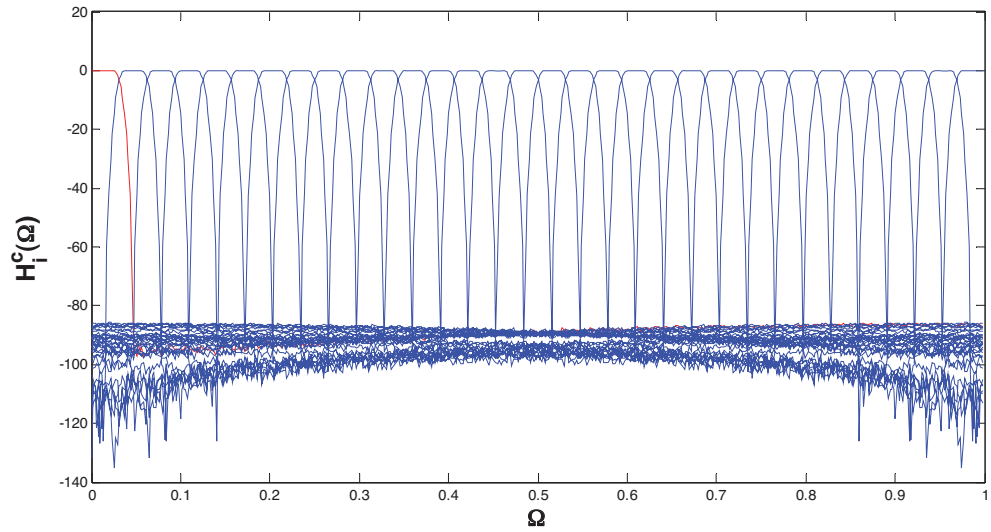


Figure 2.5: Composite magnitude responses of all subbands.

2.3.1 FRAMING

In this algorithm, the input audio is processed in frames of 1152 samples per audio channel. A granule is a group of 576 samples, i.e., half a frame of the input audio data. The subband analysis function takes one frame of the data as input (1152 samples) and produces two granules of 576 frequency lines each. Each granule is then split into 32 subbands with 18 samples in each. As seen in Program 2.1,

the input variable to the function, *data*, is of size 2×1152 , containing one frame of audio data for each channel. The outer *for* loop indexes the granules, while the inner *for* loop indexes over the channel number. The output variable is a 4-dimensional array $S(ch, gr, sb, samp)$, where *ch* refers to the channel, *gr* indicates the granule, *sb* denotes the subband number and *samp* contains the 18 samples for each subband.

2.3.2 BUILD INPUT SAMPLE VECTOR

An input sample vector of 512 elements is built using the 32 input audio samples. The most recently acquired 32 samples are inserted into the vector **x** at indices 0 to 31 and the oldest 32 samples shifted out of the vector **x**. This is done to ensure that the first 32 samples of the vector **x** contain the most recent audio samples. The vector $\mathbf{x} = [x_0, x_1, \dots, x_{511}]^T$ is updated as

$$x_i = x_{i-32}, \quad i = 511, 510, \dots, 32, \quad (2.6)$$

$$x_i = w_{31-i}, \quad i = 31, 30, \dots, 0. \quad (2.7)$$

```
% P 2.1 - Build the input sample vector for subband analysis
function S = SubbandAnalysis(data, channels)
global SBLIMIT HAN_SIZE x C M SCALE;
% Structure of the output - Subband samples
S = zeros(2, 2, 32, 18);

% initial data offset ...
gr_offset = 0;

% SBLIMIT = 32, HAN_SIZE = 512
for gr = 1:2 % Number of Granules
    for ch = 1:channels % Number of Channels
        for iter = 0:17 % 18 iterations
            % Replace 32 oldest samples with 32 new samples
            x(ch, HAN_SIZE:-1:SBLIMIT+1) = x(ch, HAN_SIZE-SBLIMIT:-1:1);
            x(ch, SBLIMIT:-1:1) = data(ch, gr_offset+iter*SBLIMIT+1:gr_offset...
                                     +(iter+1)*SBLIMIT)/SCALE;
```

Program 2.1: MATLAB Code for Building the Input Sample Vector.

2.3.3 WINDOW VECTOR \mathbf{x}

The input sample vector \mathbf{x} is windowed by the vector \mathbf{c} , which contains the coefficients of the prototype filter. The 512 coefficients of the lowpass prototype filter are plotted in Figure 2.4 and the values can be found in the file *analysis_window.m*. Figure 2.6 illustrates the sample input vector \mathbf{x} , impulse response of the window \mathbf{c} and the windowed output vector \mathbf{z} . The relation between \mathbf{x} , \mathbf{c} and \mathbf{z} can be expressed as,

$$z_i = c_i x_i, i = 0 \text{ to } 511, \quad (2.8)$$

where, z_i , x_i and c_i are the elements of the vectors \mathbf{z} , \mathbf{x} and \mathbf{c} , respectively.

```
% P 2.2 - Window Vector x by C
z = zeros(HAN_SIZE, 1);
z = x(ch, 1:HAN_SIZE) .* C;
```

Program 2.2: MATLAB Code for Windowing Vector \mathbf{x} .

2.3.4 CALCULATION OF THE COSINE MODULATION MATRIX

The cosine modulation filter bank, represented by a 32×64 matrix \mathbf{M} , is used to calculate the 32 subband samples. Program 2.3 shows the MATLAB code to compute the cosine modulation matrix.

```
% P 2.3 - Compute coefficients for the matrix M of size 32x64
% Cosine-Modulated Filter bank
M = zeros(32, 64);
for i = 0:31
    for k = 0:63
        M(i+1, k+1) = cos((2*i+1)*(16-k)*pi/64);
    end
end
```

Program 2.3: MATLAB Code for computing \mathbf{M} .

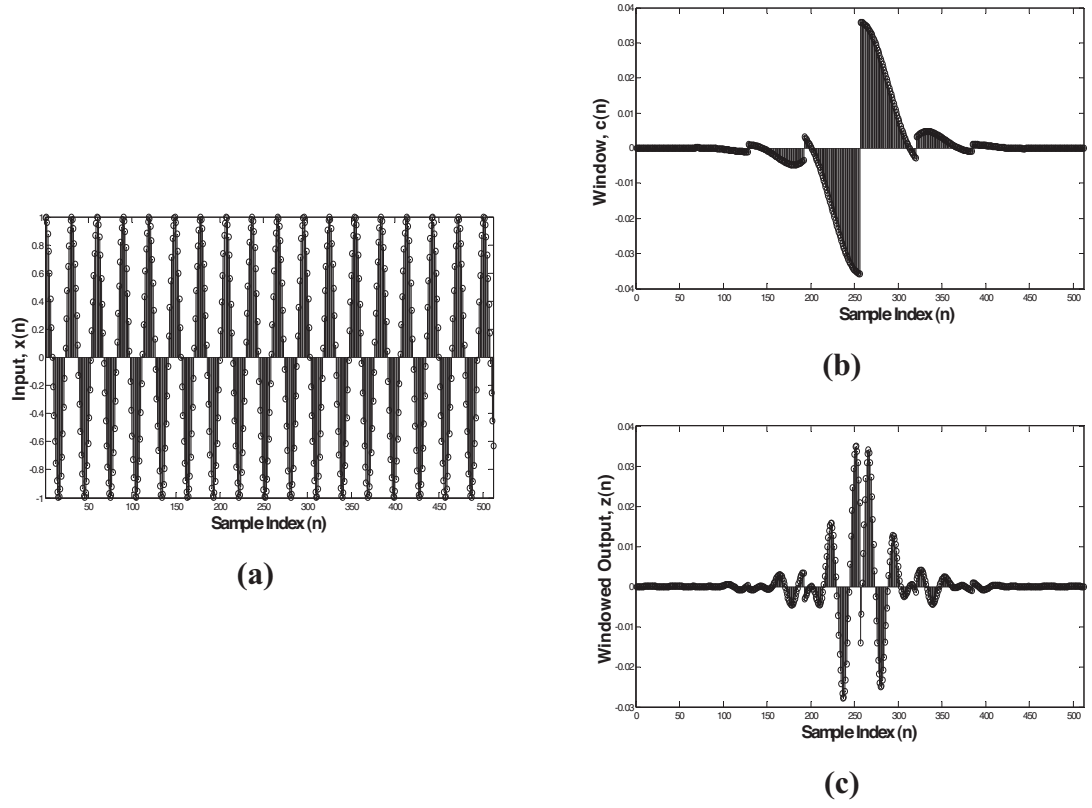


Figure 2.6: (a) Sample input vector x , (b) Impulse response of the window, (c) Windowed output vector z .

2.3.5 CALCULATION OF THE SUBBAND SAMPLES

The two steps involved in the computation of the 32 subband samples are shown below.

$$y(i) = \sum_{j=0}^7 z(i + 64j), \quad i = 0 \text{ to } 63, \quad (2.9)$$

$$s_i = \sum_{k=0}^{63} M_{i,k} y(k), \quad i = 0 \text{ to } 31. \quad (2.10)$$

```

% P 2.4 - Calculation of the output Y
y = zeros(1, 64);
for i = 1:64
    y(i) = sum(z(i:64:HAN_SIZE));
end
% Calculate the 32 subband samples
for i = 1:SBLIMIT
    S(ch, gr, i, iter+1) = sum(y.*M(i, 1:64));
end

```

Program 2.4: MATLAB Code for computing subband samples.

2.4 DEMONSTRATION WITH TEST DATA

Figure 2.7 illustrates the impulse responses and magnitude responses of three different subbands. The output **S** contains 32 subbands with 18 samples in each subband.

2.5 SUMMARY

In this chapter, we described the design and implementation of the subband analysis filter bank used in the MPEG-1 Layer III encoder. It was observed that this filter bank does not reflect the frequency dependent behavior of the human auditory system. The PCM audio signal is divided into 32 subbands by passing it through a filter bank and decimating the output signals by a factor of 32. Since it is not possible to construct filters with perfectly flat responses, aliasing effects can be introduced during the decimation process and these result in some information loss. As evident from the impulse response of the subbands, the subband filter functions do not have a linear phase. However, the cascade of the analysis and synthesis filters for each subband channel is a linear phase filter [78]. Therefore, the entire filter bank is of linear phase and the aliasing will only result in amplitude distortion. Though the aliasing errors of the filters appear substantial from the frequency responses, they are reduced substantially during reconstruction. The reader is referred to [7, 79, 80, 81, 82, 83, 84, 85] for further details on filter banks and their design.

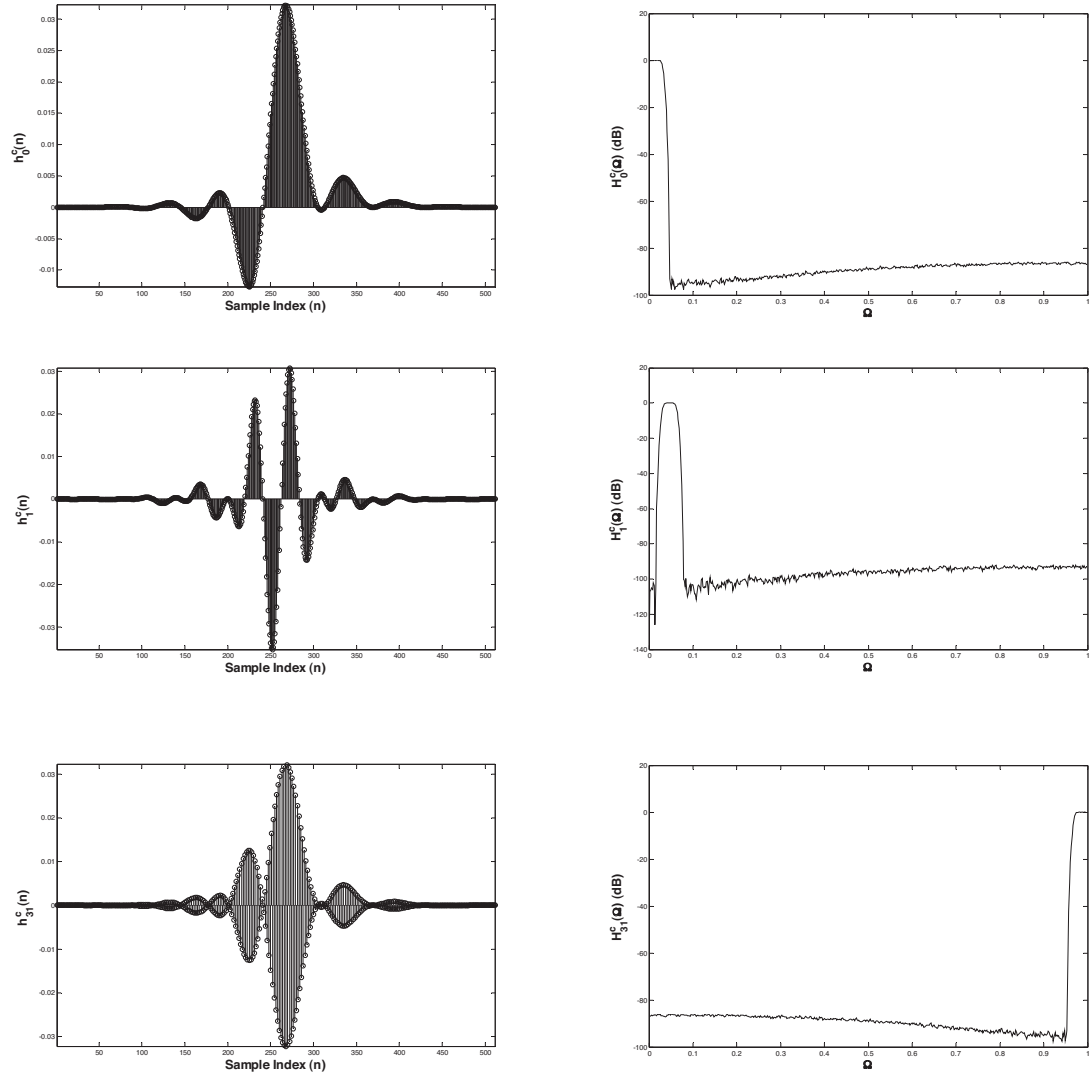


Figure 2.7: Time domain and Frequency responses of the filter bank. Subbands 0,1 and 31 are shown, respectively.

Psychoacoustic Model II

3.1 DESCRIPTION

The MPEG-I psychoacoustic model plays a central role in the compression algorithm. This module produces a time-adaptive spectral pattern that emulates the sensitivity of the human sound perception system. The MPEG audio coder compresses the audio data by removing its perceptually irrelevant components. This is done by exploiting the auditory masking characteristics. The psychoacoustic module analyzes the signal, and computes the noise and signal masking as a function of frequency. The masking characteristics of a given signal component depend on its frequency location and its intensity. The encoder estimates a masking threshold for each subband, below which an audio signal cannot be perceived by the average human listener [86]. The encoder uses this information to determine the best representation of a signal with the limited number of bits. Furthermore, it calculates an estimate of the Perceptual Entropy (PE) which is used to make window-switching decisions [87]. The computation of the masking thresholds and the PE is described in the following sections. The block diagram in Figure 3.1 illustrates the basic steps involved in the psychoacoustic model II as used in the Layer III algorithm.

3.2 ILLUSTRATION OF THE PSYCHOACOUSTIC MODEL II WITH MATLAB CODE

3.2.1 ANALYSIS

The calculation of masking threshold involves the critical band analysis of the signal. Hence, the power spectrum of the signal, which is computed from the complex FFT spectrum of the signal, needs to be partitioned into *critical bands*. The cochlear filter pass bands are of non-uniform bandwidth and the bandwidths increase with increasing frequency. The *critical bandwidth* is a function of frequency that quantifies the cochlear filter bandwidths. For an average listener, the critical bandwidth can be approximated by [73, 88]

$$BW_c(f) = 25 + 75 \left[1 + 1.4 (f/1000)^2 \right]^{0.69} \text{ (Hz)} . \quad (3.1)$$

Table 1.1 shows an idealized filter bank and Figure 3.2 illustrates the corresponding magnitude responses.

The psychoacoustic module needs to account for the delay of the filter bank and for an offset in order to center the data in the audio frame within the psychoacoustic analysis window [27]. This model uses an independent time-to-frequency mapping instead of the polyphase filter bank because

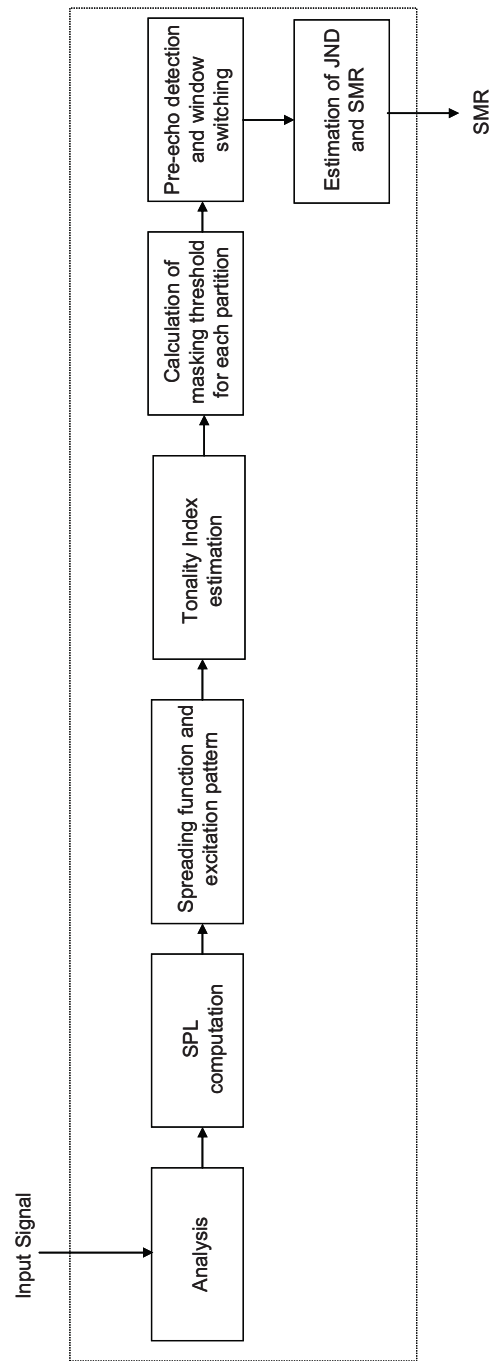


Figure 3.1: Overview of the psychoacoustic model II.

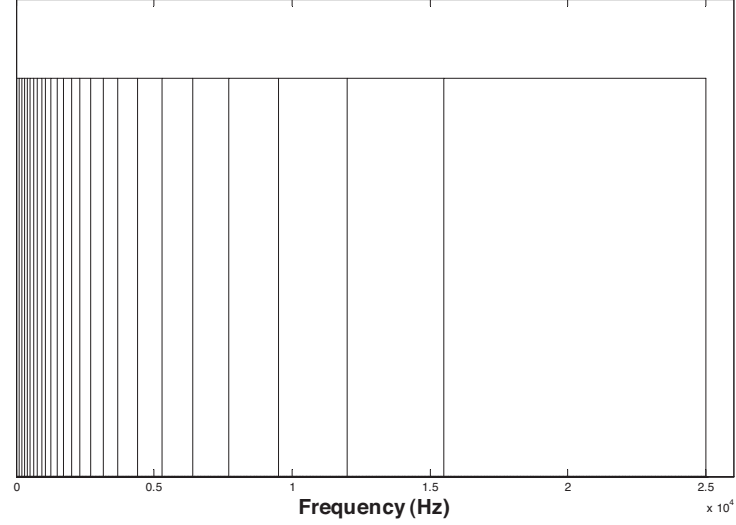


Figure 3.2: Idealized critical band filter bank.

calculation of the masking thresholds requires a finer frequency resolution. The psychoacoustic model II uses the FFT for this mapping. A Hanning window is applied to the audio data, prior to the Fourier transformation, to reduce the edge effects of the transform window. The size of an audio frame in the MPEG-1 algorithm is 1152 samples and the size of the analysis window in the case of long windows is 1024 samples [89]. It can be easily seen that the analysis window cannot cover the frame completely. Hence, Layer III performs two psychoacoustic computations per frame. The first computation deals with the first half of the 1152 samples centered in the analysis window and the other computation deals with the second half. The module finally combines the results of the two calculations and chooses the higher of the two signal-to-mask ratios for each subband. This in effect selects the lower of the two noise masking thresholds for each subband.

The frame size of 1152 samples (32 subbands*36 samples) ensures a frequency resolution of about 41.66 Hz at the sampling rate of 48 kHz. The drawback of this approach is that the quantization errors can be spread over a block of 1152 time samples. In signals containing sharp transients, this results in unmasked temporal noise and pre-echo. Therefore, the Layer III filter bank switches to a higher time resolution to avoid this artifact. During transients, Layer III uses a shorter block size of 384 samples (32 subbands*12 samples), thereby reducing the temporal spreading of the quantization noise. The psychoacoustic model II can choose between long windows of 1024 samples and short windows of 256 samples for analysis. As described earlier, the psychoacoustic model is calculated twice with a delay of 576 samples and the largest of each pair of signal to mask ratios is used. One computation is done with a delay of 192 samples to the audio data, while the other computation is

28 3. PSYCHOACOUSTIC MODEL II

carried out with a delay of 576 samples. In the former case, short windows of 256 samples are used. The analysis stage involves windowing of the time-aligned audio samples, using a Hanning window.

```
% P 3.1 - Spectral Analysis
% flush=576 ; sync_flush=768
% Delay signal by sync_flush=768 samples
delaybuffer(chn,1:sync_flush)=delaybuffer(chn,...
1+flush:sync_flush+flush);

% It is important to note which part of the data is inserted
buf_idx_start = (gr-1)*576+1;
delaybuffer(chn, sync_flush+1:sync_size) = ...
    PCMAudio(chn, buf_idx_start:buf_idx_start+flush-1);
Psycho.InputData = delaybuffer(chn, 1:BLKSIZE);

% Window data with Hanning window
waudio = window.*delaybuffer(chn, 1:BLKSIZE);
```

Program 3.1: MATLAB Code for spectral analysis in the Psychoacoustic module.

Figure 3.3 (a) illustrates the time-aligned input data to the psychoacoustic module for granule 1, while Figure 3.3 (b) shows the output after windowing the data with a Hanning window.

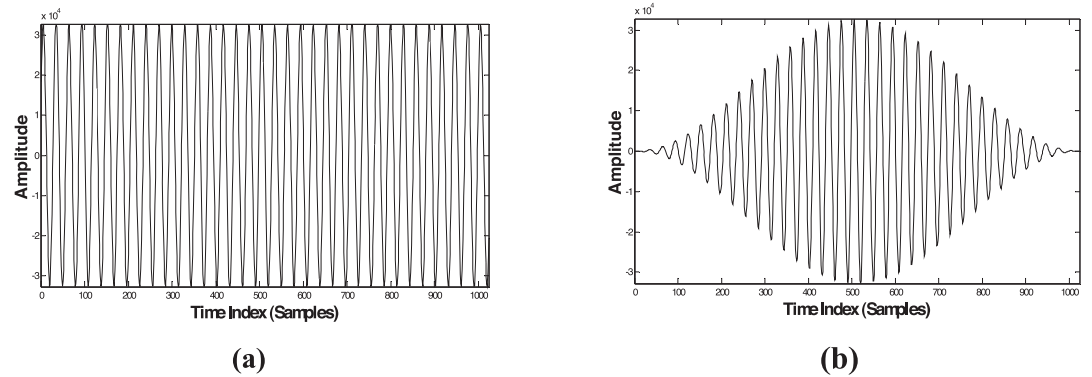


Figure 3.3: (a) Time-aligned input data, (b) after applying a Hanning window.

3.2.2 COMPUTATION OF ENERGY AND UNPREDICTABILITY IN THRESHOLD PARTITIONS

The important step involved in this module is to identify the tone-like and noise-like components in the audio signal. The masking abilities of these two components vary considerably [90, 91].

However, the psychoacoustics model II never actually separates completely the tonal and non-tonal components. Instead, it computes a likelihood measure known as the *tonality index* which determines if the component is more tone-like or noise-like. This index measure is estimated as a function of predictability of the spectral components in the current analysis window. In general, tonal components are more predictable than the broadband signals and they exhibit different masking characteristics [92]. Hence, the unpredictability measure of each spectral component is computed in this module, using the complex spectra of the time-aligned and the windowed audio data. The module uses data from two previous analysis windows to predict the components in the current window. A measure of unpredictability is given by,

$$cw(f) = \frac{\sqrt{\left[\left(R_j(f) \cos(\phi_j(f)) - \hat{R}_j(f) \cos(\hat{\phi}_j(f)) \right)^2 + \left(R_j(f) \sin(\phi_j(f)) - \hat{R}_j(f) \sin(\hat{\phi}_j(f)) \right)^2 \right]}{\left(R_j(f) + |\hat{R}_j(f)| \right)} \quad (3.2)$$

where

$$\begin{aligned} \hat{R}_j(f) &= 2R_{j-1}(f) - R_{j-2}(f), \\ \hat{\phi}_j(f) &= 2\phi_{j-1}(f) - \phi_{j-2}(f). \end{aligned}$$

Here, $R_j(f)$ denotes the magnitude and $\hat{R}_j(f)$ represents the predicted magnitude of the complex spectrum in the current threshold calculation block. It can be seen that $\hat{R}_j(f)$ is calculated from the complex spectra of the two previous blocks. The index j denotes the current block and the indices $j - 1$ and $j - 2$ index the data in the two previous blocks.

Computing the unpredictability measure for all the spectral lines up to 20 kHz would require a large number of computations. Hence, in this model, the computation is performed only for the first 206 lines and a constant value of 0.4 is set to the remaining spectral lines. The unpredictability of the first 6 lines is calculated using a long FFT (window length=1024), while that of the remaining spectral lines up to 205 is computed using a short FFT block (window length=256).

$$cw(f) = \begin{cases} c_l(f) & \text{for } 0 \leq f < 6 \\ c_s\left(\frac{f+2}{4}\right) & \text{for } 6 \leq f < 206 \\ 0.4 & \text{for } f \geq 206 \end{cases}, \quad (3.3)$$

where $c_l(f)$ and $c_s(f)$ are the unpredictability measures computed using the long FFT and short FFT, respectively. The MATLAB code section in 3.2 illustrates the computation of the unpredictability measure. Figure 3.4 (a) shows the FFT magnitude spectrum of the windowed audio data in Figure 3.3(b). The unpredictability measure of the frequency lines is illustrated in Figure 3.4 (b).

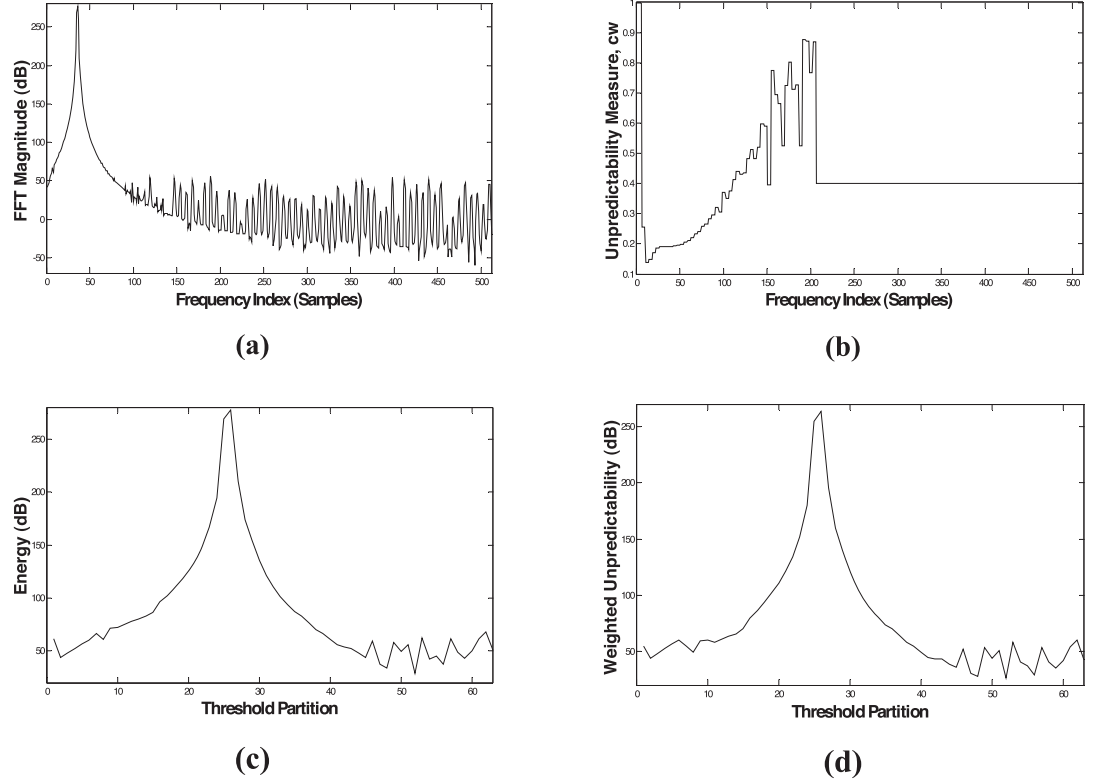


Figure 3.4: (a) FFT magnitude of the windowed audio data, (b) Unpredictability measure (cw), (c) Energy of threshold partitions (eb), (d) Weighted unpredictability of threshold partitions (cb).

It can be clearly seen that the dominant frequency components in the spectrum have a very low unpredictability measure.

To simplify the psychoacoustic calculations, the module processes the frequency lines in the perceptual domain. In model II, the spectral lines are grouped into *threshold calculation partitions*, whose widths are roughly 1/3 of a critical band or one FFT line, whichever is wider at that frequency location. At low frequencies, a single line of the FFT constitutes a single partition whereas many lines are combined into a single calculation partition at higher frequencies. The total number of partitions depends on the choice of the sampling rate. For each threshold partition, the energy is derived using the sum of energy densities in the partition. The energy in each threshold partition is

given by

$$eb(z) = \sum_{f=bl_z}^{bh_z} R^2(f) , \quad (3.4)$$

where bl_z and bh_z denote the lower and higher frequency limits of the z^{th} critical band. The weighted unpredictability of each partition is given by,

$$cb(z) = \sum_{f=bl_z}^{bh_z} R^2(f) cw(f) , \quad (3.5)$$

where $cw(f)$ is the unpredictability measure from (3.2). Figure 3.4 (c) and (d) illustrate the energy and weighted unpredictability in the threshold partitions, respectively. It can be seen from Figure 3.4 (c) that the perceptual transformation (mapping) expands the low frequency regions and compresses the high frequency regions. Since cb weights the unpredictability measure cw by the energy in each partition, the weighted unpredictability looks similar to the energy in the partition.

3.2.3 SPREADING FUNCTION AND EXCITATION PATTERN

A strong signal component reduces the audibility of weaker components in the same critical band and also the neighboring bands. Hence, the masking ability of a given signal spreads across its neighboring critical bands also [93, 94]. The psychoacoustic module emulates this by applying a *spreading function* to spread the energy of a critical band across other bands. The total masking energy of the audio frame is derived from the convolution of the spreading function with each of the maskers. This is carried out by spreading the energy of each masker in frequency and computing the sum of relative energies. The spreading function (measured in dB) used in this module is given by [95]

$$B(dz) = 15.8111389 + 7.5 * (a * dz + 0.474) - 17.5 * \sqrt{1.0 + (a * dz + .474)^2} \\ + 8 * \min \left(0, (a * dz - 0.5)^2 - 2(a * dz - 0.5) \right) , \quad (3.6)$$

where dz is the bark distance between the maskee and the masker and

$$a = \begin{cases} 3 & \text{for } dz \geq 0 \\ 1.5 & \text{else} \end{cases} . \quad (3.7)$$

It is important to note that the model uses only values of the spreading function greater than 60 dB.

$$spf(i, j) = \begin{cases} 0.0 & B(dz) \leq -60 \\ 10^{\frac{(x+B(dz))}{10}} & \text{else} \end{cases} . \quad (3.8)$$

```

% P 3.2 - Compute unpredictability measure in the threshold partitions

% complex FFT
fft_waudio = fft(waudio, 1024 ); % long FFT
energy = abs(fft_waudio).^2;
phi = angle(fft_waudio);

% Calculate unpredictability measure cw
for j = 1:6
    r_prime = 2.0 * r(chn, old, j) - r(chn, oldest, j);
    phi_prime = 2.0 * phi_sav(chn, old, j) - phi_sav(chn, oldest, j);
    r(chn, new, j) = sqrt(energy(j));
    phi_sav(chn, new, j) = phi(j);
    temp1 = r(chn, new, j) * cos(phi(j)) - r_prime * cos(phi_prime);
    temp2 = r(chn, new, j) * sin(phi(j)) - r_prime * sin(phi_prime);
    temp3 = r(chn, new, j) + abs(r_prime);

    if temp3 ~= 0.0
        cw(j) = sqrt( temp1*temp1+temp2*temp2 ) / temp3;
    else
        cw(j) = 0;
    end
end

% Compute unpredictability of the next 200 spectral lines
for sblock = 1:3
    % window data with HANN window
    k = 128 * (2 + sblock-1);
    waudio(1:BLKSIZE_s) = window_s.*delaybuffer(chn, k:k+BLKSIZE_s-1);

    % short FFT
    fft_waudio = fft(waudio, 256);
    energy_s(sblock, :) = abs(fft_waudio).^2;
    phi_s(sblock, :) = angle(fft_waudio);
end

sblock = 1;

```

Program 3.2: MATLAB Code for computing unpredictability measure. (*Continues.*)

```

% Calculate unpredictability measure cw
for j = 7:4:206
    k = (j+1)/4 ;
    r_prime = 2.0 * sqrt(energy_s(1, k)) - sqrt(energy_s(3, k));
    phi_prime = 2.0 * phi_s(1, k) - phi_s(3, k);
    r2 = sqrt(energy_s(2, k));
    phi2 = phi_s(2, k);
    temp1 = r2 * cos( phi2 ) - r_prime * cos( phi_prime );
    temp2 = r2 * sin( phi2 ) - r_prime * sin( phi_prime );
    temp3 = r2 + abs( r_prime );

    if temp3 ~= 0.0
        cw(j) = sqrt( temp1 * temp1 + temp2 * temp2 ) / temp3;
    else
        cw(j) = 0.0;
    end

    cw(j+1) = cw(j);
    cw(j+2) = cw(j);
    cw(j+3) = cw(j);
end

% Set unpredictability of the remaining spectral lines to 0.4
cw(207:HBLKSIZE) = 0.4;

% Calculate the energy and the unpredictability in the threshold
% calculated partitions
cb(1, CBANDS) = 0;
for j = 1:HBLKSIZE
    tp = partition_1(j);
    if tp > 0
        eb(tp) = eb(tp) + energy(j);
        cb(tp) = cb(tp) + cw(j)*energy(j);
    end
end
end

```

Program 3.2: (*Continued.*) MATLAB Code for computing unpredictability measure.

The code Section 3.4 demonstrates the calculation of spreading function given in (3.6).

The basilar excitation pattern per partition is computed by convolving the energy in each partition with the spreading function.

$$ecb(z) = \sum_{b=1}^{z_{\max}} eb(z_b)spf(zm_b, zm), \quad (3.9)$$

34 3. PSYCHOACOUSTIC MODEL II

```

% P 3.3 - Compute the spreading function

% s[j][i], the value of the spreading function, centered at band j, for
band i.
for i = 1:part_max
    for j = 1:part_max
        temp_x = (bval_1(i) - bval_1(j))*1.05;

        % Choice of parameter a
        if j >= i
            temp_x = (bval_1(i) - bval_1(j))*3;
        else
            temp_x = (bval_1(i) - bval_1(j))*1.5;
        end

        % Computation of B(dz) in Eq. 3.5
        if (temp_x >= 0.5) & (temp_x <= 2.5)
            temp = temp_x - 0.5;
            x = 8.0 * (temp*temp - 2.0 * temp);
        else
            x = 0.0;
        end
        temp_x = temp_x + 0.474;
        temp_y = 15.811389 + 7.5*temp_x - 17.5*sqrt(1.0+temp_x*temp_x);

        % Computation of spf in Eq. 3.7
        if temp_y <= -60.0
            s3_1(i, j) = 0.0;
        else
            s3_1(i, j) = exp( (x + temp_y)*LN_TO_LOG10 );
        end
    end
end
end

```

Program 3.3: MATLAB Code for calculating spreading function.

where z_m is the mean bark value of the partition z and z_{\max} is the largest partition index for a given sampling rate. Also the unpredictability measure in each partition is convolved with the spreading function to take the spreading effect into account.

$$ctb(z) = \sum_{b=1}^{z_{\max}} cb(z_b)spf(z_{mb}, z_m) . \quad (3.10)$$

Figure 3.5 illustrates the spreading functions centered at bands 4, 5, 6 and 7, respectively. Since the spreading function is applied in the perceptual domain, the spreading function is relatively uniform as a function of frequency. Figure 3.6 (a) shows the output obtained by convolving the

% P 3.4 – Convolve the partitioned energy and unpredictability with the spreading function.

```
for b = 1:CBANDS
    ecb(b) = sum(s3_l(b, 1:CBANDS) .* eb(1:CBANDS));
    ctb(b) = sum(s3_l(b, 1:CBANDS) .* cb(1:CBANDS));
end
```

Program 3.4: MATLAB Code for applying spreading function.

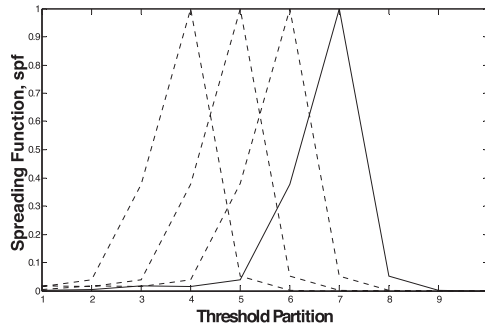


Figure 3.5: Spreading functions centered at bands 4, 5, 6 and 7, respectively.

energy in each partition with the spreading function and the spread unpredictability is shown in Figure 3.6 (b).

3.2.4 TONALITY INDEX ESTIMATION

The masking characteristics of tonal and non-tonal components are different and hence it becomes essential to separate them [96]. The tonality index is hence computed as a function of frequency, which indicates the tonal nature of the spectral component. The tonality index in each partition is calculated based on whether the signal is predictable from the spectral lines in the two previous frames. It is computed as a linear extrapolation of the components from two prior analysis windows. Tonal components are highly predictable and hence have a relatively higher index.

To compute the tonality index, the result of the spread unpredictability measure in (3.10) is normalized using the spread signal energy in (3.9),

$$cbb(z) = \log \left(\frac{ctb(z)}{ecb(z)} \right) . \quad (3.11)$$

The expression inside the parentheses in (3.11), (ctb/ecb) , is plotted in Figure 3.7 (b). It can be clearly seen that the normalization by the spread energy in each partition results in a behavior

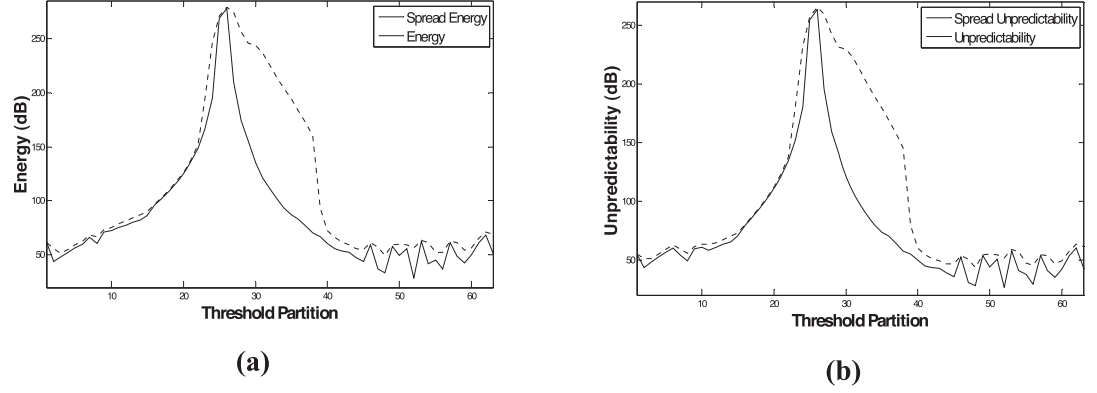


Figure 3.6: (a) Spread energy (*ecb*), (b) Spread Unpredictability (*ctb*).

similar to the unpredictability measure in Figure 3.3 (b). Figure 3.7 (c) shows the parameter *cbb* as given in (3.11).

The measure $cbb(z)$ is then mapped onto tonality index which is a function of the partition number and whose values vary between zero and one.

$$tbb(z) = -0.299 - 0.43 * cbb(z), \quad 0 < tbb(z) < 1. \quad (3.12)$$

Since the unpredictability measure is convolved with the spreading function that determines the masking energy at any frequency location, the resulting tonality index reflects the tonality of dominant maskers at that frequency location. The tonality indices of the threshold partitions are illustrated in Figure 3.7 (d). As described, the tonality indices range from 0 (high unpredictability) to 1 (low unpredictability) corresponding to the unpredictability. The relation between the parameters *tbb* and *cbb* given in (3.12) can also be clearly seen in the plots. As expected, tonal components have a higher tonality index compared to the non-tonal components. The code section 3.5 demonstrates the estimation of tonality index of the threshold partitions.

3.2.5 CALCULATION OF MASKING THRESHOLDS FOR THRESHOLD PARTITIONS

The masking thresholds are calculated with a higher frequency resolution than that provided by the polyphase filter bank. The masking threshold is determined by providing an offset to the excitation pattern, where the value of the offset strongly depends on the nature of the masker [97, 98]. The tonality indices evaluated for each partition are used to determine the offset of the re-normalized convolved signal energy, which converts it into the global masking level. The values for the offset are interpolated based on the tonality index of a noise masker to a frequency-dependent value defined in the standard for a tonal masker. The interpolated offset is compared with a frequency dependent

% P 3.5 - Calculate the tonality of each threshold calculation partition

```

for b = 1:CBANDS
    if ecb(b) ~= 0.0
        cbb = ctb(b)/ecb(b);
        if cbb < 0.01
            cbb = 0.01;
        end
        cbb = log(cbb);
    else
        cbb = 0.0 ;
    end

    tbb = -0.299 - 0.43*cbb; % conv1=-0.299, conv2=-0.43
    tbb = min(1.0, max(0.0, tbb) ) ; % 0<tbb<1
    tonality(b) = tbb;
end

```

Program 3.5: MATLAB Code for estimating tonality index.

minimum value, *minval*, defined in the standard and the larger value is used as the signal to noise ratio (SNR). In the MPEG-1 standard, Noise masking Tone (NMT) is set to 6 dB and Tone masking Noise (TMN) to 29 dB for all partitions. The offset is obtained by weighting the maskers with the estimated tonality index.

$$O(z) = 29tbb(z) + 6(1 - tbb(z)) . \quad (3.13)$$

The SNR in each of the threshold partitions can be computed as,

$$SNR(z) = \max[\min val(z), O(z)] , \quad (3.14)$$

where $\min val(z)$ is the lower limit for SNR in the partition. It is predetermined and stored in tables for the corresponding sampling rates. The computed SNR for each threshold partition is shown in Figure 3.8 (a). Transforming SNR into the power domain yields,

$$bc(z) = 10^{\frac{-SNR(z)}{10}} . \quad (3.15)$$

The spreading function, because of its shape, increases the energy estimates in each band due to the effects of spreading. Hence, a renormalization process is used, by multiplying each partition by inverse of the energy gain, assuming a uniform energy of 1 in the partition.

$$enb(z) = ecb(z)rnorm(z) , \quad (3.16)$$

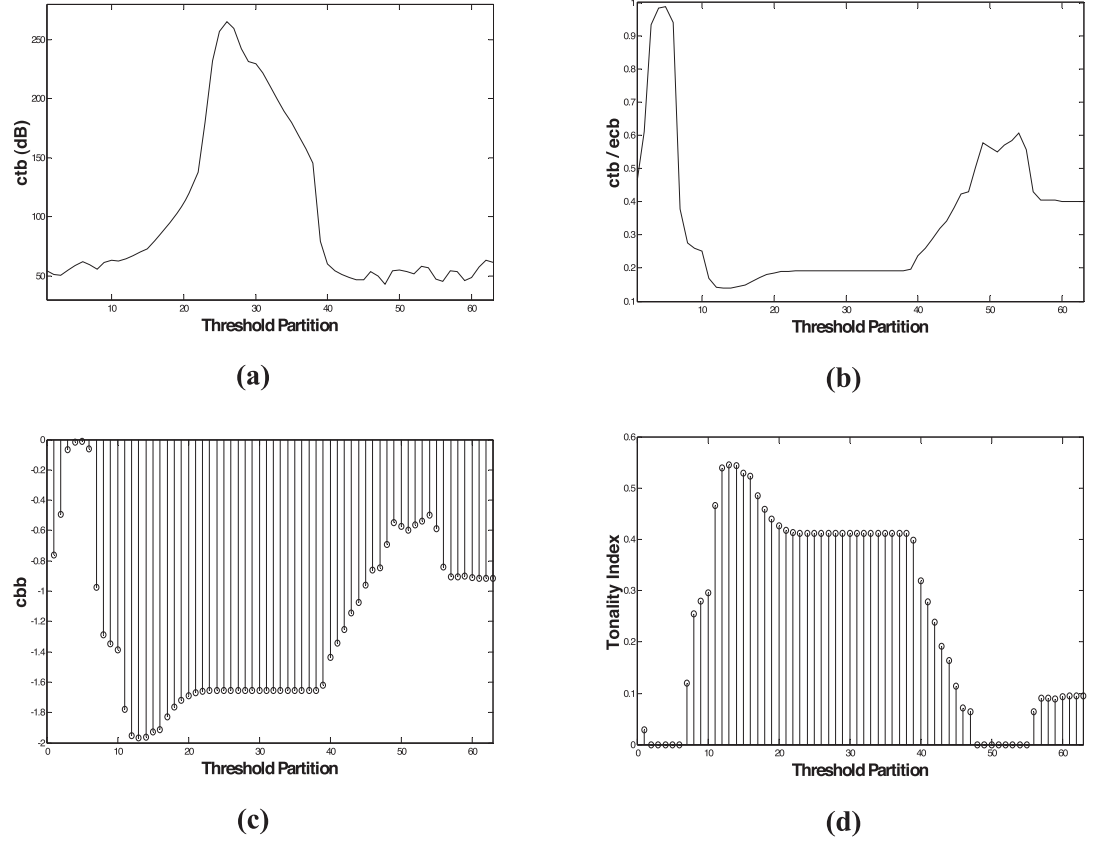


Figure 3.7: (a) Spread unpredictability (ctb), (b) Plot of ctb/ecb , (c) Normalized spread unpredictability (cbb), (d) Tonality index (tbb).

where

$$rnorm(z) = \frac{1}{\sum_{b=0}^{z_{\max}} spf(zm_b, zm)} . \quad (3.17)$$

The actual energy threshold for each partition is hence given by

$$nb(z) = enb(z)bc(z) . \quad (3.18)$$

Figure 3.8 (b) illustrates the energy, spread energy and the actual threshold energy in each threshold calculation partition.

```

% P 3.6 - Calculate the energy threshold for each partition
for b = 1:CBANDS
    % Calculation of SNR for each partition
    % TMN=29.0,NMT=6.0 for all calculation partitions
    SNR_l(b) = max( minval(b), 29.0*tbb+6.0*(1.0-tbb) );
end

% calculate the threshold for each partition
nb(1:CBANDS)=ecb(1:CBANDS).*norm_l(1:CBANDS).*exp(-...
SNR_l(1:CBANDS)*LN_TO_LOG10);

```

Program 3.6: MATLAB Code for calculating energy threshold.

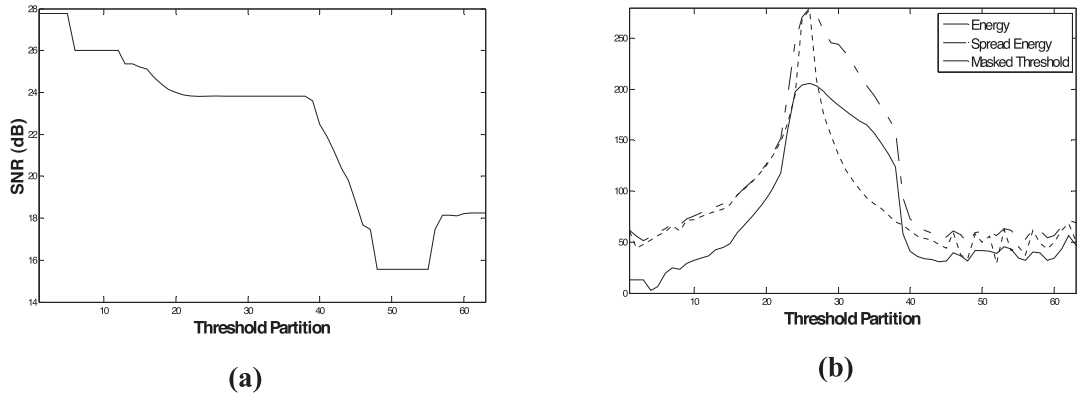


Figure 3.8: (a) SNR of the threshold partitions, (b) Masked thresholds of the threshold partitions.

The threshold derived for the current frame is compared with that of the two previous frames and the threshold in quiet. The maximum of the three values is chosen to be the actual threshold.

$$thr(z) = \max [Tq(z), \min [nb(z), nb_{t-1}(z), nb_{t-2}(z)]] , \quad (3.19)$$

$$nb_{j-1}(z) = 2.nb(z), \quad (3.20)$$

$$nb_{j-2}(z) = 16.nb(z) , \quad (3.21)$$

where $Tq(f)$ is the absolute threshold of hearing, $nb_{j-1}(z)$ and $nb_{j-2}(z)$ are the energy thresholds from the two previous frames. The absolute threshold of hearing (threshold in quiet) characterizes the amount of energy needed in a pure tone such that it can be detected by a listener in a noiseless environment [99]. The threshold is well approximated by the nonlinear function [100],

$$Tq(f) = 3.64 \left(\frac{f}{1000} \right)^{-0.8} - 6.5e^{-0.6 \left(\frac{f}{1000} - 3.3 \right)^2} + 10^{-3} \left(\frac{f}{1000} \right)^4 \text{ (dB SPL)} , \quad (3.22)$$

40 3. PSYCHOACOUSTIC MODEL II

which is representative of a young listener with acute hearing. Figure 3.9 illustrates the threshold in quiet curve as a function of frequency. The threshold in quiet is set to be the lower bound on the audibility of sound and can be interpreted as a maximum allowable energy level for coding distortions introduced in the frequency domain. The absolute threshold is typically expressed in terms of Sound Pressure Level (SPL) [88]. The SPL gives the intensity of the sound pressure in decibels relative to an internationally defined reference level.

$$L_{SPL}(\text{dB}) = 20 \log_{10} (p/p_0) , \quad (3.23)$$

where p is the sound pressure of the stimulus in Pascals and p_0 is the standard reference level of $20\mu\text{Pa}$.

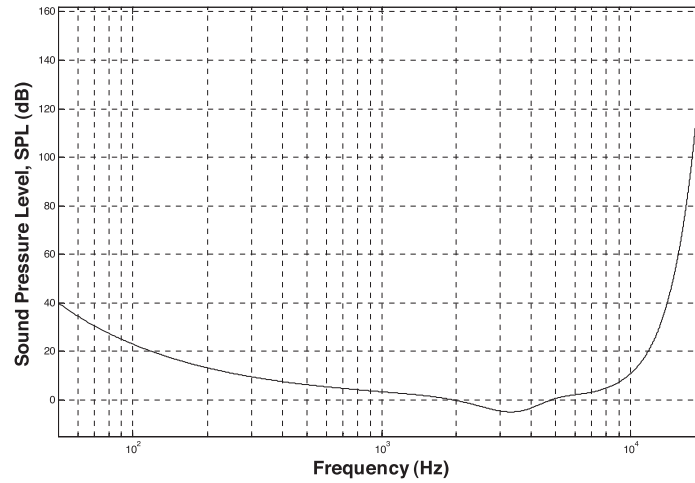


Figure 3.9: Approximate spectral curve for the threshold in quiet.

3.2.6 PRE-ECHO DETECTION AND WINDOW SWITCHING

Pre-echoes occur when a signal with a sharp attack begins near the end of a transform block immediately following a region of low energy [73]. This situation can arise when encoding recordings of percussive instruments such as the triangle, the glockenspiel, or the castanets, for example shown in Figure 3.10 (a). For a block-based algorithm, when quantization and encoding are performed, the time-frequency uncertainty dictates that the inverse transform will spread quantization distortion evenly in time throughout the reconstructed block [Figure 3.10 (b)]. This results in unmasked distortion throughout the low-energy region preceding in time the signal attack at the decoder. The MPEG-1 algorithm uses the MDCT to transform the blocks containing the subband data, which in turn are encoded and transmitted. The longer the block length the better is the frequency

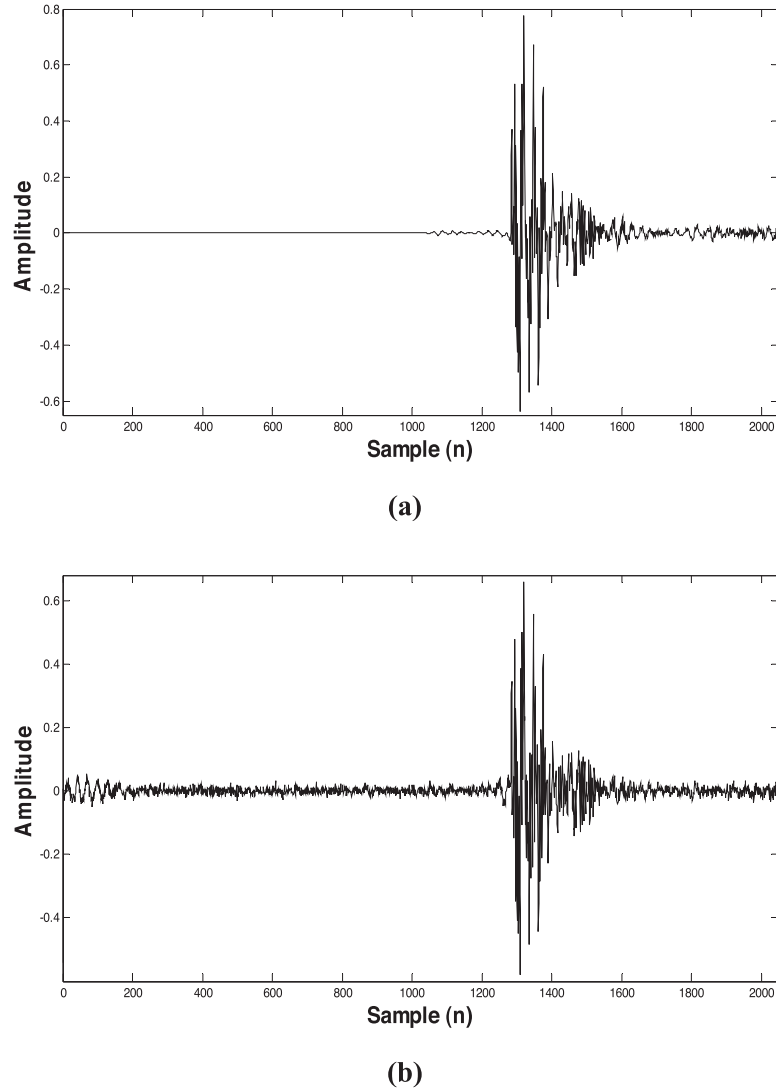


Figure 3.10: Pre-echo example: (a) uncoded castanets, (b) transform coded castanets.

resolution, which comes at the expense of losing time resolution. For quasi-stationary signals, longer blocks provide better compression. On the other hand, the transients are well captured by shorter blocks. Hence, adapting the size of blocks based on the statistics of the signal will yield best results. When long windows are used for sharper transients, the signal is coarsely quantized. In the time domain, the quantization noise spreads over the entire block and it would create a perceptible noise

42 3. PSYCHOACOUSTIC MODEL II

at the beginning of the audio. Pre-echo can be controlled by detecting such transients and making a decision to switch to short windows [101, 102]. It is implemented by calculating the perceptual entropy from the masking threshold and switching the window when it exceeds a pre-determined value [86].

Perceptual entropy indicates the average minimum number of bits per frequency sample needed to encode a signal without introducing a perceptual difference with respect to the original signal. It gives a lower bound estimate for the perceptual coding based on the computed mask threshold. Perceptual Entropy can be defined as,

$$PE = - \sum_{b=1}^{Z_{\max}} \left[cbwidth(b) \cdot \log \left(\frac{thr(b)}{eb(b) + 1} \right) \right], \quad (3.24)$$

where $cbwidth(b)$ is the width of the calculated threshold partition and $eb(b)$ is the energy in the threshold partition.

% P 3.7 - Pre-echo detection and calculation of Perceptual Entropy for window switching

% pre-echo control

```
for b = 1:CBANDS
    temp_1 = min(nb(b), min(2.0*nb_1(chn, b), 16.0*nb_2(chn, b)));
    thr(b) = max(qthr_1(b), temp_1);
    nb_2(chn, b) = nb_1(chn, b);
    nb_1(chn, b) = nb(b);
end
```

% calculate perceptual entropy

```
pe(gr, chn) = 0.0;
for b = 1:CBANDS
    tp = min(0.0, log((thr(b)+1.0)/(eb(b)+1.0)));
    pe(gr, chn) = pe(gr, chn) - numlines(b) * tp;
end
```

Program 3.7: MATLAB Code for pre-echo detection and calculation of PE.

An empirically predetermined value is used based on which window switching decision is made. The state-machine for the window switching is shown in Figure 3.11. It can be observed that whenever a sharp attack is identified or the Perceptual Entropy exceeds the predetermined value (1800), the algorithm switches to a *short* window. Notice that there are two transition windows, *start* and *stop*, which are used between the *long* and *short* windows. A description of the different window types and the scheme of window-switching can be found in Chapter 4. The corresponding program is listed in Program 3.8.

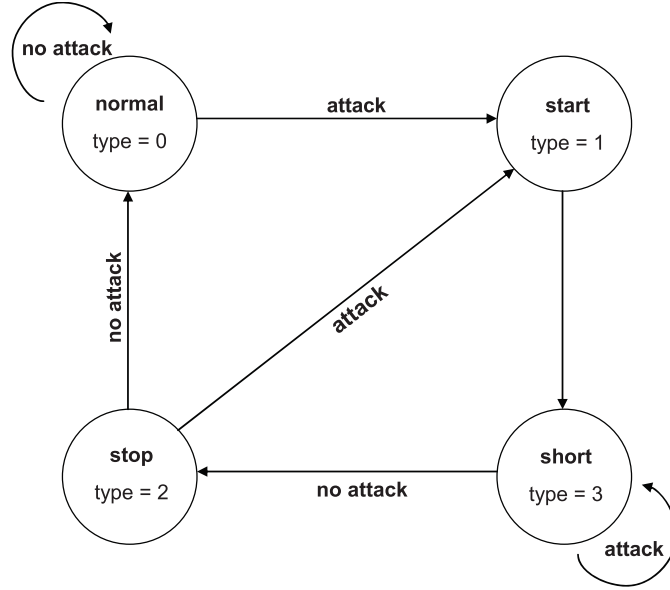


Figure 3.11: Window switching state machine.

3.2.7 ESTIMATION OF SMR

In MP3 encoding, the threshold is not spread over the FFT lines as done in the case of psychoacoustic model I. Instead, the threshold calculation partitions are converted directly into scalefactor bands. In general, scalefactors are employed to determine the quantization step size for a particular band to ensure that the resulting quantization noise level falls below the computed masked threshold. Scalefactor bands are groups of frequency lines which are scaled by a single scalefactor and they approximate critical bands. Hence, the signal to mask (SMR) ratio is evaluated in the scalefactor bands. There are predetermined tables which indicate the mapping between threshold partitions and the scalefactor band. For each sampling rate, there are 21 bands for long windows and 12 bands for short windows.

The energy in each scalefactor band is given by

$$en(sb) = w1.eb(bu) + \sum_{b=bu+1}^{b=b0-1} eb(b) + w2.eb(b0) , \quad (3.25)$$

44 3. PSYCHOACOUSTIC MODEL II

where $b0$ and bu are parameters used for converting threshold partitions into scale-factor bands. The threshold in each scalefactor band is given by

$$thm(sb) = w1.thr(bu) + \sum_{b=bu+1}^{b=b0-1} thr(b) + w2.thr(b0) . \quad (3.26)$$

The final output of the psychoacoustic model II, the signal-to-mask ratio (SMR) in each scale-factor band is defined as

$$SMR(sb) = \frac{thm(sb)}{en(sb)} . \quad (3.27)$$

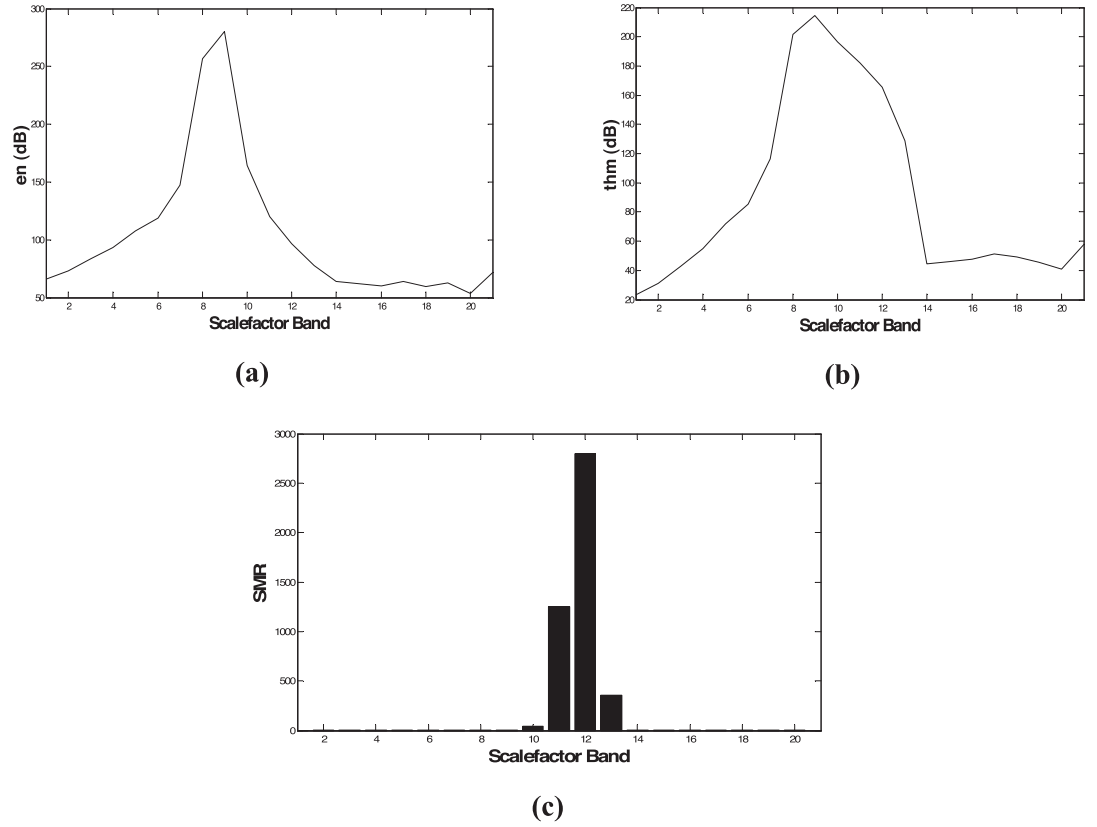


Figure 3.12: (a) Energy in scalefactor bands, (b) Thresholds in scalefactor bands, (c) Signal-to-mask ratio in scalefactor bands.

Figure 3.12 (c) illustrates the SMR in the scalefactor bands for the case when a long window is used. The program listing Program 3.8 shows the window switching and threshold computation for the case when long blocks are used. Similar steps can be carried out to perform the window switching and threshold computation for the case when short windows are to be used. Thus, the threshold calculation partitions are converted to codec partitions (scale-factor bands) and the SMR values computed are forwarded to the bit/noise allocation algorithm.

```
% P 3.8 - Window Switching State Machine and Calculation of SMR

switch_pe = 1800;
blocktype = NORM_TYPE;

if pe(gr, chn) < switch_pe      % no attack : use long blocks
    switch blocktype_old(chn)
    case NORM_TYPE,
    case STOP_TYPE,
        blocktype = NORM_TYPE;
    case SHORT_TYPE,
        blocktype = STOP_TYPE;
    case START_TYPE,
        error(dispatch(sprintf('Error in block selecting\n')));
    end

    % threshold calculation (part 2)
    % Calculation of energy and threshold in each scale-factor band
    for sb = 1:SBMAX_1
        en(sb) = w1_l(sb) * eb(bu_l(sb)+1) + w2_l(sb) * eb(bo_l(sb)+1);
        thm(sb) = w1_l(sb) * thr(bu_l(sb)+1) + w2_l(sb) * thr(bo_l(sb)+1);

        b = bu_l(sb)+1:bo_l(sb);
        en(sb) = en(sb) + sum(eb(b));
        thm(sb) = thm(sb) + sum(thr(b));

        if en(sb) ~= 0.0
            ratio(chn, sb) = thm(sb)/en(sb);
        else
            ratio(chn, sb) = 0.0;
        end
    end
end
```

Program 3.8: MATLAB Code for window switching and calculation of SMR.

3.3 SUMMARY

This chapter presented a detailed description of the psychoacoustic model II and its use in the analysis of masking characteristics of audio signals. The MPEG-1 Layer-3 algorithm achieves compression by allowing the quantization noise in the frequency subbands where the human ear is the least sensitive. The psychoacoustic model determines the maximum noise level which would be just perceptible (masking level) in each of the subbands from the input audio. Since quantization noise is directly related to the number of bits used by the quantizer, the bit allocation algorithm determines the bit assignment such that the audible distortion is minimized. The psychoacoustic model II does not make a dichotomous distinction between tonal and non-tonal components. Instead, it transforms the spectral data to a partition domain and estimates the tonality index in each partition. This measure actually determines the amount of masking. The partition domain approximates the critical band ranges. The reader in need of further details on psychoacoustics principles and the model used in perceptual coders is directed to [27, 72, 86, 88, 94].

CHAPTER 4

MDCT

4.1 DESCRIPTION

To achieve maximum coding gain, signals with strong and fine harmonic content require a high frequency resolution and coarse time resolution because the masking thresholds need to be localized in frequency. On the other hand, fast transients need adequate time resolution (short windows) to accurately estimate highly time localized temporal masking thresholds. However, most audio signals are highly non-stationary in nature and contain both steady-state and transient intervals. As a result, an effective coder should make adaptive decisions regarding the optimal time-frequency decomposition. This implies that the filter bank will have time-varying temporal and frequency resolutions. This motivates the use of hybrid filter structures in which switching decisions are made based on the changing signal characteristics. For this reason, the outputs of the 32-band polyphase filterbank are processed using the modified discrete cosine transform (MDCT) block. In essence, the 32 subband samples are further subdivided in the frequency domain using MDCT, thereby providing a good frequency resolution. The polyphase filter bank and the MDCT are together referred as the *hybrid filterbank*.

Being a lapped transform, the MDCT avoids artifacts from the block boundaries, in addition to providing energy compaction. The other important reason for the widespread use of MDCT in certain coding standards is the availability of FFT-based algorithms, which makes it suitable for real-time applications [103, 104]. The window switching for MDCT is controlled using the perceptual entropy measure obtained from the psychoacoustic module. After the audio signal is transformed to the frequency domain using the MDCT, a partial cancellation of the aliasing introduced in the analysis filter bank is performed. Figure 4.1 illustrates the hybrid filterbank structure used in MPEG-1 Layer III algorithm.

4.2 ILLUSTRATION OF THE MDCT WITH MATLAB CODE

4.2.1 MDCT CALCULATION

MDCT is a lapped transform that has half as many outputs as inputs. It is based on the type-IV discrete cosine transform with 50% overlap between the adjacent time windows. Thus, the MDCT basis functions extend across two blocks in time, thereby eliminating the block artifacts. Despite the 50% overlap, the MDCT is critically sampled and only M samples are generated for every $2M$ samples of the input block. Hence, it produces 18 frequency components for 36 time domain samples. The MDCT basis functions have a length $N = 2M$, where M denotes the number of subbands.

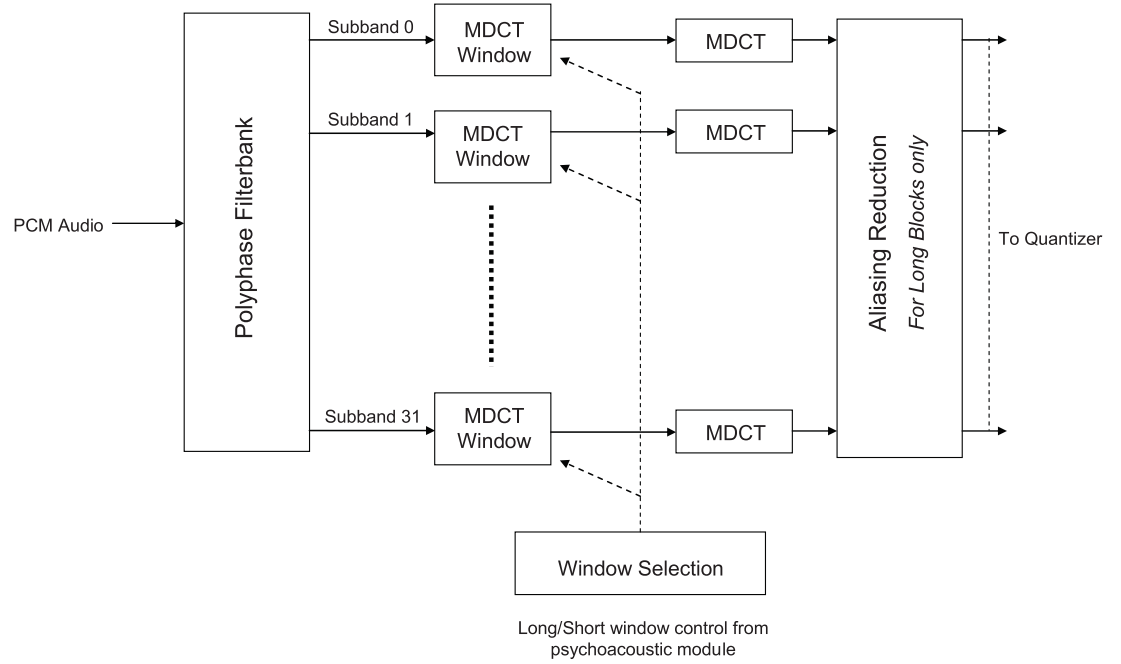


Figure 4.1: Illustration of the hybrid filter bank.

Given the analysis window $h(n)$, the output of the MDCT can be defined as

$$X(k) = \sqrt{\frac{2}{M}} \sum_{n=0}^{N-1} x(n)h(n) \cos \left[\left(n + \frac{M+1}{2} \right) \left(k + \frac{1}{2} \right) \frac{\pi}{M} \right], \quad (4.1)$$

where, $k = 0, 1, \dots, N/2 - 1$.

The analysis window $h(n)$ is chosen based on the perceptual entropy (PE) measure obtained from the psychoacoustic module. In effect, the MPEG codec changes the window length to match the signal characteristics of the input. A long window is used to maximize coding gain and achieve good channel separation for stationary segments. Short windows are used to localize time domain artifacts when pre-echos are likely. Because of the time overlap between the basis vectors, boundary filters or transition windows are used.

Clearly, the forward MDCT in (4.1) performs a series of inner products between the analysis filter impulse responses and the input. Prior to applying the MDCT to the subbands, each of the odd samples of odd subbands must undergo a frequency inversion correction so that the spectral lines appear in a proper ascending order. As seen in Program 4.1, the inversion is applied by multiplying each odd sample by -1 to compensate for the frequency inversion of the polyphase filter bank.

```

% P 4.1 - Frequency Inversion Correction

for gr = 1:mode_gr % Granules
    for ch = 1:chn % Channels

        cod_info = get_gr_info_from_l3_side(l3_side, gr, ch);
        % Type of the window as obtained from the side information
        block_type = cod_info.block_type;
        mixed_block_flag = cod_info.mixed_block_flag;

        % Compensate for inversion in the analysis filter bank
        % Inversion is applied only to the odd time samples of the odd
        subbands
        band = 2:2:32; time_index = 2:2:18;
        S(ch, gr, band, time_index) = -S(ch, gr, band, time_index);
    end
end

```

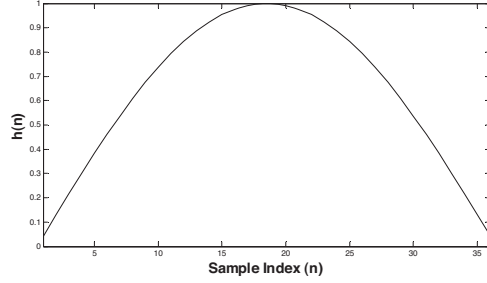
Program 4.1: MATLAB Code for frequency inversion correction.

The psychoacoustic model detects the pre-echo condition and chooses short windows for better time resolution and long windows for improved frequency resolution. When the perceptual entropy measure exceeds 1800, the MDCT filterbank is switched to use short windows. To maintain perfect reconstruction of MDCT, short and long windows are not switched instantaneously. For this purpose, two transition windows, start and stop are provided. The windows used in MDCT window switching are illustrated in Figure 4.2. It can be observed that the size of a short block is 12 samples, whereas that of a long block is 36 samples. In the short block mode, three short blocks with 50% overlap are used. Switching from a long to short window includes a transition start window while a stop window is placed in the transition from a short to a long window. Each block of data processed by MDCT contains 36 samples.

Figure 4.3 illustrates a typical case of window switching. It graphically depicts the switching from a long window to short window and back. There is a sequence of three short windows between the two long windows, with transition windows in between. For the first granule, the block of data passed to the MDCT is built using 18 samples of current subband and 18 samples from the memory. The memory contains 18 samples of second granule of the same subband in the previous frame. For the second granule, the block is built using 18 current samples and 18 samples from the first granule of the same subband in the current frame. Program 4.2 illustrates the above procedure for building data vector from the subbands for MDCT.

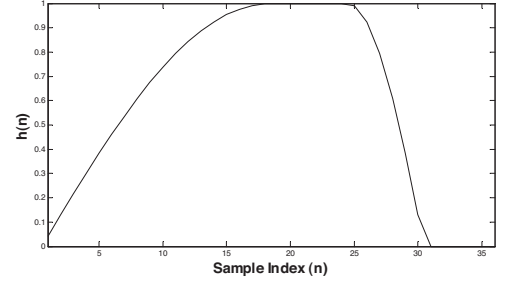
The calculation of the analysis window coefficients for the different windows as shown in Figure 4.2 can be found in the file *mdct_h.m*. Program 4.3 shows the calculation of coefficients for the different types of windows.

For a particular block of data, all the filterbank channels can have the same MDCT analysis window (short or long) or a mixed mode in which the two lowest frequency subbands have long



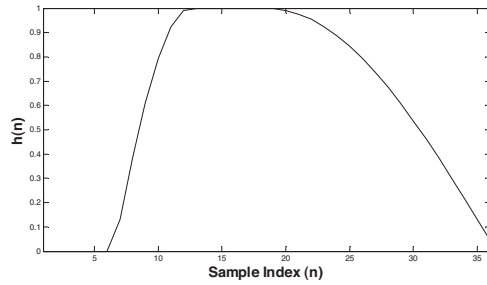
$$h(n) = x(n) \sin\left(\frac{\pi}{36}\left(n + \frac{1}{2}\right)\right), \quad n = 0 \text{ to } 35.$$

(a)



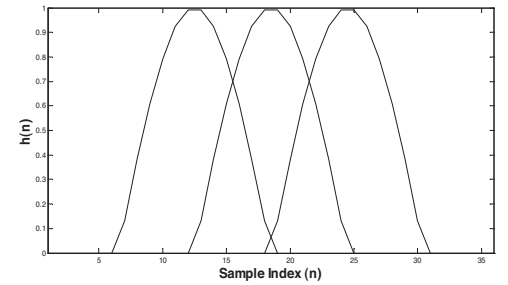
$$h(n) = \begin{cases} x(n) \sin\left(\frac{\pi}{36}\left(n + \frac{1}{2}\right)\right) & n = 0 \text{ to } 17 \\ x(n) & n = 18 \text{ to } 23 \\ x(n) \sin\left(\frac{\pi}{12}\left(n - 18 + \frac{1}{2}\right)\right) & n = 24 \text{ to } 29 \\ 0 & n = 30 \text{ to } 35. \end{cases}$$

(b)



$$h(n) = \begin{cases} 0 & n = 0 \text{ to } 5 \\ x(n) \sin\left(\frac{\pi}{12}\left(n - 6 + \frac{1}{2}\right)\right) & n = 18 \text{ to } 23 \\ x(n) & n = 24 \text{ to } 29 \\ x(n) \sin\left(\frac{\pi}{36}\left(n + \frac{1}{2}\right)\right) & n = 30 \text{ to } 35. \end{cases}$$

(c)



$$h^{(k)}(n) = y^{(k)}(n) \sin\left(\frac{\pi}{12}\left(n + \frac{1}{2}\right)\right) \quad \text{for } n = 0 \text{ to } 11, k = 0 \text{ to } 2$$

$$y^{(k)}(n) = \begin{cases} x(i+6) & k = 0, n = 0 \text{ to } 11 \\ x(i+12) & k = 1, n = 0 \text{ to } 11 \\ x(i+18) & k = 2, n = 0 \text{ to } 11. \end{cases}$$

(d)

Figure 4.2: Illustration of the windows used in the MDCT filter bank: (a) long window, (b) start window, (c) stop window, (d) 3 short windows.

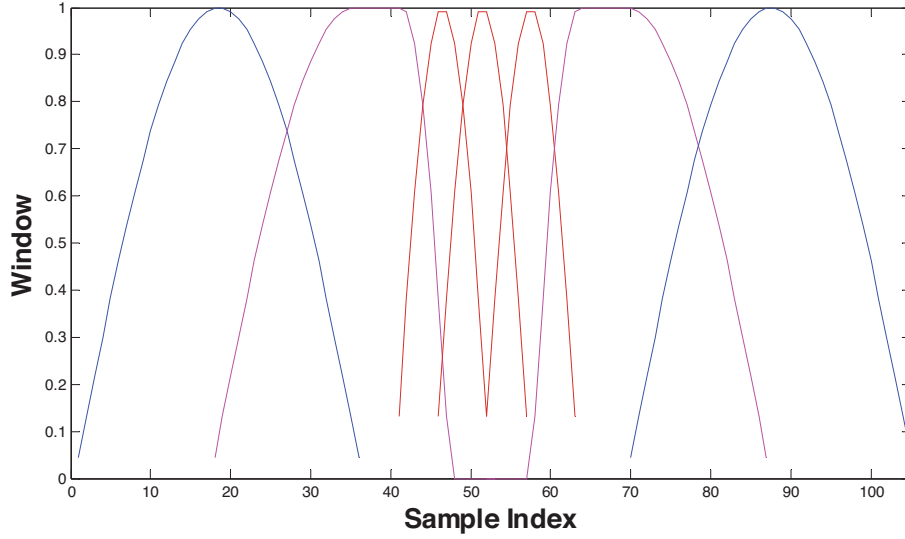


Figure 4.3: Illustration of a typical window switching.

blocks while the remaining 30 bands have short blocks. This ensures better frequency resolution at lower frequencies while having a higher time resolution for higher frequencies. The MDCT coefficients are computed as defined in Equation (4.1) and illustrated in Program 4.4.

4.2.2 ALIAS REDUCTION

The aliasing introduced due to the downsampling in the polyphase filter bank can be partially reduced at this stage, once MDCT maps the subband samples into the frequency domain. The alias reduction is applied only to the long blocks. The anti-aliasing butterfly for the encoder is shown in Figure 4.4. Each anti-aliasing butterfly is an orthonormal transformation applied to one of the eight designated pairs of spectral lines. They do not affect the reconstruction properties of the filterbank, but improve the compression factor of the coder by trying to contain the energy within each subband. The alias reduction computation can be described as

$$\begin{aligned} rMDCT(18sb - i - 1) &= rMDCT(18sb - i - 1)cs(i) - rMDCT(18sb + i)ca(i), \\ rMDCT(18sb + i) &= rMDCT(18sb + i)cs(i) - rMDCT(18sb - i - 1)ca(i), \end{aligned} \quad (4.2)$$

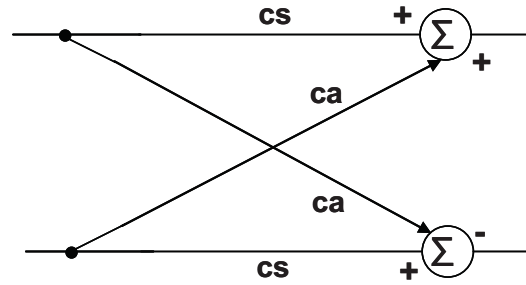


Figure 4.4: Alias reduction butterfly for the encoder.

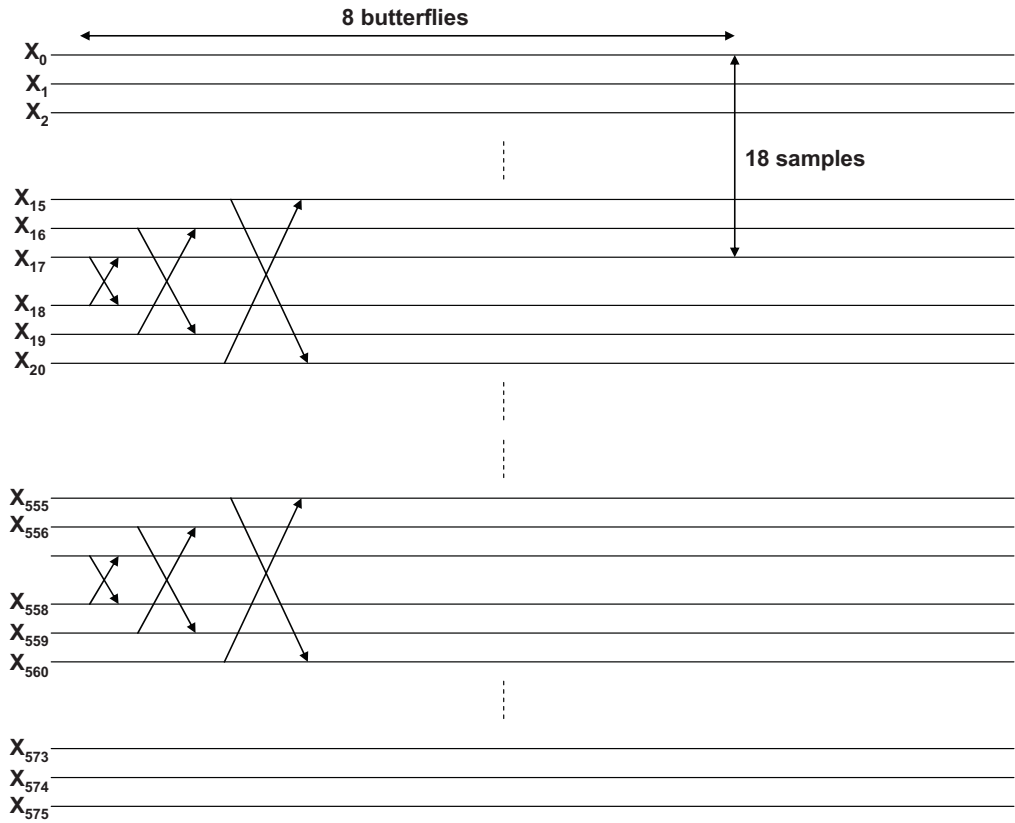


Figure 4.5: Alias reduction operations for a granule of MDCT data.

```

% P 4.2 - Build data from the subbands for MDCT computation

% MDCT of subband data using 18 previous subband samples + 18 current
subband samples

for band = 0:31
% For the first granule, get 18 previous subband samples from memory
    if gr == 1
        MDCTtime_in(1:18) = MDCTmemory(ch, band+1, 1:18);
        MDCTtime_in(19:36) = S(ch, gr, band+1, 1:18);
% For the second granule, get 18 previous subband samples from the
first granule
    else % gr == 2
        MDCTtime_in(1:18) = S(ch, gr-1, band+1, 1:18);
        MDCTtime_in(19:36) = S(ch, gr, band+1, 1:18);
    End
End

```

Program 4.2: MATLAB Code for building input data vector for the MDCT.

where $i = 0$ to 7 and $sb = 1$ to 31. Here, the indices of the array $rMDCT$ label the frequency lines in a granule, arranged in the lowest frequency to the highest frequency, with zero being the index of the lowest frequency line and 575 being the index of the highest. The butterfly coefficients cs and ci are calculated as

$$cs_i = \frac{1}{\sqrt{1 + c_i^2}} \quad \text{and} \quad ca_i = \frac{cs_i}{\sqrt{1 + c_i^2}}, \quad (4.3)$$

where c_i indicate the Layer III coefficients for alias reduction [27].

Figure 4.5 illustrates the alias reduction operations for a granule (576 samples) of MDCT data. Unlike the usual time-frequency plane, they are now rearranged for alias reduction, first in the order of subband and then in time. Therefore, 18 time-domain samples of each subband are grouped together. The MATLAB implementation of this alias-reduction procedure can found in Program 4.5.

4.3 SUMMARY

In Chapter 4, we outlined the design of a hybrid filterbank structure, in which switching decisions are made based on the changing signal characteristics. The MPEG-1 Layer III algorithm uses a set of 512-tap FIR filters for both analysis and synthesis of the PCM audio data. These alias-cancellation filters are designed such that high sidelobe attenuation is achieved. This algorithm employs an efficient implementation of these filters using a polyphase structure thereby reducing the computations. In the Layer-3 algorithm, each of the 32 subbands from the analysis subband filter bank is further subdivided into 18 spectral lines using the MDCT followed by the

```

% P 4.3 - Generation of Analysis Window coefficients

% type 0: normal = long blocks
win(1, 1:36) = sin(pi/36*((0:35) + 0.5));

% type 1: start block
win(2, 1:18) = sin(pi/36*((0:17) + 0.5));
win(2, 19:24) = 1;
k = 24:29;
win(2, 25:30) = sin(pi/12*(k + 0.5 - 18));
clear k;

% type 3: stop block
k = 6:11;
win(4, 7:12) = sin(pi/12*(k + 0.5 - 6));
clear k;
win(4, 13:18) = 1;
win(4, 19:36) = sin(pi/36*((18:35) + 0.5));

% type 2: short block
win(3, 1:12) = sin(pi/12*((0:11) + 0.5));

```

Program 4.3: MATLAB Code for computing analysis window coefficients.

application of alias reduction butterflies. The MDCT module provides better frequency resolution and the transient effects are controlled using different window functions. The reader in need of further details on pseudo QMF and its design is referred to [105, 106, 107, 108, 109, 110, 111].

```

% P 4.4 - The core MDCT routine

% INPUT VARIABLES
%   Time_in: vector of 36 time-domain subband outputs
%   blk_type: type of time-freq block - long, short, transition
% OUTPUT VARIABLES
%   Freq_out: vector of 18 MDCT-domain subband outputs

function [Freq_out] = MDCTcore(Time_in, blk_type)

% Initialized by mdct_h.m
global win cos_s cos_l;
Freq_out = zeros(1, 18);

% type 2: short block, but indexing for win[] = win(3, :)
if blk_type == 2
    N = 12; k = 1:N;
    for l = 0:2
        for m = 0:N/2-1
            Freq_out(3*m+1+1) = sum(win(blk_type+1,
                                         k).*Time_in(6+6*l+k).*cos_s(m+1, k));
        end
    end
else
    N = 36; k = 1:N;
    for m = 1:N/2
        Freq_out(m) = sum(win(blk_type+1, k).*Time_in(k).*cos_l(m, k));
    end
end
return

```

Program 4.4: MATLAB Code for computing MDCT coefficients.

```

% P 4.5 - Alias reduction butterflies for long blocks

if ( block_type ~= 2 )
% flash the alias-reduction block
    updateMainGui('Alias Reduction', 'start');

for band = 1:31 %Index for subband
    for k = 0:7 % Index for the spectral line

        bu = rMDCT(gr, ch, band*18-k)*cs(k+1) + rMDCT(gr, ch,
            band*18+k+1)*ca(k+1);

        bd = rMDCT(gr, ch, band*18+k+1)*cs(k+1) - rMDCT(gr, ch, band*18-
            k)*ca(k+1);

        rMDCT(gr, ch, band*18-k) = bu;
        rMDCT(gr, ch, band*18+k+1) = bd;
    end
end

```

Program 4.5: MATLAB Code for alias-reduction of MDCT data.

Bit Allocation, Quantization and Coding

5.1 DESCRIPTION

The bit allocation process determines the number of bits allocated to each of the subbands based on the information obtained from the psychoacoustic module. The Layer III loop module performs the key steps of non-uniform quantization and Huffman coding of the frequency lines from the MDCT module. Bit allocation and quantization of the spectral lines are realized in a nested loop procedure. The inner loop adjusts the non-uniform quantizer step sizes for each block until the number of bits required to encode the transform components falls within the available bit budget. The outer loop evaluates the quality of the encoded signal (analysis-by-synthesis) in terms of quantization noise relative to the masked thresholds. The non-uniform quantization depends on the variation of the audio signal and does not have a fixed execution time.

In Figure 5.1, we consider the case of a single masking tone occurring at the center of a critical band. Audio intensity levels in the figure are given in terms of Sound Pressure Level (dB). In the illustration, a hypothetical masking tone occurs at some masking level. This generates an excitation along the basilar membrane that is modeled by a spreading function and a corresponding masking threshold. For the band under consideration, the minimum masking threshold denotes the in-band minimum of the spreading function. Assuming the masker is quantized using an m -bit uniform scalar quantizer, noise might be introduced at the level m . SMR and noise-to-mask ratio (NMR) denote the log distances from the minimum masking threshold to the masker and noise levels, respectively. The Layer III encoder quantizes the spectral values by allocating the appropriate number of bits needed in each subband to maintain perceptual transparency at a given bit-rate. It controls and shapes the spectrum of the quantization noise to lie below audible levels. Therefore, the scheme is referred to as noise allocation, as opposed to bit allocation.

The bit allocation process of the Layer III encoder illustrated in Figure 5.2 can be described in three levels [27]. The top level, *loops frame program*, determines the scalefactor selection information (*scfsi*). The unit referred as the *outer loop* calculates the allowed distortion for each scalefactor band, as determined by the psychoacoustic module. It also computes the maximum number of bits available for each granule based on the bit reservoir size and the perceptual entropy. The *inner loop* quantizes the input vectors and modifies the quantizer step size until the output vector can be encoded with the available number of bits. The *outer loop* checks the distortion introduced by quantization in each scale factor band and if it exceeds the allowed distortion, amplifies the scalefactor band and calls

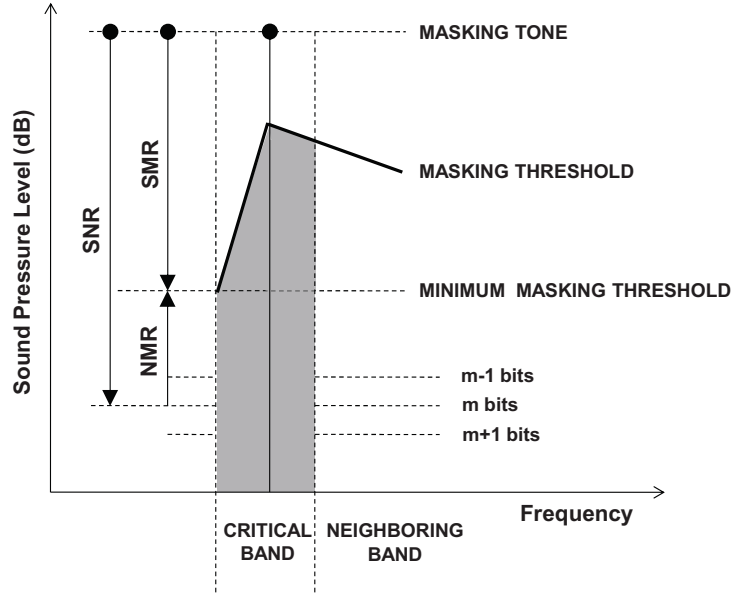


Figure 5.1: Illustration of simultaneous masking ([73, 93]).

the *inner loop* again. The encoder thus varies the quantizers iteratively using the inner and the outer loops to perform bit allocation.

The quantized values are encoded using Huffman coding. Huffman coding is a lossless entropy encoding algorithm and uses variable-length code tables. Noise shaping is performed to limit the quantization noise below the masking threshold. Hence, a global gain value that determines the quantization step size, and scalefactors that determine the noise shaping factors for each scalefactor band, are applied before the actual quantization. The process of finding the optimal gain and scalefactors for a given block at a specified bit-rate, using the Just Noticeable Difference (JND) thresholds obtained from the psychoacoustic module, is performed in the Layer III loop module.

5.2 THE LOOPS FRAME FUNCTION

5.2.1 CALCULATION OF THE SCALEFACTOR SELECTION INFORMATION (SCFSI)

The *scfsi* determines if the scalefactors of the first granule can be re-used for the second granule in the coding process. This implies that these scalefactors need not be transmitted twice and hence the bits saved in this process can be used in the Huffman coding. If the encoder operates in short block mode, then different scalefactors are always transmitted for each granule. The parameter *scfsi*

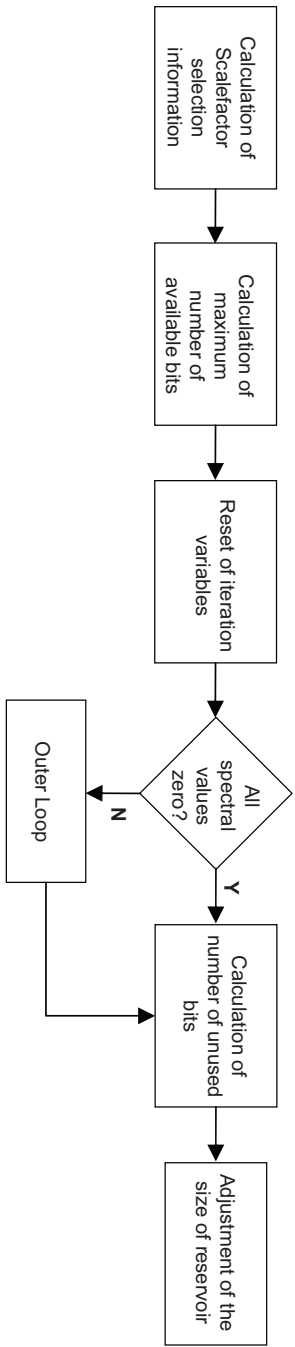


Figure 5.2: Implementation of the Layer III loop module.

60 5. BIT ALLOCATION, QUANTIZATION AND CODING

in Layer III is represented by one bit for each subband. To evaluate the parameter *scfsi*, the following information is needed for each granule.

- The block type (long/short).
- The total energy in the granule

$$en_{tot} = \text{int} \left\{ \log_2 \left(\sum_{i=1}^n |xr(i)|^2 \right) \right\} , \quad (5.1)$$

where n is the total number of spectral values and xr indicates the unquantized MDCT components.

- The energy in each scalefactor band, i.e.,

$$en(sb) = \text{int} \left\{ \log_2 \left(\sum_{i=lbl(sb)}^{lbl(sb)+bw(sb)-1} |xr(i)|^2 \right) \right\} , \quad (5.2)$$

where $lbl(sb)$ refers to the index of the first coefficient belonging to the scalefactor band sb and $bw(sb)$ indicates the number of coefficients within the scalefactor band.

- The distortion allowed for each scalefactor band, which is given by

$$xm(sb) = \text{int} \left\{ \log_2 (x \min(i)) \right\} , \quad (5.3)$$

where $xmin(sb)$ is the allowed distortion for each scalefactor band obtained from the psychoacoustics module.

The scalefactors for the first granule are always transmitted. When the second granule is encoded, the following comparisons of the two granules are performed in order to determine if the parameter *scfsi* can be enabled.

- All spectral values are not zero,
- The granules do not contain short blocks,
- $|en_{tot0} - en_{tot1}| < en_{totkrit}$,
- $\sum_{\text{all scalefactor bands}} |en(sb)_0 - en(sb)_1| < en_{difkrit}$.

The constants with the subscript *krit* have been pre-determined such that *scfsi* is enabled only if there is similarity in energy/distortion. If either of these criteria is not satisfied, *scfsi* is disabled, i.e., it is set to 0 in all scalefactor bands. Here, index 0 indicates the first granule and index 1 refers to the second.

5.3 DISTORTION CONTROL (OUTER LOOP)

The outer iteration loop controls the noise produced by the quantization of the frequency lines which is performed in the inner iteration loop. The scalefactors are used to shape the quantization noise. Quantization noise can be spectrally shaped such that it can be masked completely. The spectral shaping of the noise is done by multiplying the lines within the scalefactor bands with the actual scalefactors before quantization. In Layer III, several frequency lines are grouped to form scalefactor bands that are designed to resemble critical bands as closely as possible. The division of the spectrum into scalefactor bands is fixed for every block length and sampling frequency and is stored at both the encoder and decoder. The scalefactors are logarithmically quantized and the quantization step size is set by the parameter *scalefac_scale*. The scalefactor for frequency lines, beyond the highest line indicated by the tables, is zero, which means that the actual multiplication factor is 1. The distortion control loop always begins with *scalefac_scale*=0. If the distortion in any of the scalefactor bands exceeds the permissible distortion, the quantization step size is suitably adjusted for those scalefactor bands. Figure 5.3 shows the block diagram describing the implementation of the outer loop.

The scalefactors of all the scalefactor bands, *scalefac_l* and *scalefac_s*, and the quantizer step size are saved in this loop. If the execution of the outer loop is terminated, without having reached the final result, these values together with the quantized spectrum can be transmitted. For each outer iteration loop, the inner iteration loop is called to perform the actual quantization. The parameters required for performing rate control (inner loop) are the outputs of the hybrid filterbank with the scalefactors applied to the lines within the corresponding scalefactor bands and the total number of available bits. The actual distortion introduced by the quantization in the inner loop for each of the scalefactor bands is calculated as

$$xfsf(sb) = \sum_{i=lbl(sb)}^{lbl(sb)+bw(sb)-1} \frac{\left(|xr(i)| - ix(i)^4/3 * \sqrt[4]{2}^{qquant+quantaf} \right)^2}{bandwidth(sb)}, \quad (5.4)$$

where *lbl(sb)* indicates the index of the coefficient representing the lowest frequency in a scalefactor band, *bw(sb)* is the number of coefficients in the band and *qquant* indicates the quantizer step size.

The option for preemphasis, determined by the parameter *preflag*, enables the outer loop to amplify the upper part of the spectrum using the preemphasis tables. The *preflag* is switched *on* if the actual distortion exceeds the threshold in all of the four upper scalefactor bands, after the first call of the inner iteration loop. Furthermore, all spectral values of the scalefactor bands which have a distortion that exceeds the allowed distortion need to be amplified by a factor, which is transmitted by the parameter *scalefac_scale*. The process of the outer loop is continued until

- There is no scalefactor band that exceeds the allowed distortion;
- All scalefactor bands are already amplified;

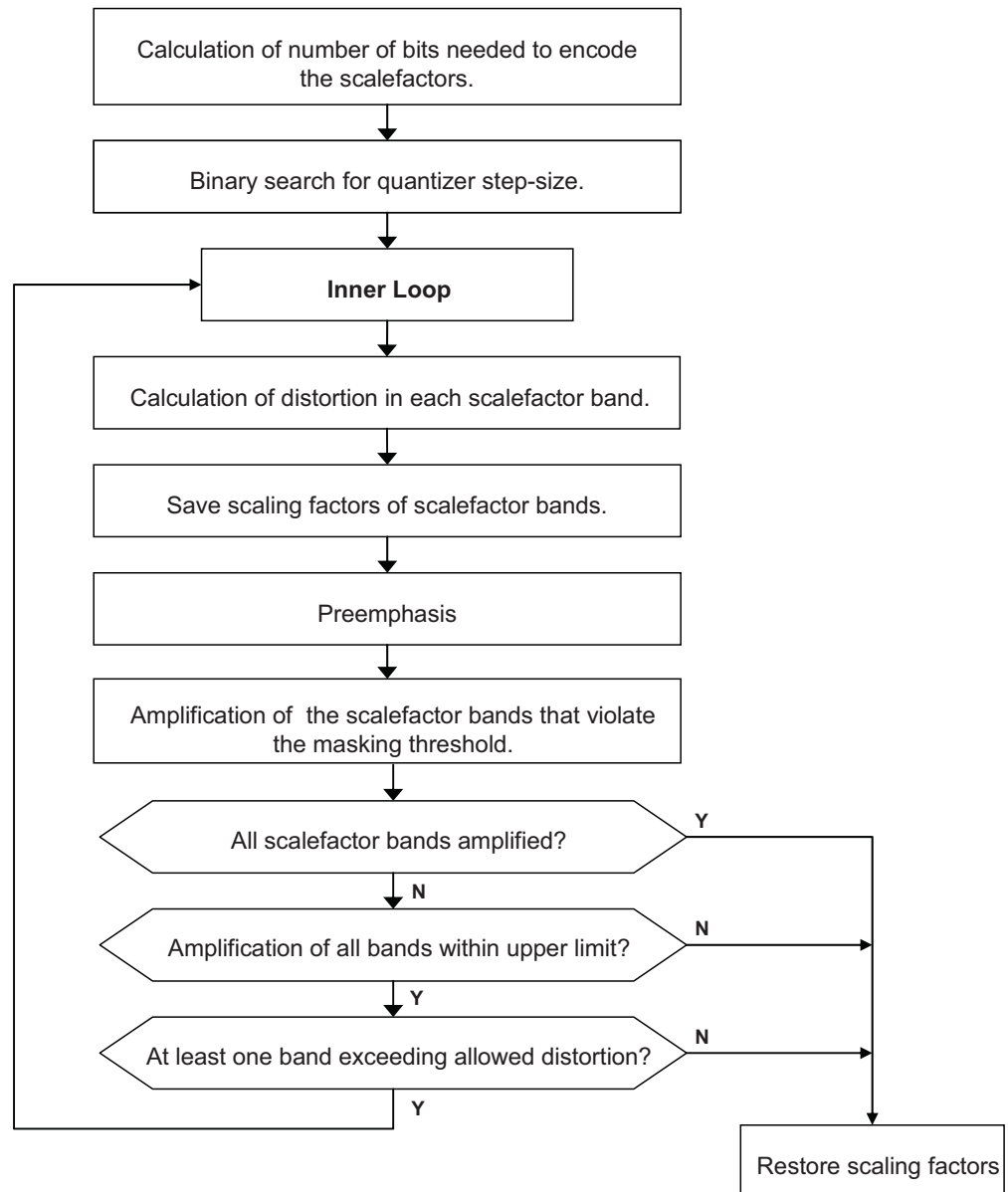
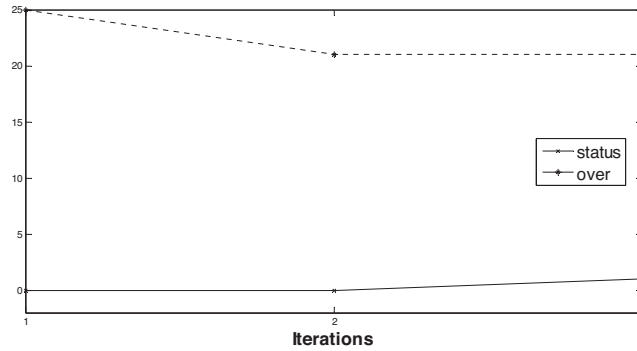


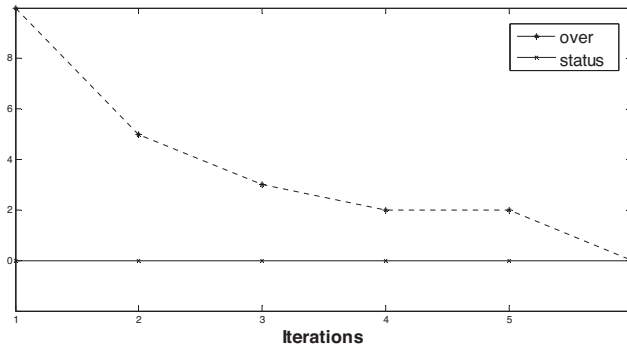
Figure 5.3: Implementation of the Outer Loop (Distortion Control).

- Amplification of at least one band exceeds the upper limit, which is determined by the transmission format of the scalefactors;
- Time-out is reached, in case of real-time implementations.

It can be seen from the code section in Program 5.1 that the two variables, *status* and *over*, control the execution of the outer loop. The parameter *status* is zero if there is at least a scalefactor band that has not been amplified and this indicates that the outer loop should be executed again. The variable *over* indicates if there is at least one scalefactor band that violates the masking threshold. Figure 5.4 illustrates two cases that demonstrate the effect of the two parameters in the outer loop execution. In Figure 5.4 (a) the termination is caused by the variable *status*, whereas in Figure 5.4 (b) the variable *over* terminates the outer loop.



(a)



(b)

Figure 5.4: Illustration of the effect of the variables *status* and *over* in the outer loop execution.

% P 5.1 - Outer Iteration Loop

```

status = 0;
over = 2^32;      % actual value returned by amp_scalefac_bands()

while status == 0 & over > 0
    cod_info = get_gr_info_from_l3_side(l3_side, gr, ch);
    iteration = iteration + 1;
    cod_info.part2_length = part2_length(fr_ps, gr, ch, l3_side);
    huff_bits = max_bits - cod_info.part2_length;

    % Call Inner Loop
    [bits xr_quant cod_info] = inner_loop( xr, huff_bits, cod_info);

    % Distortion Calculation
    xfsf = calc_noise(xr, xr_quant, cod_info, xfsf);

    % Save scaling factors
    scalesave_l(1:CBLIMIT) = scalefac.l(gr, ch, 1:CBLIMIT);
    for sfb = 1:SFB_SMAX
        scalesave_s(sfb, 1:3) = scalefac.s(gr, ch, sfb, 1:3);
    end
    save_preflag = cod_info.preflag;
    save_compress = cod_info.scalefac_compress;

    % Update the struct
    l3_side = put_gr_info_to_l3_side(cod_info, l3_side, gr, ch);
    % Preemphasis
    [xr, l3_xmin, l3_side] = preemphasis(xr, xfsf, l3_xmin, gr, ch,
                                         l3_side);

    % Amplify scalefactor bands
    [over xr l3_xmin scalefac] = amp_scalefac_bands(xr, xfsf, l3_xmin,
                                                    l3_side, scalefac, gr, ch, iteration);

    cod_info = get_gr_info_from_l3_side(l3_side, gr, ch);

```

Program 5.1: MATLAB Code for Outer loop. (*Continues.*)

```

% Checking for Termination
status = loop_break(scalefac, cod_info, gr, ch);

if status == 0
    [status cod_info] = scale_bitcount(scalefac, cod_info, gr, ch);
    % Update the struct
    l3_side = put_gr_info_to_l3_side(cod_info, l3_side, gr, ch);
end
end

```

Program 5.1: (*Continued.*) MATLAB Code for Outer loop.

5.4 INNER LOOP (RATE CONTROL)

The inner iteration loop does the actual quantization of the frequency domain data and in addition performs the rate control. The calculation of the codebook for each sub-region and computation of the number of bits needed to encode the values in the sub-regions are performed in this loop. The Huffman code tables assign shorter code words to smaller quantized values. If the total number of bits resulting from Huffman coding exceeds the number of bits available to encode one frame, the global gain is adjusted to result in a larger quantization step size. The overall steps involved in the inner loop can be seen from the block diagram in Figure 5.5 and the skeletal MATLAB code in Program 5.2.

5.4.1 NON-UNIFORM QUANTIZATION

Layer III uses a non-uniform mid-tread quantizer. Due to the required symmetry, a uniform quantizer with an even number of levels cannot have a reconstruction level at zero. This type of quantizer is referred to as mid-rise quantizer. On the other hand, a mid-tread quantizer that has an odd number of levels will have a reconstruction level at zero. The quantizer raises its input to the $3/4$ power before applying the quantizer. This is inverted in the decoder, where the quantized values are re-linearized by raising them to the $4/3$ power. This step ensures that the bigger values are quantized with less accuracy than small values, thereby increasing the signal-to-noise ratio at the low level input. The quantization of the vector of spectral values is performed according to the expression

$$ix(i) = \text{nint} \left(\left(\frac{|xr(i)|}{\sqrt[4]{2} \text{quantizerStepSize}} \right)^{0.75} - 0.0946 \right), \quad (5.5)$$

where xr indicates the frequency lines output from the MDCT module, $ix(i)$ is the quantized value of the input $xr(i)$, i is the index ranging from 0 to 575.

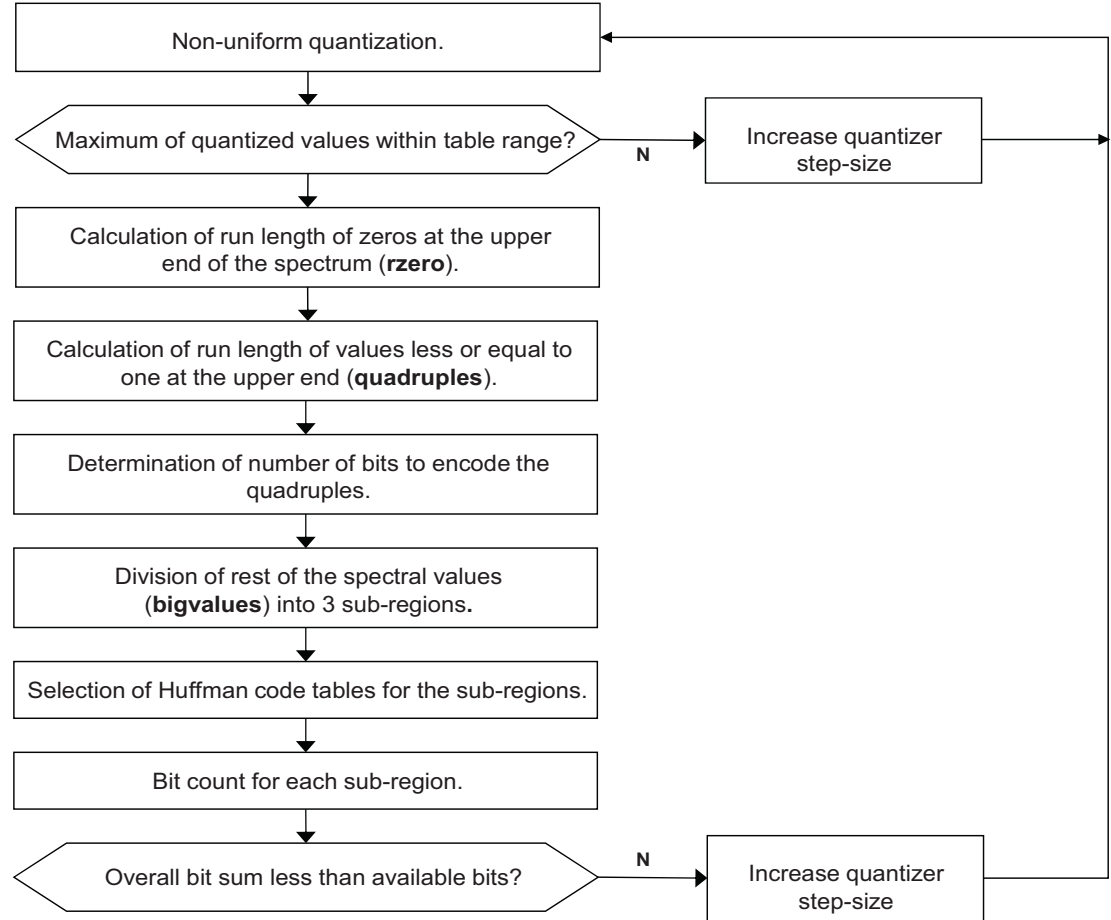


Figure 5.5: Implementation of the Inner Loop (Rate Control).

5.4.2 HUFFMAN CODING

The entropy coding of the quantized frequency lines is performed using Huffman coding. Huffman coding is an entropy encoding algorithm used for lossless data compression. Huffman coding uses a variable-length code table for encoding a source symbol where the code table is derived based on the estimated probability of occurrence for each possible value of the source symbol. In this algorithm, there are 32 Huffman code tables for coding the pairs of quantized values. The major differences between these tables are the maximum value that can be encoded and the signal statistics for which they are optimized. Huffman coding yields a variable length code, i.e., the length of code is varied based on the data statistics. There exists a limit on the maximum quantized value (of the frequency


```

% P 5.2 - Inner Loop

while bits > max_bits

    % dummy flag to simulate an exit controlled "do{}while" loop
    max_l3_enc = 9000;

    % Check if maximum of all quantized values is within the table range
    while max_l3_enc > (8191 + 14)
        cod_info.quantizerStepSize = cod_info.quantizerStepSize + 1.0;
        l3_enc = quantize(xrs, cod_info);
        max_l3_enc = max(l3_enc);
    end

    % Calculate run length of zeros, count1, big_values
    cod_info = calc_runlen(l3_enc, cod_info);

    % Count number of bits to code count1
    [bits cod_info] = count1_bitcount(l3_enc, cod_info);

    % SUBDIVIDE bigvalues
    cod_info = subdivide(cod_info);

    % Codebook Selection for subregion
    cod_info = bigv_tab_select(l3_enc, cod_info);

    % Count number of bits to code subregions
    bits = bits + bigv_bitcount(l3_enc, cod_info);

end

```

Program 5.2: MATLAB Code for Inner loop.

lines) allowed. This forces a constraint on the table size, if a table-lookup is used to requantize the quantized values. Hence, before any processing is carried out, the quantizer step size is increased only until the maximum of the quantized values is within the range of the largest Huffman code table.

The MP3 algorithm delimits the ordered frequency lines into three distinct regions defined by the variables: *rzero*, *count1* and *bigvalues*. In general, the ordering is by increasing frequency except for the short block mode. In the short block mode, there are three sets of window values in a subband and hence the ordering is done by frequency, then by window and then by scalefactor. The ordering is helpful because large values tend to be at the lower frequencies while long runs of zeros or very small values occur at high frequencies. Dividing the frequency lines into three different

regions enables the encoder to encode each region with a different set of Huffman tables specifically tuned for the statistics of that region.

The run length region, *rzero*, identifies and counts the pairs of spectral coefficients in the upper end of the spectrum that are quantized to zero. This region need not be encoded as its size can be easily deduced from the size of the other two regions. The run length of quadruples consists of spectral coefficients equal to -1, 0 or 1 is counted as the region *count1*. Two Huffman code tables are used to encode this region and they code four values at a time. The third region that covers all remaining coefficient values is called the *bigvalues*. The 30 Huffman tables for this region encode the coefficient values in pairs. Figure 5.6 (a) illustrates the unquantized MDCT for a granule of the audio data. Noted that the psychoacoustics module employs a short block for analysis of this granule and the coefficients are reordered for entropy coding. The rate control loop iteratively changes the quantizers until the target rate (128 kbps in this example) is achieved while maintaining the noise floor below the masking threshold. The resulting coefficients are shown in Figure 5.6 (b). It can be observed that some of the low-energy coefficients present at high frequencies are discarded in this process.

There are two Huffman codebooks with the corresponding code length tables to encode the *count1* region, where one Huffman word is used to encode one of the *count1* quadruples. The number of bits needed to encode this region is given by

$$bitsum_count1 = \min (sum_0, sum_1) ,$$

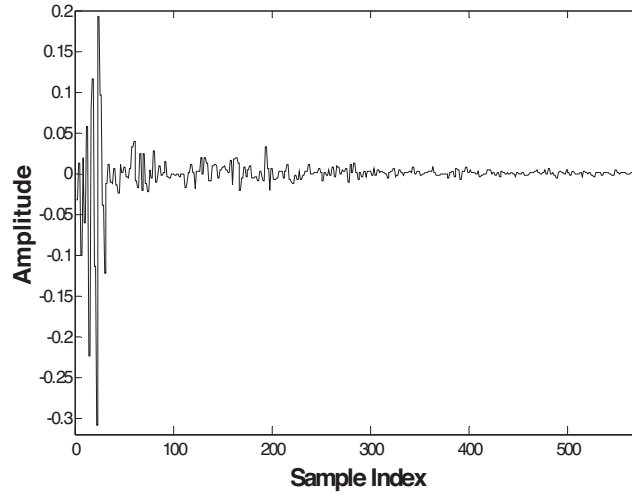
where

$$\begin{aligned} sum_0 &= \sum_{k=1}^{count1} ht_0 (ix(4k) + 2ix(4k+1) + 4ix(4k+2) + 8ix(4k+3)), \text{ and} \\ sum_1 &= \sum_{k=1}^{count1} ht_1 (ix(4k) + 2ix(4k+1) + 4ix(4k+2) + 8ix(4k+3)) . \end{aligned} \quad (5.6)$$

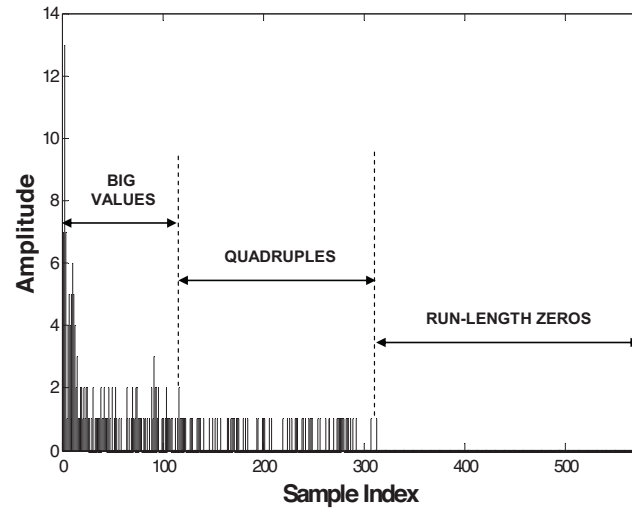
In (5.6) ht_0 and ht_1 refer to the two Huffman code tables that are used to encode the *count1* region and ix is the set of quantized samples that are encoded. We need to make sure that both the tables include the number of bits needed to encode the sign bits. The table that is selected to encode the *count1* region is transmitted by the parameter *count1_select*, which is 0 for ht_0 and 1 for ht_1 , respectively.

The pairs of quantized values that are not included in the *rzero* and *count1* regions are characterized as the *bigvalues*. The routine *SUBDIVIDE* further splits the scalefactor bands corresponding to the *bigvalues* into three regions. The number of scalefactor bands in the first and second regions is contained in *region0_count* and *region1_count*, respectively. The number of bands in the third region can be easily calculated from the *bigvalues*. When a large number of symbols are to be encoded, the construction of an optimal Huffman code table becomes very difficult. Hence, an *ESCAPE* value is used before coding the frequency lines to improve the coding efficiency.

The tables contain codes for values up to 15 and for values ≥ 16 , there are two tables provided, in which the largest value 15 is an *ESCAPE* character. In this case, the value 15 is encoded in an



(a)



(b)

Figure 5.6: (a) Unquantized MDCT coefficients for a short block. (b) MDCT coefficients (magnitude) quantized to meet target bit rate 128 kbps.

% P 5.3 - Calculate run length regions

```

rzero = 0;
cod_info.count1 = 0 ;

if cod_info.window_switching_flag & cod_info.block_type == 2
    % short blocks
    cod_info.count1 = 0;
    % big_values = 576/2 = 288
    cod_info.big_values = 288;
else
    for i = 576:-2:2
        if ix(i) == 0 & ix(i-1) == 0
            rzero = rzero + 1;
        else
            break
        end
    end

    % If we get all 576/2 = 288 runlength zeros, set big_values = 0
    if rzero == 288
        i = 0;
    end

    while i > 3
        if all(abs(ix(i:-1:i-3)) <= 1)
            cod_info.count1 = cod_info.count1 + 1;
            i = i - 4;
        else
            break;
        end
    end

    cod_info.big_values = i/2;
end

```

Program 5.3: MATLAB Code for calculating the run length regions.

```

% P 5.4 - Counting number of bits for count1

sum0 = 0; sum1 = 0;
i = cod_info.big_values*2 + 1;
for k = 1:cod_info.count1
    v = abs( ix(i:i+3) );
    i = i + 4;
    p = v(1) + v(2)*2 + v(3)*4 + v(4)*8 + 1;
    signbits = 0;
    % Sign bits are taken into account
    if v(1) ~= 0
        signbits = signbits + 1;
    end
    if v(2) ~= 0
        signbits = signbits + 1;
    end
    if v(3) ~= 0
        signbits = signbits + 1;
    end
    if v(4) ~= 0
        signbits = signbits + 1;
    end
    sum0 = sum0 + signbits;
    sum1 = sum1 + signbits;
    % ht - Huffman tables: 33, 34 are count1-tables
    sum0 = sum0 + ht{33}.hlen(p);
    sum1 = sum1 + ht{34}.hlen(p);
end
if sum0 < sum1
    bitsum_count1 = sum0;
    cod_info.count1table_select = 0;
else
    bitsum_count1 = sum1;
    cod_info.count1table_select = 1;
end

```

Program 5.4: MATLAB Code for counting number of bits for region *count1*.

additional word using a linear PCM code, which has a word length indicated by *linbits*. Figure 5.7 illustrates the main data organization and the Huffman encoding scheme. Thus, by partitioning the MDCT coefficients into regions and sub-regions, error propagation can be well controlled. Within the bitstream, the Huffman codes for the different values are ordered from low to high frequency. Figure 5.8 illustrates the bit usage for encoding different sub-regions in the inner loop. As it can be

72 5. BIT ALLOCATION, QUANTIZATION AND CODING

seen, the inner loop varies the quantization step and continues until the total bit usage is within the bit budget.

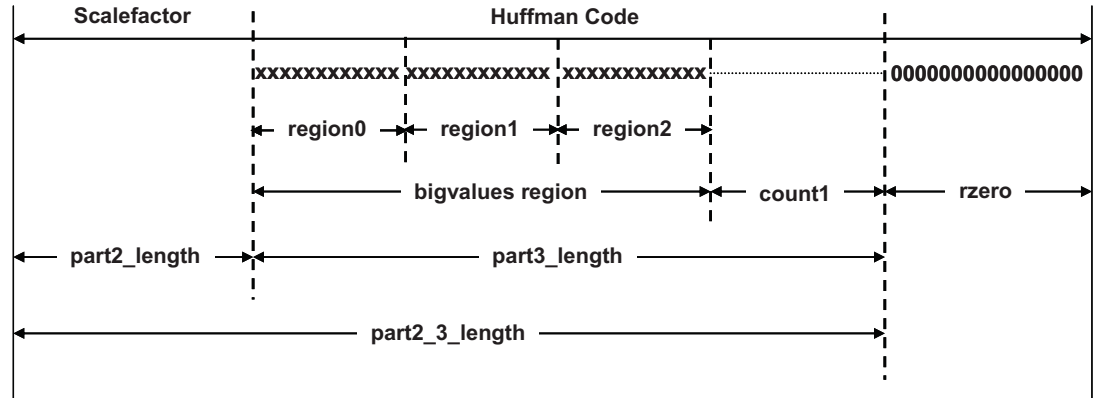


Figure 5.7: Illustration of main data organization and encoding scheme.

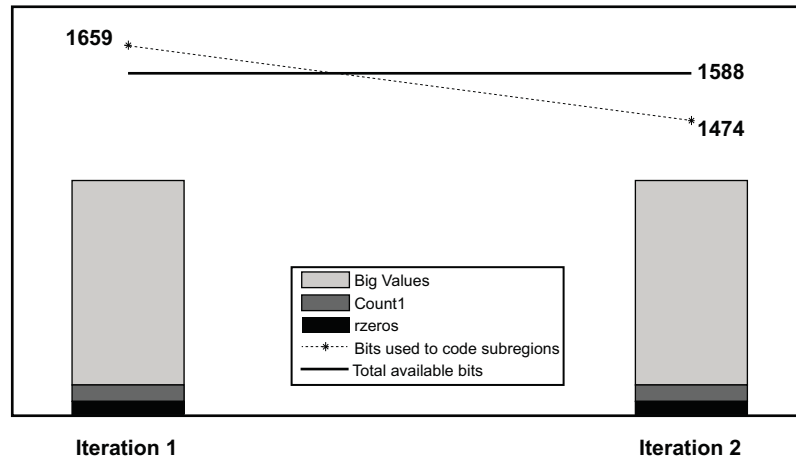


Figure 5.8: Illustration of bit usage in the inner loop.

To summarize, the *global_gain* and the scalefactors are determined by the inner and outer iteration loops. The outer loop shapes the noise spectrum by adjusting the scale factors, while the inner loop determines the *global_gain* that prevents the number of bits to encode the granule exceed *max_bits*.

Figure 5.9 illustrates an example case of the relationship between the number of bits required to encode a granule and the *global_gain*. During initialization, the maximum allowed distortion for

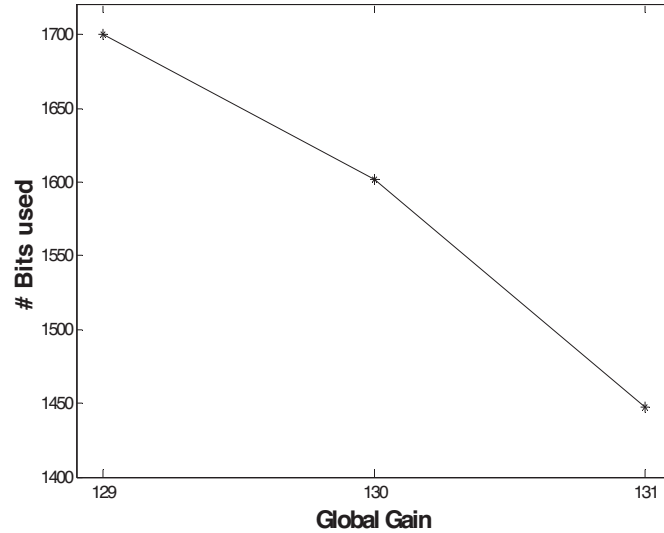


Figure 5.9: Illustration of relationship between *global_gain* and the number of bits required to encode a granule.

each scalefactor band, $xmin$, is determined using the SMR obtained from the psychoacoustic module. During the iteration loop, the scalefactors are increased in small amounts until the quantization noise is below $xmin$ or the scalefactors cannot be modified anymore. Figures 5.10 (a), (b) show the ratio of quantization noise over the allowed distortion at the beginning and end of the outer loop. The number inside the bars indicate the scalefactor values.

5.5 BITSTREAM FORMATTING

The design of the Layer III bitstream fits the encoder's time-varying demand on the coding bits. The encoded data representing the samples do not necessarily fit into a fixed length frame in the encoded bitstream. Instead, a bit reservoir is used with which the encoder can donate bits. Bits are donated when fewer than the average number of bits is required to encode a frame. Similarly, the encoder can borrow bits when needed. Figure 5.12 shows the arrangement of various bit fields in the Layer III bitstream. The bitstream format, which will be again discussed in detail in the decoder section, is described below:

Header: The first 32 bits constitute the header that contains the synchronization and state information. The header information is common to all layers. The layer of the MPEG-1

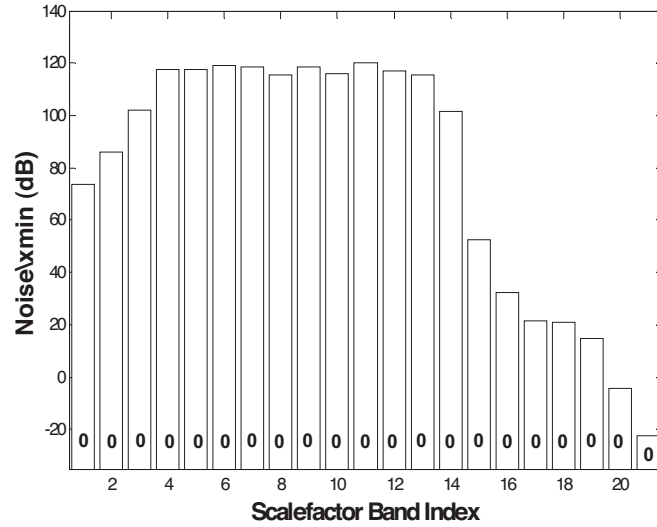


Figure 5.10: (a) Ratio of quantization noise over x_{min} at the start of the outer loop.

encoding algorithm used, bitrate, sampling frequency, mode, copyright, type of de-emphasis used are some of the information it specifies.

CRC: This is a 16 bit parity-check word that is used for optional error protection.

Side Information: Information in the bitstream used for controlling the decoder. Single channel audio frames contain 17 bytes (136 bits) of side information while dual channel audio frames contain 32 bytes (256 bits) of side information.

Main Data: This contains the Huffman encoded information of the audio samples. The *main_data_begin* pointer is used to determine the location of the first bit of the main data in a frame. It also contains the scale factor selection information and a variable, *part2_3_length*, specifying the number of main data bits used. In addition, it contains the information on the block type used in the frame and the Huffman table selection.

Ancillary Data: This is user defined information and need not be necessarily related to the audio stream itself. The number of ancillary bits available can be evaluated from the difference between the available number of bits and the number of bits actually used for the header, side information, error check and main audio data.

5.5.1 BIT RESERVOIR

The effect of pre-echo distortion was discussed in Chapter 3. Consider the case when a period of silence is followed by the sound of a percussive instrument. Such a transient sound requires an increase

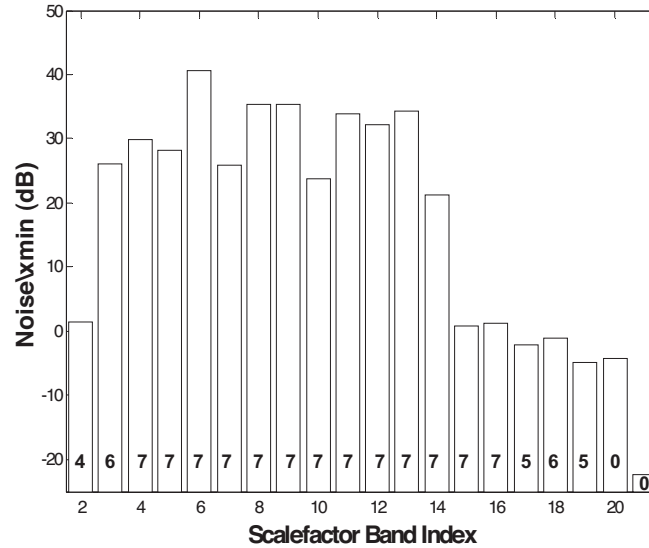


Figure 5.11: (b) Ratio of quantization noise over x_{min} at the end of the outer loop.

HEADER (32)	CRC (0,16)	SIDE INFORMATION (136,256)	MAIN DATA	ANCILLARY BITS
----------------	---------------	-------------------------------	-----------	-------------------

Figure 5.12: Frame Format for Layer III bitstream.

in the number of bits and can cause large quantization errors. These pre-echoes become distinctly audible at low bit rates. The effect of pre-echoes can be mitigated by temporal masking, provided if the transform block size is sufficiently small. Several methodologies have been proposed to mitigate the effect of pre-echoes [112, 113, 114, 115] and using a bit reservoir is one such technique [27, 102].

Although most algorithms are fixed rate, the instantaneous bit rates required to satisfy the masking thresholds in each frame is in general time varying. Bit reservoir is a short-term buffer technique that is used to enhance the coding efficiency of the Huffman coder. The idea is to store surplus bits during periods of low demand and then allocate bits from the reservoir suitably during periods of high demand. This results in a time-varying instantaneous bit rate but a fixed average bit rate. The bit reservoir can provide additional bits which may be used for coding the granule. The number of bits provided is determined within the iteration loops. The main data contains the encoded scale factor and Huffman encode frequency lines. The length of the main data depends on the bitrate. The length of the scalefactor part depends on whether the scalefactors are reused between granules and also on the window length (short or long). The bit reservoir allows unused

76 5. BIT ALLOCATION, QUANTIZATION AND CODING

main data bits stored in one frame to be used by up to two consecutive frames. Hence, the main data are not necessarily located adjacent to the side information in each frame as shown in Figure 5.13.

In the example illustrated in Figure 5.13, the main data in frame 1 uses bits from frame 0 and frame 1. Frame 2 also uses bits from frame 1 and hence the bits available for frame 2 are totally saved. The next frame, frame 3 requires a large number of bits and hence uses bits from frames 1, 2, and 3. Finally, the frame 4 uses the bits only from frame 4 and is hence self sufficient. Hence, it is observed that, when such a buffering technique is applied, the main data of a particular frame need not be necessarily located adjacent to its side info.

Estimation of Average Number of Bits (*mean_bits*)

The average number of bits per granule is calculated from the frame size. For example, consider the bitrate of 64 kbits/s at sampling rate 48 kHz.

Number of granules = 2.

Bits per frame = $64,000 \times \frac{1,152}{48,000} = 1,536$ bits.

The header is 4 bytes long and contains information about the layer, bitrate, sampling frequency and stereo mode. The side information takes 17 bytes (136 bits) in single channel mode and contains the necessary information to decode the main data such as Huffman table selection, scalefactors, requantization parameters and window selection.

Side info length + Header = $136 + 32 = 168$ bits.

\therefore Average number of bits = $\frac{(\text{Bits per frame} - \text{Side info length})}{\text{Number of granules}} = 684$ bits.

Bit Reservoir Functions

Bits are saved to the reservoir when fewer than the estimated average number of bits is used to encode the granule. If bits are saved for a frame, the size of the reservoir is increased accordingly. The number of bits available for the main data is calculated from the actual estimated threshold (perceptual entropy calculated in the psychoacoustics module), average number of bits and the actual content of the bit reservoir. Initially the maximum size of the reservoir is updated. If the number of bytes available to the inner iteration loop is not used for Huffman encoding, the number is added to the bit reservoir. If the bit reservoir is greater than zero, all bytes exceeding 8 times the maximum allowed content of the bit reservoir are made available for the main data. Program 5.4 illustrates the determination of maximum size of the reservoir.

The maximum bit allowance for each granule is based on the reservoir size at the beginning of that granule and perceptual entropy. First, the number of bits in addition to the average number of bits as demanded by the value of the perceptual entropy of the granule is determined ($\text{more_bits} = 3.1PE - \text{Average number of bits}$). If more_bits is greater than 100, then

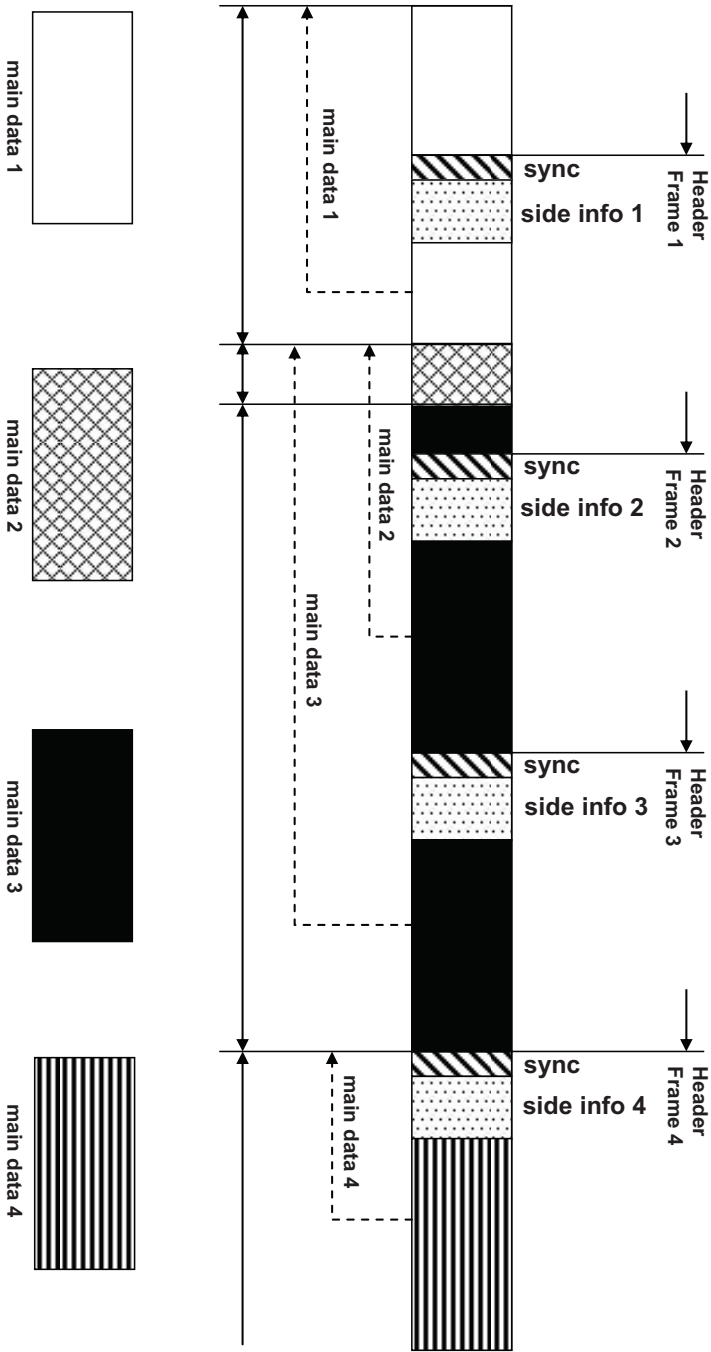


Figure 5.13: Illustration of main data buffer handling.

```

% P 5.5 - Bit Reservoir function

function [] = ResvFrameBegin(fr_ps, mean_bits, bitsPerFrame)
global ResvSize ResvMax;
global l3_side;

if fr_ps.header.version == 1
    mode_gr = 2;
    resvLimit = 4088; % main_data_begin has 9 bits in MPEG 1
else
    mode_gr = 1;
    resvLimit = 2040; % main_data_begin has 8 bits in MPEG 2
end;

% main_data_begin was set by the formatter to the expected value for
%the next call -- this should agree with our reservoir size
expectedResvSize = l3_side.main_data_begin * 8;
fullFrameBits = mean_bits*mode_gr;

% Determine maximum size of reservoir: ResvMax + frameLength <= 7680
if bitsPerFrame > 7680
    ResvMax = 0;
else
    ResvMax = 7680 - bitsPerFrame;
end

% Limit max size to resvLimit bits because main_data_begin cannot
% indicate a larger value
if ResvMax > resvLimit
    ResvMax = resvLimit;
end

```

Program 5.4: MATLAB Code to update maximum size of the reservoir.

$\max(\text{more_bits}/8, 6 * \text{Reservoir_Size})$ bytes are taken from the bit reservoir and made available for the main data, in addition to the average number of bits. Program 5.5 demonstrates this computation.

After the loop computations are complete, the number of bytes not used for the main data are added to the bit reservoir and hence the reservoir size is adjusted to reflect the granule's usage. Figure 5.14 illustrates the bit availability for coding each granule of the data frames. As described earlier, the total bit availability depends on the bit reservoir size at each stage.

% P 5.6 - Bit allocation for each granule

```

function [max_bits] = ResvMaxBits(fr_ps, pe, mean_bits)
global l3_side;
global ResvSize ResvMax;

mean_bits = mean_bits/fr_ps.stereo;
max_bits = fix(mean_bits);
if max_bits > 4095
    max_bits = 4095;
end
if ResvMax == 0
    return;
end

% Compute more_bits
more_bits = fix(pe * 3.1 - mean_bits);
add_bits = 0;

if more_bits > 100
    % check if frac is integer
    frac = fix((ResvSize*6)/10);

    if frac < more_bits
        add_bits = frac;
    else
        add_bits = more_bits;
    end
end

over_bits = fix(ResvSize - ((ResvMax*8)/10) - add_bits);

if over_bits > 0
    add_bits = add_bits + over_bits;
end

max_bits = max_bits + add_bits;

if max_bits > 4095
    max_bits = 4095;
end

```

Program 5.5: MATLAB Code to calculate bit allocation for each granule.

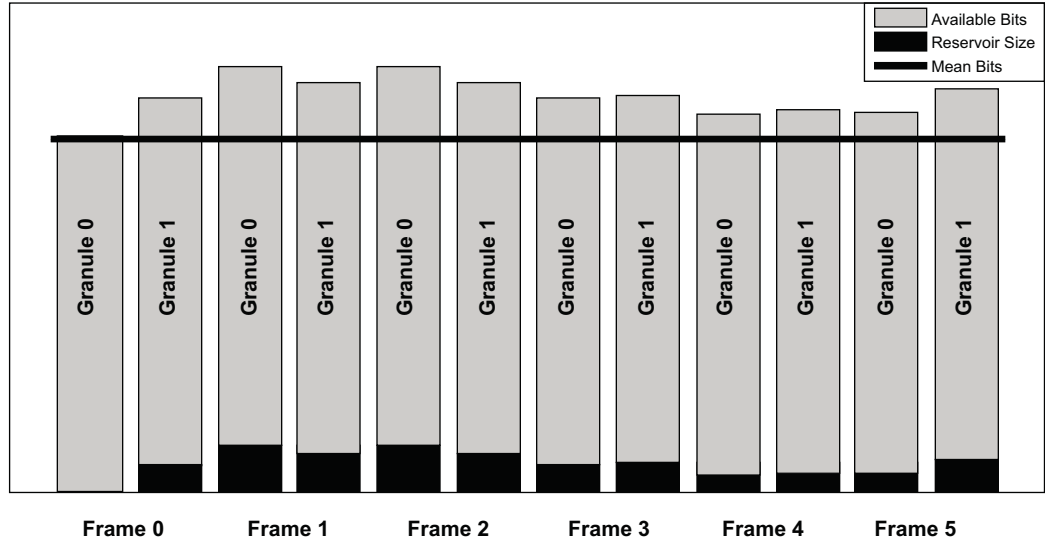


Figure 5.14: Illustration of total available bits to encode data frames.

5.6 SUMMARY

This chapter presented a detailed overview of the bit allocation strategies used in the MP3 algorithm and provides details on the bitstream formatting performed at the encoder. The use of short blocks in the MDCT module enables temporal pre-masking of pre-echoes during transients; longer blocks during steady-state periods improve coding gain, while also reducing side information and hence the overall bit rate. Bit allocation and quantization of the spectral lines are realized in a nested loop procedure that uses both lossy non-uniform quantization and lossless Huffman coding. The inner loop adjusts the non-uniform quantizer step sizes for each block until the number of bits required to encode the transform components falls within the bit budget. The outer loop evaluates the quality of the coded signal (analysis-by-synthesis) in terms of quantization noise relative to the JND thresholds. The use of a bit reservoir enables this perceptual coding algorithm to achieve a time-varying instantaneous bit rate but a fixed average bit rate. The reader is directed to [27, 89, 102, 103] for further details on bit allocation strategies and the bit reservoir usage.

CHAPTER 6

Decoder

The MP3 decoder operates on the encoded (received) bitstream to reconstruct the PCM audio data. The basic block diagram of the decoder is illustrated in Figure 6.1. This design shows the decoding of the single channel case and can be easily scaled for the stereo decoder. The contents of the Layer III Bitstream are organized into frames and each frame contains the necessary information to reconstruct the original PCM audio samples. A frame as explained in Section 5.5 consists of 5 parts: the header, CRC, side information, main data and ancillary data. In the following sections we will describe in detail the different steps involved in the decoding process.

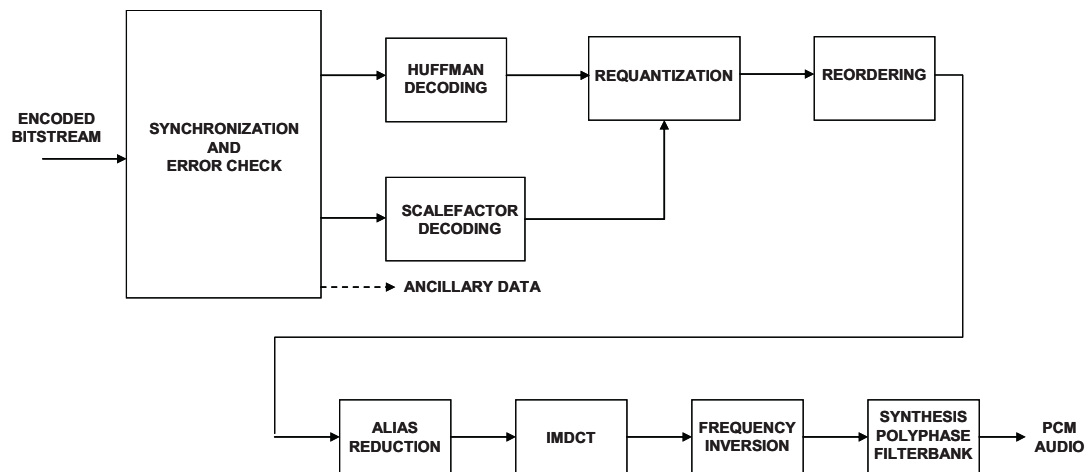


Figure 6.1: Block diagram of the MP3 decoder.

6.1 SYNCHRONIZATION AND ERROR CHECKING

This is the first unit of the decoder that receives the encoded bitstream, identifies and extracts the contents of the bitstream and transmits the information to the other blocks of the decoder. In addition to describing the operations performed in this module, a detailed analysis of the bitstream format is also presented in this section.

The illustration of the bitstream format of a Layer III frame can be found in Figure 5.5. The header part of the frame contains a 12-bit synchronization word and the system information.

82 6. DECODER

Each frame starts with this 12-bit synchronization word and the decoder detects the beginning of a new frame using this information. The remaining part of the header specifies the layer used in the frame, the bitrate used for transmission, the sampling frequency, whether the coded audio signal is a single channel or dual channel and some additional information related to the system. We cannot completely avoid spurious synchronization words from appearing in other parts of the frame. Therefore, in practice, to obtain a stable synchronization, all the bits in the header that are already known in advance by the decoder should be used as the synchronization word.

The MP3 encoder provides an optional error detection feature for each frame using a 16-bit CRC check word. The CRC word is placed in the bit positions 16 to 31 of the header and side information (see Figure 6.2). The encoder ensures that only the most sensitive information in each

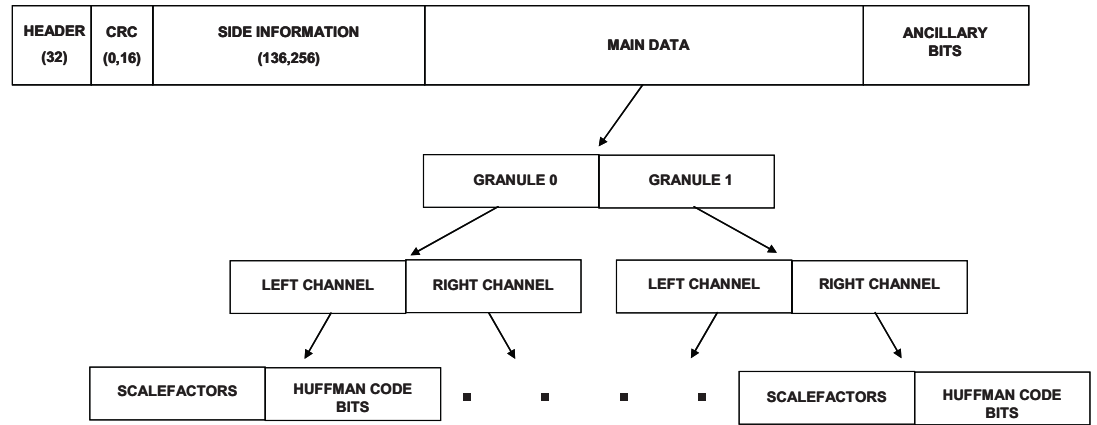


Figure 6.2: Organization of scalefactors and Huffman coded data.

frame is included in the CRC check word. This is due to that fact that any error in these values can corrupt the entire frame whereas errors in the main data will corrupt only a part of the frame. The Hamming distance of the CRC check word is $d = 4$, which detects up to 3 single bit errors or an error burst of up to 16 bits. If the *protection bit* in the header is equal to 0, it indicates that a CRC check word has been inserted in the bitstream by the encoder just after the header. The error detection method used has a generator polynomial of the form

$$G(x) = x^{16} + x^{15} + x^2 + 1. \quad (6.1)$$

The CRC check diagram is shown in Figure 6.3. The shift register has the initial state ‘1111 1111 1111’ and the data bits to be included to the CRC check are fed as inputs to the circuit. For every bit of the data, the shift register is shifted by one bit. After the shift operations are complete, the output bits $b_{15}...b_0$ constitute a word to be compared to the CRC check word in the bitstream. A transmission error in the protected field is detected if the words are not identical. If the frame is

corrupted, the common practice is to mute the actual frame or repeat the previous frame if it was free of errors.

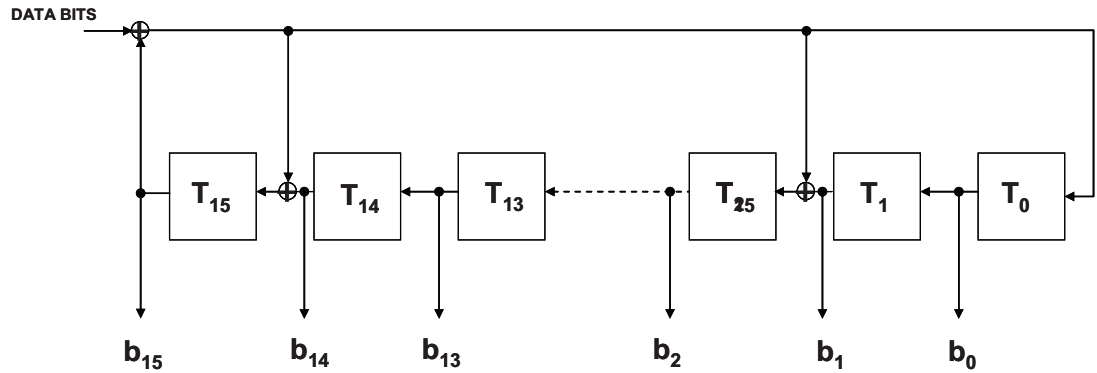


Figure 6.3: CRC check diagram.

The Synchronization and Error Checking module receives the incoming bitstream and identifies the location of each frame. If location of a frame is known, it is possible for the succeeding blocks in the decoder to extract all information contained in the frame. The decoder begins by searching for the 12-bit synchronization word in the bitstream. The validity of the word is tested by reading the header information and performing CRC check on the header and side information, if available. The synchronization word can be discarded if the CRC fails and search for a new synchronization word begins.

The MATLAB code section in Program 6.1 lists the steps involved in seeking the synchronization word in the bitstream. The code section in Program 6.2 shows the different parameters extracted from the header information in the bitstream.

6.2 DECODING SIDE INFORMATION

The side information section of the bitstream contains the necessary information to decode the main data. In particular, it contains the information pertaining to the Huffman tables to be used during the Huffman decoding process and the information for reconstruction of the scalefactors. The side information also contains the pointer to the beginning of main data, i.e., the location where the main data actually begins. For a single channel, the side information is 136 bits, while 256 bits are used for a dual channel (stereo).

6.2.1 EXTRACTING PARAMETERS FOR HUFFMAN DECODING

Though not shown explicitly in Figure 6.1, prior to performing Huffman decoding, the decoder needs to extract the necessary parameters from the side information. Therefore, the first step is to

% P 6.1 - Seek the synchronization word and place the bitstream pointer

```
function [bs, retval] = seek_sync(bs, sync, N)

global ALIGNING
maxi = round(2^N - 1);
aligning = rem(bs.totbit, ALIGNING);

if (aligning)
    [bs val] = getbits(bs, (ALIGNING-aligning));
end

[bs val] = getbits(bs, N);

while (bitand(val, maxi) ~= sync) & ~(bs.eobs)
    val = bitshift(val, ALIGNING);
    [bs v] = getbits(bs, ALIGNING);
    val = bitor(val, v);
end
if bs.eobs
    retval = 0;
else
    retval = 1;
end
```

Program 6.1: MATLAB Code to seek synchronization word.

identify where to find the Huffman code bits in the bitstream and also determine which Huffman tables need to be used. The decoder has to also find whether ESCAPE values are present in the Huffman code bits (refer to Section 5.4.2 for details). The other important task here is to ensure that irrespective of the number of frequency lines represented by the Huffman code bits, 576 frequency lines are generated when the decoding actually occurs. If this requirement is not satisfied, the frame is filled by zero padding.

6.2.2 EXTRACTING PARAMETERS FOR SCALEFACTOR DECODING

The main data section of the bitstream contains the encoded scalefactor values and the Huffman coded data. The beginning of the main data is determined by a pointer placed in the side information bits. The length of the pointer is 9 bits and the bit reservoir cannot exceed the value $8(2^9 - 1) = 4088$ bits. The variable *part2_length* contains the number of bits in the main data that are used for the scalefactors and the variable *part2_3_length* contains the total number of bits used by the scalefactors and the Huffman coded data. Hence, these values are used to determine the location of the scalefactors, main data and the ancillary data, if available. As seen in the encoder sections, the scale factor selection information needs to be considered during scalefactor decoding. For the bands

```

% P 6.2 - Decode the header information from the bitstream

function [bs, fr_ps] = decode_info(bs, fr_ps)

[bs fr_ps.header.version] = get1bit(bs);

[bs val] = getbits(bs,2);
fr_ps.header.lay = 4-val;

[bs val] = get1bit(bs);
fr_ps.header.error_protection = ~val; % error protect. TRUE/FALSE

[bs val] = getbits(bs,4);
fr_ps.header.bitrate_index = val;

[bs val] = getbits(bs,2);
fr_ps.header.sampling_frequency = val;

[bs val] = get1bit(bs);
fr_ps.header.padding = val;

[bs val] = get1bit(bs);
fr_ps.header.extension = val;

[bs val] = getbits(bs,2);
fr_ps.header.mode = val;
[bs val] = getbits(bs,2);
fr_ps.header.mode_ext = val;

[bs val] = get1bit(bs);
fr_ps.header.copyright = val;

[bs val] = get1bit(bs);
fr_ps.header.original = val;

[bs val] = getbits(bs,2);
fr_ps.header.emphasis = val;

```

Program 6.2: MATLAB Code to decode header information.

in which *scfsi* is set to '1', the scalefactors in the first granule are used for the second granule also. If it is set to '0', different scalefactors are to be used. The other important parameters that need to be known are the length of the scalefactors for the different bands (*slen1* and *slen2*). This can be determined from the variable *scalefac_compress* extracted from the side information bit sequence. Program 6.3 illustrates the extraction of some of the parameters from the side information bit sequence that will be used for scalefactor and Huffman decoding.

% P 6.3 - Extracting parameters from Side Information.

% These will be used for scalefactor and Huffman Decoding

```
[bs si.main_data_begin] = getbits(bs, 8);

if (stereo == 1)
    [bs si.private_bits] = getbits(bs, 1);
else
    [bs si.private_bits] = getbits(bs, 2);
end

for gr = 1:1
    for ch = 1:stereo
        [bs si.ch(ch).gr(gr).part2_3_length] = getbits(bs, 12);
        [bs si.ch(ch).gr(gr).big_values] = getbits(bs, 9);
        [bs si.ch(ch).gr(gr).global_gain] = getbits(bs, 8);
        [bs si.ch(ch).gr(gr).scalefac_compress] = getbits(bs, 9);
        [bs si.ch(ch).gr(gr).window_switching_flag] = getlbit(bs);

        if (si.ch(ch).gr(gr).window_switching_flag)
            [bs si.ch(ch).gr(gr).block_type] = getbits(bs, 2);
            [bs si.ch(ch).gr(gr).mixed_block_flag] = getlbit(bs);

            [bs si.ch(ch).gr(gr).table_select(1)] = getbits(bs, 5);
            [bs si.ch(ch).gr(gr).table_select(2)] = getbits(bs, 5);

            [bs si.ch(ch).gr(gr).subblock_gain(1)] = getbits(bs, 3);
            [bs si.ch(ch).gr(gr).subblock_gain(2)] = getbits(bs, 3);
            [bs si.ch(ch).gr(gr).subblock_gain(3)] = getbits(bs, 3);
```

Program 6.3: MATLAB Code to extract some of the parameters from side information.

6.3 SCALEFACTOR DECODING

The purpose of the scalefactor decoding block in Figure 6.1 is to decode the scalefactors, which form the first part of the main data. The parameters extracted from the side information are input to this block along with the encoded scalefactors. The decoded scalefactors are returned by this module,

which will in turn be used during the requantization process. The organization of scalefactors within a frame of data is shown in Figure 6.2. As seen in the figure, each frame of data is split into two granules and for each granule, the scalefactors and the Huffman coded data are presented for the two channels separately.

As described in Section 5.2, the scalefactors are decoded according to the *slen1* and *slen2* which themselves are determined from the values of *scalefac_compress*. The decoded values can be used as entries into a table or used to calculate the factors for each scalefactor band directly. When decoding the second granule, the *scfsi* has to be considered. As described in the previous section, the side information contains the information to decode the scalefactors. Each of the scalefactors transmitted, applies for one scalefactor band. The number of scalefactor bands and also the number of scalefactors transmitted depends on the type of the window used in the MDCT module of the encoder. A scalefactor band in Layer III indicates a set of frequency lines which are scaled by the same scalefactor. In general, there are 21 scalefactor bands at each sampling frequency in a granule for the case of long windows (types 0,1 or 3) and 12 bands in the case of short windows (type 2). Hence, there can be a maximum of 36 scalefactors for a channel within a granule and this situation arises when three short windows are used prior to the MDCT module. The total number of scalefactors for a channel must be known before the decoding process.

The MATLAB code sections Program 6.4 (a) and 6.4 (b) describe the steps to decode the scalefactors for the cases when short and long windows are used in the granule during encoding, respectively. In Layer III, the variable *scfsi_band* is used to apply the scalefactor selection informa-

```
% P 6.4(a) - Obtain scalefactors for the case of Short Windows (type 2)
for i = 1:2
    for sfb = sfbtable.s(i)+1:sfbtable.s(i+1)+1
        for window = 1:3
            scalefac(ch).s(window, sfb) = hgetbits(slen(i,
                gr_info.scalefac_compress+1));
        end
    end
end
sfb=13;
window = 1:3;
scalefac(ch).s(window, sfb) = 0;
```

Program 6.4: (a): MATLAB Code to get the scalefactors (Case for short windows).

tion, *scfsi*, to groups of scalefactor bands instead of a single scalefactor band. The application of scalefactors to the granules is controlled by the *scfsi*. For long windows with a total of 21 scalefactor bands, the variable *scfsi_band*, allows 4 groups of scalefactor bands, as can be seen in Program 6.4 (b). If the block type is switched to a short window for one of the granules, separate scalefactors are transmitted for each granule in the frame.

```

% P 6.4(b) - Obtain scalefactors for the case of Long Windows (types
0,1,or 3)
for i = 1:4
    if (si.ch(ch).scfsi(i) == 0) | (gr == 0+1)
        for sfb = sfbtable.l(i)+1:sfbtable.l(i+1)
            if i < 3
                scalefac(ch).l(sfb) = hgetbits(slen(1,
                                                gr_info.scalefac_compress+1));
            else
                scalefac(ch).l(sfb) = hgetbits(slen(2,
                                                gr_info.scalefac_compress+1));
            end
        end
    end
end
scalefac(ch).l(23) = 0;

```

Program 6.4: (b): MATLAB Code to obtain the scalefactors (Case for long windows).

6.4 HUFFMAN DECODING

The purpose of this block is to decode the Huffman code bits from the main data. Individual code words in the incoming Huffman code are not separated. Hence, a single code word in the middle of the code bits cannot be identified without actually decoding from the start of the code word. As a result, an error occurring somewhere in the Huffman code bits will make it very difficult to decode the remaining bits. All the necessary information bits to decode the Huffman coded bits are obtained from the side information as explained earlier. The *bigvalues* are decoded first using the tables indicated by *table_select*. The frequency lines in the regions *region0*, *region1* and *region2* are decoded in pairs until the number of pairs reaches the value specified by *bigvalues*. The value *count1* is the number of quadruples of decoded values and can be determined from *count1table_select*. The remaining Huffman code bits are then decoded using the table specified by *count1table_select*. The decoding is performed until all bits have been decoded or the quantized values for 576 frequency lines have been decoded. If there are more coded bits used than necessary to decode 576 lines, they are discarded.

As described in Section 5.4.2, ESCAPE values are introduced during Huffman coding at the encoder. When quantized frequency lines exceed the value 15, only the value 15 is encoded. The remaining information is represented in the Huffman code bits as an ESCAPE value. Hence, the Huffman decoder must detect the presence of ESCAPE values and sign bits too, in order to reconstruct the quantized frequency lines. Program 6.5 shows the MATLAB code section illustrates the overall Huffman decoding process.

```

% P 6.5 - Huffman Decoding

%Find region boundary for short block case

if si.ch(ch).gr(gr).window_switching_flag &
    si.ch(ch).gr(gr).block_type == 2
    % Region2
    region1Start = 36;
    region2Start = 576; % No Region2 for short block case
else
% Find region boundary for long block case
    region1Start = sfBandIndex(sfreq).l(si.ch(ch).gr(gr).region0_count
                                        +1+1);
    region2Start = sfBandIndex(sfreq).l(si.ch(ch).gr(gr).region0_count +
                                        si.ch(ch).gr(gr).region1_count +2+1);
end

grBits      = part2_start + si.ch(ch).gr(gr).part2_3_length;
currentBit = totbit;

% Read big_values
for i = 0:2:si.ch(ch).gr(gr).big_values*2-1
    if i < region1Start
        h = ht(si.ch(ch).gr(gr).table_select(1)+1);
    elseif i < region2Start
        h = ht(si.ch(ch).gr(gr).table_select(2)+1);
    else
        h = ht(si.ch(ch).gr(gr).table_select(3)+1);
    end

    [error x y v w] = huffman_decoder(h);
    is(floor(i/SSLIMIT)+1, rem(i, SSLIMIT)+1) = x;
    is(floor((i+1)/SSLIMIT)+1, rem((i+1), SSLIMIT) +1) = y;
end
grBits      = part2_start + si.ch(ch).gr(gr).part2_3_length;
currentBit = totbit;

```

Program 6.5: MATLAB Code to perform Huffman decoding. (*Continues.*)

```

% Read count1 area
h = ht(si.ch(ch).gr(gr).count1table_select+32+1);

i = si.ch(ch).gr(gr).big_values*2;
while (totbit < part2_start + si.ch(ch).gr(gr).part2_3_length ) & i <
                                            SSLIMIT*SBLIMIT

    [error x y v w] = huffman_decoder(h);
    is(floor(i/SSLIMIT)+1, rem(i, SSLIMIT)+1) = v;
    is(floor((i+1)/SSLIMIT)+1, rem((i+1), SSLIMIT)+1) = w;
    is(floor((i+2)/SSLIMIT)+1, rem((i+2), SSLIMIT)+1) = x;
    is(floor((i+3)/SSLIMIT)+1, rem((i+3), SSLIMIT)+1) = y;
    i = i + 4;
end

grBits      = part2_start + si.ch(ch).gr(gr).part2_3_length;
currentBit = totbit;

if totbit > part2_start + si.ch(ch).gr(gr).part2_3_length
    i = i- 4;
    rewindNbits(totbit - part2_start - si.ch(ch).gr(gr).part2_3_length);
end

% Dismiss stuffing Bits
grBits      = part2_start + si.ch(ch).gr(gr).part2_3_length;
currentBit = totbit;

if currentBit < grBits
    hgetbits(grBits - currentBit);
end

```

Program 6.5: (*Continued.*) MATLAB Code to perform Huffman decoding.

6.5 REQUANTIZATION

The primary purpose of the requantization is to generate a perceptually identical copy of the frequency lines as generated by the MDCT block in the encoder. This is done through a descaling operation which is based on the scaled quantized frequency lines obtained from the Huffman decoder module and the scalefactors extracted from the scalefactor decoder. The non-uniform quantizer at the encoder uses a power law and hence the frequency line, is , from the Huffman decoder are raised to the power $4/3$. This is the inverse power of that used in the encoder. This is either done using a table lookup or by explicitly calculating it. The calculation of the frequency lines belonging to one channel within a granule for the cases of long and short windows are shown in (6.2) and (6.3). Given the Huffman decoded values is_i , at buffer index i , the input to the synthesis filter bank, for the case of long

windows is given by

$$xr_i = \frac{\text{sign}(is_i) |is_i|^{\frac{4}{3}} 2^{\frac{1}{4}(\text{global_gain}[gr]-210)}}{2^{(\text{scalefac_multiplier}(\text{scalefac_l}[sfb][ch][gr]+\text{preflag}[gr].\text{pretab}[sfb]))}} \cdot \quad (6.2)$$

Similarly, the frequency channels within a granule for the case of short windows can be computed as

$$xr_i = \frac{\text{sign}(is_i) |is_i|^{\frac{4}{3}} 2^{\frac{1}{4}(\text{global_gain}[gr]-210-8.\text{subblock_gain}[\text{window}][gr])}}{2^{(\text{scalefac_multiplier}.\text{scalefac_s}[gr][ch][sfb][\text{window}])}} \cdot \quad (6.3)$$

It can be observed from Equations (6.2) and (6.3) that the parameters *global_gain* and *subblock_gain* (in the case of short windows) affect all the values within a time window. Note that the scalefactors and *preflag* adjust the gain within each scalefactor band. The *global_gain* defines the quantization step size used for one channel within a granule. The constant ‘210’ used in these two expressions is a system constant needed to scale the output appropriately. The scalefactors are logarithmically quantized in the encoder which can be identified using the *scalefac_scale* flag. The value for *scalefac_multiplier* is determined based on the value of this flag. The variables *pretab* and *preflag* are used only in the case of long windows and they were applied in the MDCT module at the encoder. The variable *pretab* specifies a single value per scalefactor band that is used for preemphasis. The parameter *preflag* that is added to the scalefactors is used for additional amplification of the higher frequencies. The scalefactors *scalefac_l* and *scalefac_s* for the cases of long and short windows, respectively, are determined by the scalefactor decoder block. In the case of short windows, a gain factor specified by the parameter *subblock_gain* can be obtained from the side information. Program 6.6 illustrates a part of the MATLAB code section for the requantization in the Layer III decoder.

6.6 REORDERING

After requantization, the frequency lines generated are not always ordered the same way. In the MDCT block at the encoder, if a long window (Figure 4.2) is used prior to transformation, the frequency lines generated would be ordered first by subband and then by frequency. However, using short windows would generate frequency lines first ordered by subband, then by window and at last by frequency. However, to improve the efficiency of the Huffman coding, the frequency lines for the short windows (three overlapped) are reordered first by subband, then by frequency and finally by the window type. The purpose of this reordering step is to detect if short windows have been applied to either of the subbands and reorder the values appropriately. The variables *window_switching_flag* and *block_type* obtained from the side information are used to determine if the reordering step is needed. The sample code section for performing reordering in the case of short blocks is shown in Program 6.7.

% P 6.6 - Requantization step

% This does not show the entire code for performing requantization. It illustrates few important steps.

% Compute overall (global) scaling.

```
xr(sb+1, ss+1) = 2.0^(0.25 * (gr_info.global_gain - 210.0));
```

% Do long/short dependent scaling operations.

```
if (gr_info.window_switching_flag & ...
    (((gr_info.block_type == 2) & (gr_info.mixed_block_flag == 0)) |
     ((gr_info.block_type == 2) & gr_info.mixed_block_flag & (sb >=
                                                                    2))))
    xr(sb+1, ss+1) = xr(sb+1, ss+1) * power(2.0, 0.25*-8.0*
        gr_info.subblock_gain(floor((sb*18+ss-cb_begin)/cb_width)+1));

    xr(sb+1, ss+1) = xr(sb+1, ss+1)* power(2.0, 0.25*-2.0*
        (1.0+gr_info.scalefac_scale)*scalefac(ch).s(floor((sb*18+ss-
        cb_begin)/cb_width)+1, cb));
```

```
else
```

% LONG block types 0,1,3 & 1st 2 subbands of switched blocks

```
    xr(sb+1, ss+1) = xr(sb+1, ss+1) * ...
        power(2.0, -0.5*(1.0+gr_info.scalefac_scale)* ...
            ( scalefac(ch).l(cb)+gr_info.preflag*pretab(cb) ));
```

```
end
```

% Scale quantized value

```
if is(sb+1, ss+1) < 0
    sign = 1;
```

```
else
    sign = 0;
```

```
end
```

```
xr(sb+1, ss+1) = xr(sb+1, ss+1) * ...
    power( double(abs(is(sb+1, ss+1))), double(4.0/3.0) );
```

```
if sign
```

```
    xr(sb+1, ss+1) = -xr(sb+1, ss+1);
```

```
end
```

Program 6.6: MATLAB Code illustrating the steps in requantization.

```

% P 6.7 - Reordering step

% This does not show the entire code for performing reordering. It
% illustrates a few important steps, when the window is purely short

sfb_start = 0;
sfb_lines = sfBandIndex(sfreq).s(2);

for sfb = 1:13
    for window = 0:2
        for freq = 1:sfb_lines
            src_line = sfb_start*3 + window*sfb_lines + freq;
            des_line = (sfb_start*3) + window + (freq*3);
            ro(ceil(des_line/SSLIMIT), rem(des_line, SSLIMIT)+1) = ..
                xr(ceil(src_line/SSLIMIT), rem(src_line, SSLIMIT)+1);
        end
    end
    sfb_start = sfBandIndex(sfreq).s(sfb);
    sfb_lines = sfBandIndex(sfreq).s(sfb+1) - sfb_start;
end

```

Program 6.7: MATLAB Code illustrating reordering.

6.7 ALIAS RECONSTRUCTION

In Section 4.2.2, it was described that an alias reduction was applied after performing the MDCT. In order to obtain a correction reconstruction of the PCM audio, the aliasing artifacts must be added to the signal again at the decoder. The alias reconstruction involves eight butterfly calculations for each subband as shown in Figure 6.5. Figure 6.4 illustrates the alias reduction butterfly for the decoder. The coefficients cs and ca can be obtained as described in Equation (4.3). The MATLAB code section illustrating the alias reconstruction process is shown in Program 6.8. It can be observed that the indices of the array xr indicate the frequency lines in a granule, arranged in the order of lowest to the highest frequency (zero being the lowest and 575 the highest). Similar to the case of the encoder, alias reduction is not applied for granules that use short blocks.

6.8 THE INVERSE MODIFIED DISCRETE COSINE TRANSFORM

The frequency lines from the alias reconstruction block are processed using the IMDCT block and mapped to the polyphase filter subband samples. The analytical expression of the IMDCT is given

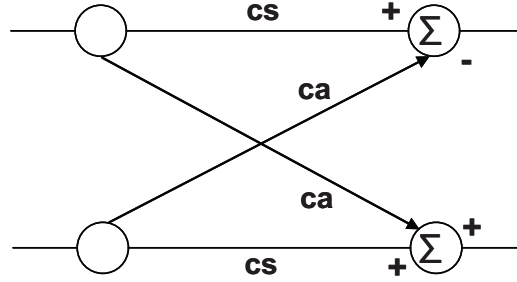


Figure 6.4: Alias reduction butterfly for the decoder.

by

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos\left(\frac{\pi}{2n} (2i + 1 + n/2) (2k + 1)\right), \text{ for } i = 0 \text{ to } n - 1, \quad (6.4)$$

where the frequency lines are denoted by X_k and n is 12 for short blocks and 36 for long blocks. The different types of windows used in the MDCT window switching are shown in Figure 4.2. For all block types other than short blocks, the result x_i is multiplied with the appropriate window. When the block type is *short*, the window is multiplied to each of the three blocks in the subband, and these windowed short blocks are then overlapped and concatenated. The result of these calculations is a vector for each subband. These vectors are then overlapped with half a window and added, resulting in 32 subband vectors, S_i . The first half of the block of 36 values is overlapped with the second half of the previous block. The second half of the current block is stored for use in the next block. The overlap-add operation used in the IMDCT module is illustrated in Figure 6.5. The MATLAB code for the implementation of the IMDCT module is illustrated in Program 6.9.

6.9 POLYPHASE SYNTHESIS FILTERBANK

In order to compensate for frequency inversions of the synthesis polyphase filter bank, every odd time sample of every odd subband is multiplied with -1. To summarize, the frequency lines are preprocessed by the alias reduction method described in Section 6.7 and fed into the IMDCT matrix (each 18 into one transform block). The first half of the output values are added to the stored overlap values from the last block. These values are the new output values and are input values for the polyphase filterbank. The second half of the output values is stored for overlap with the next data granule. For every second subband of the polyphase filterbank, every second input value is multiplied by -1 to correct for the frequency inversion of the polyphase filterbank. The algorithm for computing the polyphase filter bank is illustrated in Program 6.10. Once the samples from each of the 32 subbands have been calculated, they are applied to the synthesis polyphase filter bank.

```

% P 6.8 - Decoder anti-aliasing butterflies

% More specifically, they are the %'aliasing' butterflies, that
% recreate the aliasing caused by the analysis filterbank

function [hybridIn] = III_antialias(xr, gr_info)
% Globals for IMDCT
global ca cs;
global SBLIMIT SSLIMIT;

% clear all inputs
hybridIn = xr;
if gr_info.window_switching_flag & gr_info.block_type == 2 & ...
    ~gr_info.mixed_block_flag
    return;
end
if gr_info.window_switching_flag & gr_info.mixed_block_flag & ...
    gr_info.block_type == 2
    sblim = 1+1;
else
    sblim = SBLIMIT-1;
end
% 31 alias-reduction operations between each pair of sub-bands
% with 8 butterflies between each pair
ss = 0:7;
for sb = 1:sblim
    bu = xr(sb, 18-ss);
    bd = xr(sb+1, ss+1);
    hybridIn(sb, 18-ss) = bu.*cs(ss+1) - bd.*ca(ss+1);
    hybridIn(sb+1, ss+1) = bd.*cs(ss+1) + bu.*ca(ss+1);
end

```

Program 6.8: MATLAB Code illustrating alias reconstruction.

The synthesis polyphase filter bank transforms the 32 subband blocks of 18 samples each in each granule into 18 blocks of 32 PCM samples. In the synthesis operation, the 32 subband values are transformed to the vector \mathbf{V} , using a variant of the IMDCT (matrixing). Then, \mathbf{V} is pushed into a FIFO which can store 16 vectors. The vector \mathbf{U} is created from the alternate 32 component blocks in the FIFO and a window function \mathbf{D} is applied to \mathbf{U} to compute \mathbf{W} . The reconstructed samples are obtained from the \mathbf{W} vector by decomposing it into 16 blocks each containing 32 samples and summing these blocks. The window function \mathbf{D} , is illustrated in Figure 6.6 and the implementation of the synthesis polyphase filter bank is shown in Figure 6.7.

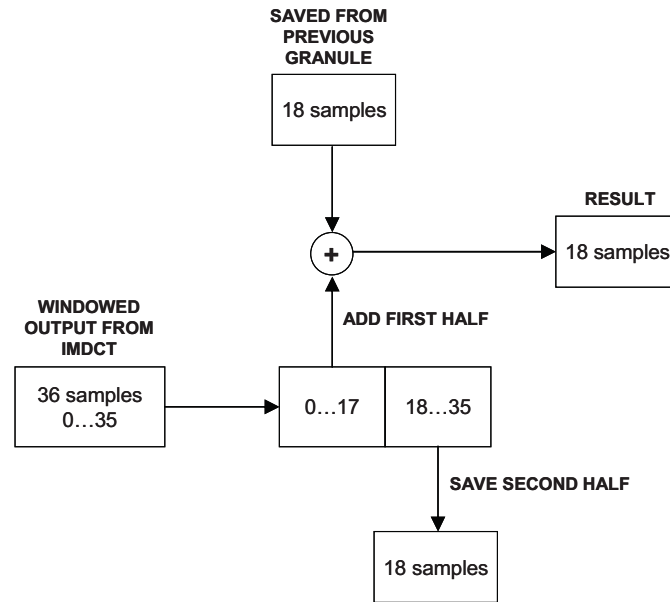


Figure 6.5: Overlap add operation used in the IMDCT module.

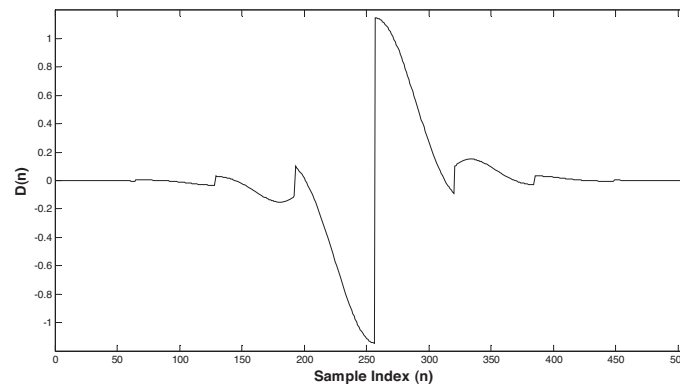


Figure 6.6: Window function, D , for the synthesis filter bank.

% P 6.9 - IMDCT: Inverse Modified Discrete Cosine Transform

```

function [out] = imdct(in, blk_type)
global win cos_i;
out = zeros(1, 36);
tmp = zeros(1, 12);

% type 2: short block
if blk_type == 2
    N = 12; m = 0:N/2-1;
    for i = 1:3
        for p = 0:N-1
            tmp(p+1) = (cos(pi/(2*N)*(2*p+1+N/2)*(2*m+1))*in(i+3*m)) *
                        win(blk_type+1, p+1);
        end
        out(6*i+1:6*i+N) = out(6*i+1:6*i+N) + tmp(1:N);
    end
else
    N = 36; m = 0:N/2-1;
    for p = 0:N-1
        out(p+1) = (cos_i(rem((2*p+1+N/2)*(2*m+1), 4*36)+1)*in(m+1)) *
                    win(blk_type+1, p+1);
    end
end
end

```

Program 6.9: MATLAB Code illustrating IMDCT.

6.10 SUMMARY

In this chapter, a detailed description of the design of a MPEG-1 Layer III audio decoder was presented. The decoder deciphers the received bitstream, restores the quantized subband values and reconstructs the audio signal from the subband values. The primary advantage of this decoder is the simplicity of its implementation, since it just reconstructs the bitstream and does not consider the psychoacoustics model or the quality of the encoded data. In fact, it is simple enough to allow single-chip, real-time decoder implementations. The encoded bitstream also supports an optional cyclic redundancy check (CRC) error detection code. Though the MPEG-1 Layer III coder is lossy, this algorithm aims at providing a perceptually lossless compression. The reader in need of additional details on the decoder implementation is referred to [116, 117, 118].

% P 6.10 - Subband synthesis

```

function [synth32, clip] = SubBandSynthesis(w, chn)

global SBLIMIT HAN_SIZE y D N
global SCALE
clip = 0;
synth32 = zeros(1, SBLIMIT);
y(chn, 2*HAN_SIZE:-1:65) = y(chn, 2*HAN_SIZE-64:-1:1);

for i = 1:64
    y(chn, i) = N(i, 1:SBLIMIT)*w;
end

i = 0:15;
i = bitshift(i+1, -1, 16)*64;
for j = 1:32
    k = j:32:512;
    l = k + i;
    synth32(j) = sum(D(k).*y(chn, l));
% Casting truncates towards zero for both positive and negative numbers
    if synth32(j) > 0
        foo = fix(synth32(j)*double(SCALE));
    else
        foo = fix(synth32(j)*double(SCALE));
    end

    if foo >= fix(SCALE)
        synth32(j) = fix(SCALE-1);
        clip = clip + 1;
    elseif foo < fix(-SCALE)
        synth32(j) = fix(-SCALE);
        clip = clip + 1;
    else
        synth32(j) = foo;
    end
end
end

```

Program 6.10: MATLAB Code illustrating polyphase synthesis filterbank.

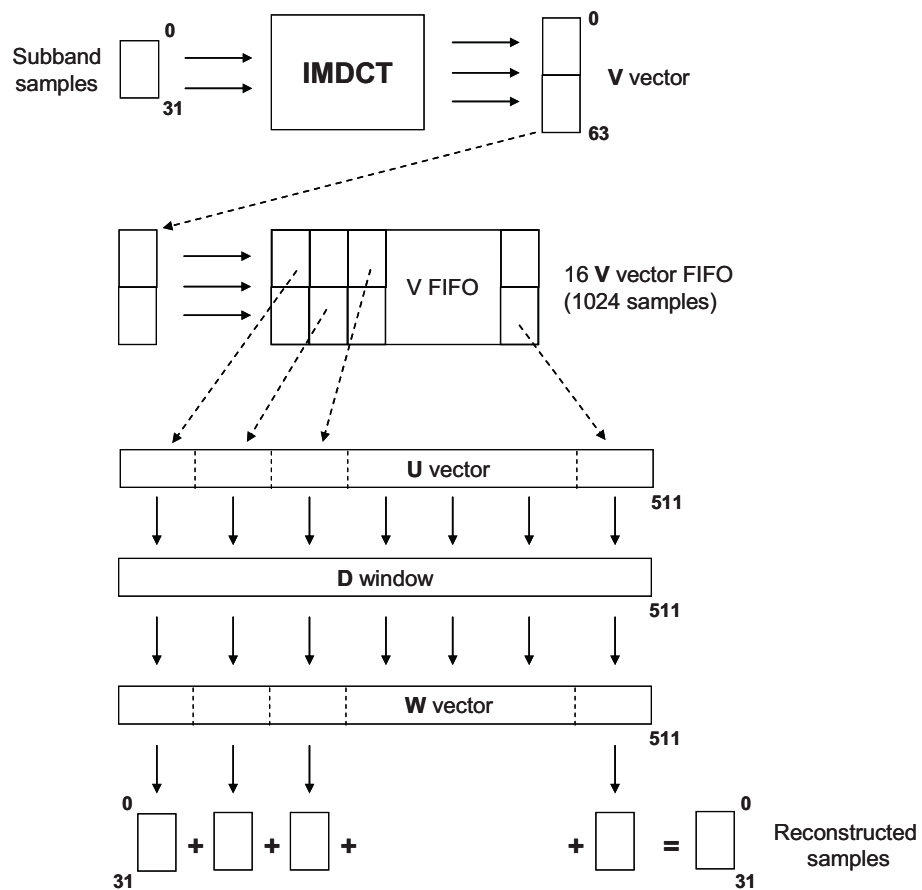


Figure 6.7: Polyphase implementation of the synthesis filter bank [78].

APPENDIX A

Complexity Profile of the MPEG-1 Layer III Algorithm

Analysis of the computational complexity of the MP3 encoder and decoder is essential in understanding the behavior of the algorithm. The complexity profile for any algorithm is highly dependent on the platform used for the implementation (C, MATLAB), operating system and the underlying hardware architecture. The choice of a suitable platform for implementing algorithms is based on different factors such as the speed of execution, availability of advanced libraries, visualization capabilities and flexibility for testing. In this chapter, the complexity analysis of both the C implementation (ISO dist-10) and the MATLAB implementation are presented. MATLAB is a high level double-precision floating-point engine that provides a highly advanced and flexible interface for testing and visualization. On the flip side, execution is slower than the more traditional compiled binaries and libraries.

As illustrated in Figure A.1 the MATLAB implementation of the MPEG-1 algorithm incurs much higher complexity at the encoder in comparison to the decoder. As described earlier, the decoder

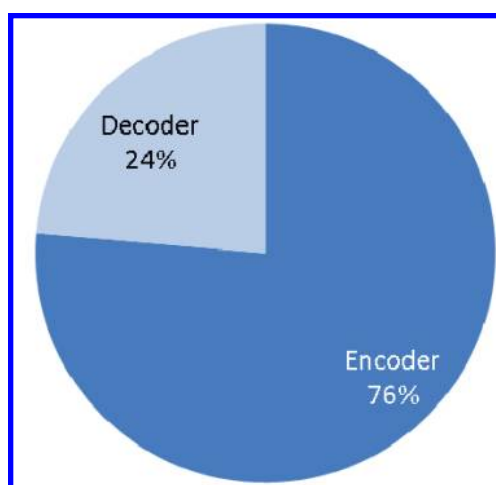
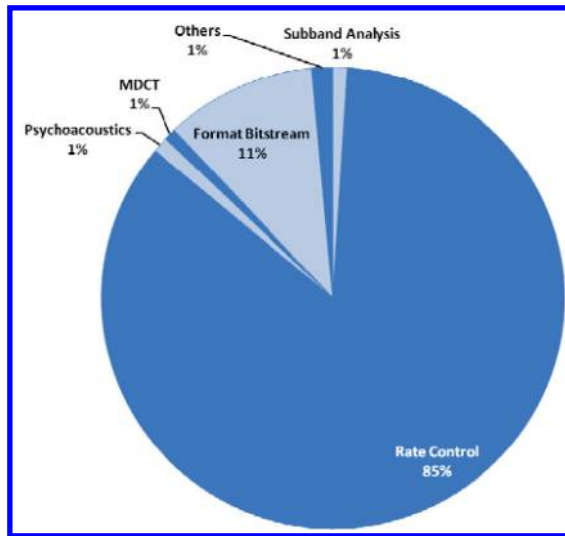


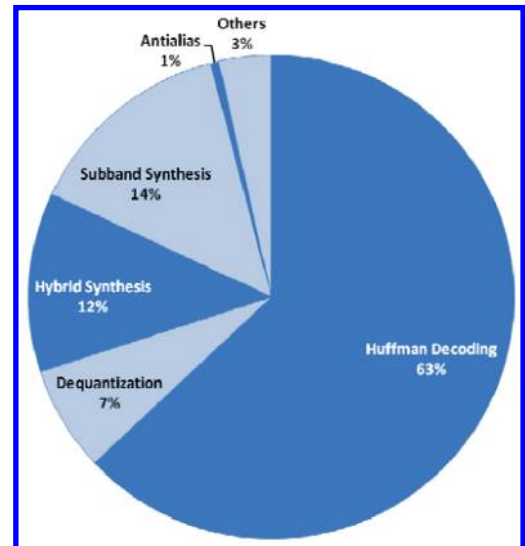
Figure A.1: Complexity Profile of MATLAB implementation of the MPEG-1 Layer III algorithm.

is simple enough to allow single-chip, real-time implementations. The complexity at the encoder can be attributed primarily to the extensive amount of bit-manipulation routines involved.

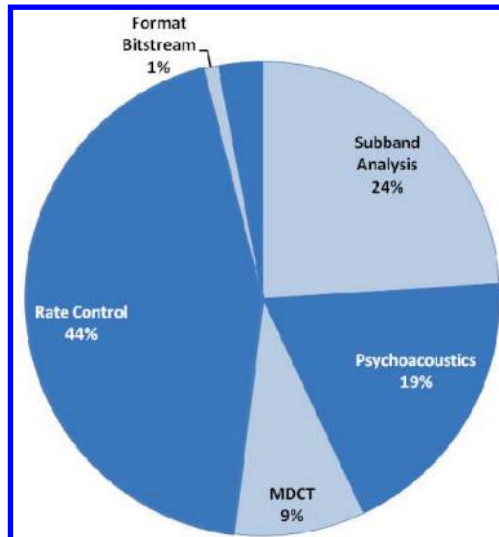
Figure A.2 illustrates the complexity profiles of the MATLAB and C implementations of the MP3 algorithm. Bit-manipulation operations, nested for-loops and dynamic memory allocation are very expensive in MATLAB. Therefore, the rate-control loop takes up most of the computational horsepower at the encoder. Most of the remaining processing bandwidth is used up in the bitstream formatting. At the decoder, Huffman decoding is costly in terms of bit-manipulation and, as expected, is the most expensive part of the algorithm. In contrast, the C implementation is very efficient in bit-wise processing and hence the complexity of the psychoacoustics model and the subband analysis are comparable to that of the rate control loop. Figure A.3 shows the complexity profiles of the rate control loop and the bitstream formatting module. The inner loop, which dynamically adjusts the non-uniform quantizer step sizes, and the binary step size search uses most of the computational power.



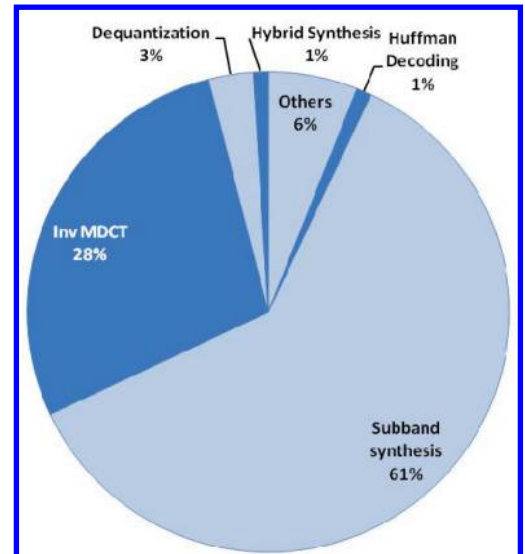
(a)



(b)



(c)



(d)

Figure A.2: Complexity Profile (a) Encoder implementation in MATLAB, (b) Decoder implementation in MATLAB, (c) ISO encoder implementation in C, (d) ISO decoder implementation in C.

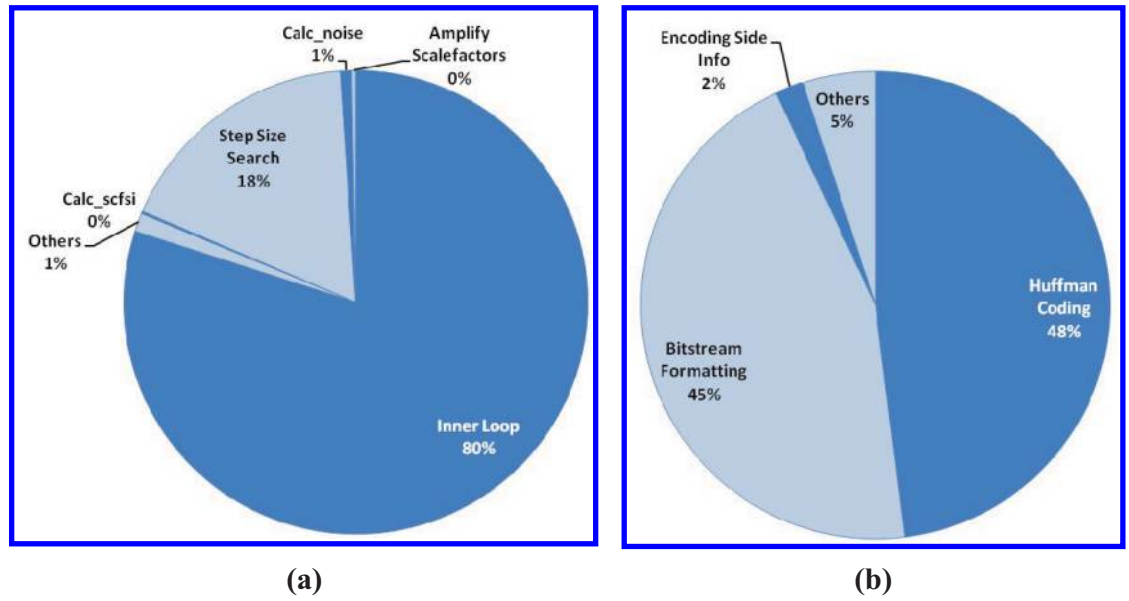


Figure A.3: Complexity Profile of the MATLAB implementation, (a) rate control module, (b) bitstream formatting.

Bibliography

- [1] J. Blauert, Spatial Hearing. The MIT Press: Cambridge, MA, 1974. Cited on page(s) 1
- [2] B. C. J. Moore and B. Glasberg, "Suggested Formulae for Calculating Auditory-Filter Bandwidths and Excitation Patterns," *J. Acous. Soc. Am.*, vol. 74, pp. 750–753, 1983.
DOI: [10.1121/1.389861](https://doi.org/10.1121/1.389861) Cited on page(s) 1
- [3] B. C. J. Moore, "Masking in the Human Auditory System," in *Collected Papers on Digital Audio Bit-Rate Reduction*, N. Gilchrist and C. Grewin, Eds., Aud. Eng. Soc., pp. 9–19, 1996. Cited on page(s) 1
- [4] B.C.J. Moore, An Introduction to the Psychology of Hearing, Academic Press, Fifth Edition, Jan. 2003. Cited on page(s) 1
- [5] R. E. Crochiere, et al., "Digital Coding of Speech in Sub-Bands," *Bell Sys. Tech. J.*, vol. 55, n. 8, pp. 1069–1085, Oct. 1976. DOI: [10.1109/ICASSP.1976.1170079](https://doi.org/10.1109/ICASSP.1976.1170079) Cited on page(s) 1
- [6] M. A. Krasner, "Digital Encoding of Speech and Audio Signals based on the Perceptual Requirements of the Auditory System," *Technical Report #535* of the MIT Lincoln Laboratory, Lexington 1979. Cited on page(s) 1
- [7] H.J. Nussbaumer, "Pseudo QMF Filter Bank," *IBM Tech. Disclosure Bulletin*, vol. 24, pp. 3081–3087, Nov. 1981. Cited on page(s) 1, 22
- [8] D. Krahe, "New Source Coding Method for High Quality Digital Audio Signals," *NTG Fachtagung Hoerrundfunk*, Mannheim, 1985. Cited on page(s) 1
- [9] E. Schroder, and W. Voessing, "High Quality Digital Audio Encoding with 3.0 Bits/Sample using Adaptive Transform Coding," *Proc. 80th Conv. Audio Eng. Soc.*, preprint #2321, Mar. 1986. Cited on page(s) 1
- [10] G. Theile, et al., "Low-Bit Rate Coding of High Quality Audio Signals," *Proc. 82nd Conv. Aud. Eng. Soc.*, preprint #2432, Mar. 1987. Cited on page(s) 1
- [11] DTS-HD Master Audio, Available at http://www.dts.com/DTS_Audio_Formats/DTS-HD_Master_Audio.aspx. Cited on page(s) 1, 3
- [12] MPEG-4 Audio Lossless Codec, Available at http://www.nue.tu-berlin.de/menue/forschung/projekte/beendete_projekte/mpeg-4_audio_lossless_coding_als/parameter/en/. Cited on page(s) 1, 4

106 BIBLIOGRAPHY

- [13] Windows Media Audio 9 Lossless Codec, Available at [http://msdn.microsoft.com/en-us/library/ff819508\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ff819508(v=VS.85).aspx). Cited on page(s) 1, 3
- [14] K. Brandenburg, "OCF - A New Coding Algorithm for High Quality Sound Signals," *Proc. IEEE ICASSP-87*, pp. 5.1.1–5.1.4, May 1987. DOI: [10.1109/ICASSP.1987.1169893](https://doi.org/10.1109/ICASSP.1987.1169893) Cited on page(s) 1
- [15] J. Johnston, "Perceptual Transform Coding of Wideband Stereo Signals," *Proc. IEEE ICASSP-89*, pp. 1993–1996, May 1989. DOI: [10.1109/ICASSP.1989.266849](https://doi.org/10.1109/ICASSP.1989.266849) Cited on page(s) 1
- [16] D. Sinha and A. Tewfik, "Low Bit Rate Transparent Audio Compression Using Adapted Wavelets," *IEEE Trans. Sig. Proc.*, pp. 3463–3479, Dec. 1993. DOI: [10.1109/78.258086](https://doi.org/10.1109/78.258086) Cited on page(s) 1
- [17] S. Boland and M. Deriche, "High Quality Audio Coding Using Multipulse LPC and Wavelet Decomposition," *Proc. IEEE ICASSP-95*, pp. 3067–3069, May 1995. DOI: [10.1109/ICASSP.1995.479493](https://doi.org/10.1109/ICASSP.1995.479493) Cited on page(s) 1
- [18] K. Hamdy, *et al.*, "Low Bit Rate High Quality Audio Coding with Combined Harmonic and Wavelet Representations," in *Proc. IEEE ICASSP-96*, pp. 1045–1048, May 1996. DOI: [10.1109/ICASSP.1996.543542](https://doi.org/10.1109/ICASSP.1996.543542) Cited on page(s) 1
- [19] K. Brandenburg and J. Johnston, "Second Generation Perceptual Audio Coding: The Hybrid Coder," *Proc. 88th Conv. Aud. Eng. Soc.*, preprint #2937, Mar. 1990. Cited on page(s) 1
- [20] Y.F. Dehery, *et al.* (1991), "A MUSICAM Source Codec for Digital Audio Broadcasting and Storage," *Proc. IEEE ICASSP-91*, pp. 3605–3608, May 1991. DOI: [10.1109/ICASSP.1991.151054](https://doi.org/10.1109/ICASSP.1991.151054) Cited on page(s) 1
- [21] M. Davis, "The AC-3 Multichannel Coder," *Proc. 95th Conv. Aud. Eng. Soc.*, preprint #3774, Oct. 1993. Cited on page(s) 1, 2
- [22] J. Princen and J. Johnston, "Audio Coding with Signal Adaptive Filterbanks," *Proc. IEEE ICASSP-95*, pp. 3071–3074, May 1995. DOI: [10.1109/ICASSP.1995.479494](https://doi.org/10.1109/ICASSP.1995.479494) Cited on page(s) 1
- [23] D. Sinha and J. Johnston, "Audio Compression at Low Bit Rates Using a Signal Adaptive Switched Filterbank," in *Proc. IEEE ICASSP-96*, pp. 1053–1056, May 1996. DOI: [10.1109/ICASSP.1996.543544](https://doi.org/10.1109/ICASSP.1996.543544) Cited on page(s) 1, 2
- [24] W-Y Chan and A. Gersho, "High Fidelity Audio Transform Coding with Vector Quantization," *Proc. IEEE ICASSP-90*, pp. 1109–1112, May 1990. DOI: [10.1109/ICASSP.1990.116130](https://doi.org/10.1109/ICASSP.1990.116130) Cited on page(s) 1

- [25] N. Iwakami, *et al.*, “High-Quality Audio-Coding at Less Than 64 kbit/s by Using Transform-Domain Weighted Interleave Vector Quantization (TWINVQ),” *Proc. IEEE ICASSP-95*, pp. 3095–3098, May. 1995. DOI: [10.1109/ICASSP.1995.479500](https://doi.org/10.1109/ICASSP.1995.479500) Cited on page(s) 1
- [26] D. Wiese and G. Stoll, “Bitrate Reduction of High Quality Audio Signals by Modeling the Ear’s Masking Thresholds,” *Proc. 89th Conv. Aud. Eng. Soc.*, preprint #2970, Sep. 1990. Cited on page(s) 1
- [27] ISO/IEC JTC1/SC29/WG11 MPEG, IS11172–3 “Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 Mbit/s, Part 3: Audio” 1992. (“MPEG-1”). Cited on page(s) 2, 13, 25, 46, 53, 57, 75, 80
- [28] ISO/IEC JTC1/SC29/WG11 MPEG, IS13818–3 “Information Technology - Generic Coding of Moving Pictures and Associated Audio, Part 3: Audio” 1994. (“MPEG-2 BC-LSF”). Cited on page(s) 2
- [29] G.C.P. Lokhoff, “Precision adaptive sub-band coding (PASC) for the digital compact cassette (DCC),” *IEEE Trans. Consumer Electronics*, pp. 784–789, Nov. 1992. DOI: [10.1109/30.179966](https://doi.org/10.1109/30.179966) Cited on page(s) 2
- [30] J. Johnston, *et al.*, “AT&T Perceptual Audio Coding (PAC),” in *Collected Papers on Digital Audio Bit-Rate Reduction*, N. Gilchrist and C. Grewin, Eds., *Aud. Eng. Soc.*, pp. 73–81, 1996. Cited on page(s) 2
- [31] G. Davidson and M. Bosi, “AC-2: High Quality Audio Coding for Broadcasting and Storage,” *Proc. 46th Annual Broadcast Eng. Conf.*, pp. 98–105, Apr. 1992. Cited on page(s) 2
- [32] L. Fielder and G. Davidson, “AC-2: A Family of Low Complexity Transform-Based Music Coders,” *Proc. 10th Conv. Aud. Eng. Soc.*, Sep. 1991. Cited on page(s) 2
- [33] L. Fielder, *et al.*, “AC-2 and AC-3: Low-Complexity Transform-Based Audio Coding,” in *Collected Papers on Digital Audio Bit-Rate Reduction*, N. Gilchrist and C. Grewin, Eds., *Aud. Eng. Soc.*, pp. 54–72, 1996. Cited on page(s) 2
- [34] T. Yoshida, “The Rewritable MiniDisc System,” *Proc. IEEE*, pp. 1492–1500, Oct. 1994. DOI: [10.1109/5.326407](https://doi.org/10.1109/5.326407) Cited on page(s) 2
- [35] K. Tsutsui, *et al.*, “ATRAC: Adaptive Transform Acoustic Coding for MiniDisc,” in *Collected Papers on Digital Audio Bit-Rate Reduction*, N. Gilchrist and C. Grewin, Eds., *Aud. Eng. Soc.*, pp. 95–101, 1996. Cited on page(s) 2
- [36] T. Robinson, “SHORTEN: Simple lossless and near-lossless waveform compression,” *Technical Report 156*, Engineering Department, Cambridge University, Dec. 1994. Cited on page(s) 2

108 BIBLIOGRAPHY

- [37] F. Wylie, "apt-X100: Low-Delay, Low-Bit-Rate-Subband ADPCM Digital Audio Coding," in *Collected Papers on Digital Audio Bit-Rate Reduction*, N. Gilchrist and C. Grewin, Eds., Aud. Eng. Soc., pp. 83–94, 1996. Cited on page(s) [2](#)
- [38] ISO/IEC JTC1/SC29/WG11 MPEG, Committee Draft 13818–7 "Generic Coding of Moving Pictures and Associated Audio: Audio (non backwards compatible coding, NBC)" 1996. ("MPEG-2 NBC/AAC"). Cited on page(s) [2](#)
- [39] M. Smyth, "White paper: An overview of the coherent acoustics coding system," June 1999 (available on the Internet – DTS web-page). Cited on page(s) [2](#)
- [40] P. G. Craven, M. Law, and J. Stuart, "Lossless compression using IIR prediction filters," *Proc. 102nd Conv. Aud. Eng. Soc.*, Munich, preprint #4415, March, 1997. Cited on page(s)
- [41] A. Wegener, "MUSICompress: Lossless, Low-MIPS Audio Compression in Software and Hardware," in *Proc. of the International Conference on Signal Processing Applications and Technology*, Sept. 1997. Cited on page(s) [2](#)
- [42] M. Purat, T. Liebchen, and P. Noll, "Lossless Transform Coding of Audio Signals," *Proc. 102nd Conv. Audio Eng. Soc.*, preprint #4414, March, 1997. Cited on page(s) [2](#)
- [43] M. Hans and R. W. Schafer, "AudioPaK – An integer arithmetic lossless audio codec," in *Proc. Data Compression Conf. Snowbird, UT*, 1998, p.550. DOI: [10.1109/DCC.1998.672286](https://doi.org/10.1109/DCC.1998.672286) Cited on page(s)
- [44] M. Hans and R. W. Schafer, "Lossless compression of digital audio," in *IEEE Sig. Proc. Mag.*, July 2001. DOI: [10.1109/79.939834](https://doi.org/10.1109/79.939834) Cited on page(s) [2](#)
- [45] ISO/IEC JTC1/SC29/WG11 (MPEG), International Standard ISO/IEC 14496–3: "Coding of Audio-Visual Objects – Part 3: Audio," 1999. ("MPEG-4"). Cited on page(s) [2](#)
- [46] M. A. Gerzon, J. R. Stuart, R. J. Wilson, P. G. Craven, and M. J. Law, "The MLP Lossless Compression System," in *AES 17th Int. Conf. High-Quality Audio Coding*, Florence, Italy, Sept. 1999, pp. 61–75. Cited on page(s) [2](#), [3](#)
- [47] ISO/IEC JTC1/SC29/WG11 (MPEG), International Standard ISO/IEC 14496–3 AMD-1: "Coding of Audio-Visual Objects – Part 3: Audio," 2000. ("MPEG-4 version-2"). Cited on page(s) [2](#)
- [48] ISO/IEC JTC1/SC29/WG11 MPEG, IS15938–4 "Multimedia Content Description Interface: Audio," International Standard, Sept. 2001. ("MPEG-7: Part-4"). Cited on page(s) [2](#), [3](#)
- [49] R. Geiger, T. Sporer, J. Koller, and K. Brandenburg, "Audio coding based on integer transforms," in *Proc. 111th Conv. Aud. Eng. Soc.*, New York, 2001. Cited on page(s) [2](#)

- [50] R. Geiger, J. Herre, J. Koller, and K. Brandenburg, "IntMDCT – A link between perceptual and lossless audio coding," *Proc. IEEE ICASSP-2002*, vol.2, pp. 1813–1816, May 2002. DOI: [10.1109/ICASSP.2002.5744976](https://doi.org/10.1109/ICASSP.2002.5744976) Cited on page(s) 2
- [51] D. Reefman and E. Janssen, "White paper on signal processing for SACD," Philips IP&S, [Online] Available: <http://www.superaudiocd.philips.com,clickDOWNLOADS>. Cited on page(s) 2, 3
- [52] E. Janssen and D. Reefman, "Super-Audio CD: An Introduction," in *IEEE Sig. Proc. Mag.*, pp. 83–90, July 2003. DOI: [10.1109/MSP.2003.1226728](https://doi.org/10.1109/MSP.2003.1226728) Cited on page(s) 2
- [53] MPEG Requirements Group, MPEG-21 Overview, ISO/MPEG N5231, Oct. 2002. Cited on page(s) 2, 3
- [54] T. Holman, "New factors in sound for cinema and television," in *J. Audio Eng. Soc.*, Vol. 39, pp. 529–539, July–August 1999. Cited on page(s) 2
- [55] M. Bosi, et al, "Aspects of current standardization activities for high-quality low rate multi-channel audio coding," *Proc. of IEEE Workshop on App. of Sig. Proc. to Audio and Acoustics*, New Paltz, New York, Oct 1993. DOI: [10.1109/ASPAA.1993.379999](https://doi.org/10.1109/ASPAA.1993.379999) Cited on page(s) 2
- [56] M. Bosi, "High-quality multichannel audio coding: Trends and Challenges," *J. Audio Eng. Soc.*, Vol. 48, No. 6, pp.588, 2000. Cited on page(s) 2
- [57] R. Koenen, "Overview of the MPEG-4 Standard," ISO/IEC JTC1/SC29/WG11 N2323, Jul. 1998. (<http://www.cselt.it/mpeg/standards/mpeg-4/mpeg-4.html>.) Cited on page(s) 2
- [58] R. Koenen, "Overview of the MPEG-4 Standard," ISO/IEC JTC1/SC29/WG11 N3156, Dec. 1999. (<http://www.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm>.) Cited on page(s) 2
- [59] E. Allamanche, R. Geiger, J. Herre, Th. Sporer, "MPEG-4 Low Delay Audio Coding Based on the AAC Codec," *107th AES Convention*, Munich 1999. Cited on page(s) 2
- [60] S. Wabnik, G. Schuller, J. Hirschfeld and U. Kramer, "Reduced Bit Rate Ultra Low Delay Audio Coding," *120th AES Convention*, May 2006, Paris, France. Cited on page(s) 2
- [61] U. Kramer, G. Schuller, S. Wabnik, J. Klier, J. Hirschfeld and, "Ultra Low Delay Audio Coding with Constant Bit Rate," *117th AES Convention*, Oct 2004, San Francisco. Cited on page(s) 2
- [62] ISO/IEC JTC1/SC29/WG11 MPEG, "MPEG-7 Applications Document," Doc. N3934, Jan. 2001. (Also available on-line: <http://www.cselt.it/mpeg>.) Cited on page(s) 3
- [63] Sony Adaptive Transform Acoustic Coding. Available at <http://www.sony.net/Products/ATRAC3/>. Cited on page(s) 3

110 BIBLIOGRAPHY

- [64] H. Purnhagen and N. Meine, "HILN-the MPEG-4 parametric audio coding tools," *Proceedings of ISCAS 2000 Geneva*, vol.3, no., pp.201–204 vol.3, 2000.
DOI: [10.1109/ISCAS.2000.856031](https://doi.org/10.1109/ISCAS.2000.856031) Cited on page(s) 3
- [65] P. G. Craven and M. Gerzon, "Lossless coding for Audio Discs," *J. Audio Eng. Soc.*, Vol. 44, no.9, pp. 706–720, Sept. 1996. Cited on page(s) 3
- [66] Apple Lossless Audio Codec, Available at <http://www.apple.com/itunes/features/#importing>. Cited on page(s) 3
- [67] Dolby TrueHD, Available at <http://www.dolby.com/consumer/understand/playback/dolby-truehd.html>. Cited on page(s) 3
- [68] "DTS-HD Audio: Consumer White Paper for Blu-ray Disc and HD DVD Applications", November 2006. Cited on page(s) 4
- [69] T. Cover and J. Thomas, *Elements of Information Theory*. John Wiley and Sons, Inc.: New York, 1991. Cited on page(s) 5
- [70] I. Witten, "Arithmetic Coding for Data Compression," *Comm. ACM*, vol. 30, n. 6, pp. 520–540, Jun. 1987. DOI: [10.1145/214762.214771](https://doi.org/10.1145/214762.214771) Cited on page(s) 5
- [71] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Info. Theory*, vol. IT-23, n. 3, pp. 337–343, May 1977. DOI: [10.1109/TIT.1977.1055714](https://doi.org/10.1109/TIT.1977.1055714) Cited on page(s) 5
- [72] A. Spanias, T. Painter and V. Atti, *Audio signal processing and coding*. Wiley-Interscience, John Wiley & Sons, Sep. 2007. Cited on page(s) 6, 7, 8, 9, 11, 13, 46
- [73] T. Painter and A. Spanias, "Perceptual coding of digital audio," *Proceedings of the IEEE*, vol.88, no.4, pp.451–515, Apr 2000. DOI: [10.1109/5.842996](https://doi.org/10.1109/5.842996) Cited on page(s) 13, 14, 25, 40, 58
- [74] P. P. Vaidyanathan, "Quadrature mirror filter banks, M-band extensions, and perfect-reconstruction techniques," *IEEE Acoust., Speech, Signal Processing Mag.*, pp. 4–20, Jul. 1987. DOI: [10.1109/MASSP.1987.1165589](https://doi.org/10.1109/MASSP.1987.1165589) Cited on page(s) 13
- [75] P. P. Vaidyanathan, "Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial," *Proc. IEEE*, vol. 78, pp. 56–93, Jan. 1990. DOI: [10.1109/5.52200](https://doi.org/10.1109/5.52200) Cited on page(s) 13
- [76] J. Johnston, S. Quackenbush, G. Davidson, K. Brandenburg, and J. Herre, "MPEG audio coding," in *Wavelet, Subband, and Block Transforms in Communications and Multimedia*, A. Akansu and M. Medley, Eds. Boston, MA: Kluwer Academic, 1999, ch. 7. Cited on page(s) 13

- [77] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1983. Cited on page(s) 13
- [78] S. Shlien, "Guide to MPEG-1 audio standard," *IEEE Trans. Broadcast.*, pp. 206–218, Dec. 1994. DOI: [10.1109/11.362938](https://doi.org/10.1109/11.362938) Cited on page(s) 22, 99
- [79] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice-Hall, 1993. Cited on page(s) 22
- [80] A. Akansu and M. J. T. Smith, Eds., *Subband and Wavelet Transforms, Design and Applications*. Norwell, MA: Kluwer Academic, 1996. Cited on page(s) 22
- [81] H. S. Malvar, *Signal Processing with Lapped Transforms*. Norwood, MA: Artech House, 1991. Cited on page(s) 22
- [82] K. Brandenburg, E. Eberlein, J. Herre, and B. Edler, "Comparison of filter banks for high quality audio coding," in *Proc. IEEE ISCAS*, 1992, pp. 1336–1339. DOI: [10.1109/ISCAS.1992.230257](https://doi.org/10.1109/ISCAS.1992.230257) Cited on page(s) 22
- [83] J. H. Rothweiler, "Polyphase quadrature filters: A new subband coding technique," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP-83)*, May 1983, pp. 1280–1283. DOI: [10.1109/ICASSP.1983.1172005](https://doi.org/10.1109/ICASSP.1983.1172005) Cited on page(s) 22
- [84] T. Ramstad, "Cosine modulated analysis–synthesis filter bank with critical sampling and perfect reconstruction," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP-91)*, May 1991, pp. 1789–1792. DOI: [10.1109/ICASSP.1991.150684](https://doi.org/10.1109/ICASSP.1991.150684) Cited on page(s) 22
- [85] H. Malvar, "Lapped transforms for efficient transform/subband coding," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 969–978, June 1990. DOI: [10.1109/29.56057](https://doi.org/10.1109/29.56057) Cited on page(s) 22
- [86] J. Johnston, "Transform Coding of Audio Signals Using Perceptual Noise Criteria," *IEEE J. Sel. Areas in Comm.*, pp. 314–323, Feb. 1988. DOI: [10.1109/49.608](https://doi.org/10.1109/49.608) Cited on page(s) 25, 42, 46
- [87] J. Johnston, "Estimation of Perceptual Entropy Using Noise Masking Criteria," *Proc. IEEE ICASSP-88*, pp. 2524–2527, May 1988. DOI: [10.1109/ICASSP.1988.197157](https://doi.org/10.1109/ICASSP.1988.197157) Cited on page(s) 25
- [88] E. Zwicker and H. Fastl, *Psychoacoustics Facts and Models*. Berlin, Germany: Springer-Verlag, 1990. Cited on page(s) 25, 40, 46
- [89] D. Pan, "A Tutorial on MPEG/Audio Compression," *IEEE Multimedia*, vol.2, no.2, pp.60–74, 1995. DOI: [10.1109/93.388209](https://doi.org/10.1109/93.388209) Cited on page(s) 27, 80

112 BIBLIOGRAPHY

- [90] B. Scharf, "Critical bands," in *Foundations of Modern Auditory Theory*. New York: Academic, 1970. Cited on page(s) [28](#)
- [91] R. Hellman, "Asymmetry of masking between noise and tone," *Percep. Psychophys.*, vol. 11, pp. 241–246, 1972. DOI: [10.3758/BF03206257](#) Cited on page(s) [28](#)
- [92] J. L. Hall, "Auditory psychophysics for coding applications," in *The Digital Signal Processing Handbook*, V. Madisetti and D. Williams, Eds. Boca Raton, FL: CRC Press, 1998, pp. 39.1–39.25. Cited on page(s) [29](#)
- [93] P. Noll, "Wideband speech and audio coding," *IEEE Commun. Mag.*, pp. 34–44, Nov. 1993. DOI: [10.1109/35.256878](#) Cited on page(s) [31](#), [58](#)
- [94] N. Jayant, J. D. Johnston, and R. Safranek, "Signal compression based on models of human perception," *Proc. IEEE*, vol. 81, pp. 1385–1422, Oct. 1993. DOI: [10.1109/5.241504](#) Cited on page(s) [31](#), [46](#)
- [95] M. Schroeder, B. S. Atal, and J. L. Hall, "Optimizing digital speech coders by exploiting masking properties of the human ear," *J. Acoust. Soc. Amer.*, pp. 1647–1652, Dec. 1979. DOI: [10.1121/1.383662](#) Cited on page(s) [31](#)
- [96] B. C. J. Moore, "Masking in the human auditory system," in *Collected Papers on Digital Audio Bit-Rate Reduction*, N. Gilchrist and C. Grewin, Eds., 1996, pp. 9–19. Cited on page(s) [35](#)
- [97] K. Brandenburg, "Perceptual coding of high quality digital audio," in *Applications of Digital Signal Processing to Audio and Acoustics*, M. Kahrs and K. Brandenburg, Eds. Boston, MA: Kluwer Academic, 1998. Cited on page(s) [36](#)
- [98] P. Papamichalis, "MPEG audio compression: Algorithms and implementation," in *Proc. DSP 95 Int. Conf. DSP*, June 1995, pp. 72–77. Cited on page(s) [36](#)
- [99] H. Fletcher, "Auditory patterns," *Rev. Mod. Phys.*, pp. 47–65, Jan. 1940. DOI: [10.1103/RevModPhys.12.47](#) Cited on page(s) [39](#)
- [100] E. Terhardt, "Calculating virtual pitch," *Hearing Res.*, vol. 1, pp. 155–182, 1979. DOI: [10.1016/0378-5955\(79\)90025-X](#) Cited on page(s) [39](#)
- [101] K. Tsutsui, "ATRAC (adaptive transform acoustic coding) and ATRAC 2," in *The Digital Signal Processing Handbook*, V. Madisetti and D. Williams, Eds. Boca Raton, FL: CRC Press, 1998, pp. 43.16–43.20. Cited on page(s) [42](#)
- [102] J. Johnston, D. Sinha, S. Dorward, and S. Quackenbush, "AT&T perceptual audio coding (PAC)," *Collected Papers on Digital Audio Bit-Rate Reduction*, pp. 73–81, 1996. Cited on page(s) [42](#), [75](#), [80](#)

- [103] D. Pan, "Digital audio compression," *Digital Tech. J.*, vol. 5, no. 2, pp. 28–40, 1993. Cited on page(s) [47](#), [80](#)
- [104] K. Konstantinides, "Fast subband filtering in MPEG audio coding," *IEEE Signal Processing Lett.*, vol. 1, pp. 26–28, Feb. 1994. DOI: [10.1109/97.300309](#) Cited on page(s) [47](#)
- [105] P. L. Chu, "Quadrature mirror filter design for an arbitrary number of equal bandwidth channels," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-33, pp. 203–218, Feb. 1985. DOI: [10.1109/TASSP.1985.1164529](#) Cited on page(s) [54](#)
- [106] J. Masson and Z. Picel, "Flexible design of computationally efficient nearly perfect QMF filter banks," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP-85)*, Mar. 1985, pp. 14.7.1–14.7.4. DOI: [10.1109/ICASSP.1985.1168361](#) Cited on page(s) [54](#)
- [107] R. Cox, "The design of uniformly and nonuniformly spaced pseudo QMF," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 1090–1096, Oct. 1986. DOI: [10.1109/TASSP.1986.1164945](#) Cited on page(s) [54](#)
- [108] R. Koilpillai and P. P. Vaidyanathan, "New results on cosine-modulated FIR filter banks satisfying perfect reconstruction," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP-91)*, May 1991, pp. 1793–1796. DOI: [10.1109/ICASSP.1991.150685](#) Cited on page(s) [54](#)
- [109] R. Koilpillai and P. P. Vaidyanathan, "Cosine-modulated FIR filter banks satisfying perfect reconstruction," *IEEE Trans. Signal Processing*, vol. SP-40, pp. 770–783, Apr. 1992. DOI: [10.1109/78.127951](#) Cited on page(s) [54](#)
- [110] J. Princen and A. Bradley, "Analysis/synthesis filter bank design based on time domain aliasing cancellation," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 1153–1161, Oct. 1986. DOI: [10.1109/TASSP.1986.1164954](#) Cited on page(s) [54](#)
- [111] G. Smart and A. Bradley, "Filter bank design based on time-domain aliasing cancellation with nonidentical windows," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP-94)*, May 1995, pp. III-185–III-188. DOI: [10.1109/ICASSP.1994.390059](#) Cited on page(s) [54](#)
- [112] M. Link, "An attack processing of audio signals for optimizing the temporal characteristics of a low bit-rate audio coding system," in *Proc. 95th Conv. Aud. Eng. Soc.*, 1993, preprint 3696. Cited on page(s) [75](#)
- [113] K. Akagiri, "Technical description of Sony preprocessing," SO/IEC JTC1/SC29/WG11 MPEG1, Input Doc., 1994. Cited on page(s) [75](#)
- [114] J. Herre and J. Johnston, "Enhancing the performance of perceptual audio coders by using temporal noise shaping (TNS)," in *Proc. 101st Conv. Aud. Eng. Soc.*, 1996, preprint 4384. Cited on page(s) [75](#)

114 BIBLIOGRAPHY

- [115] M. Bosi, K. Brandenburg, S. Quackenbush, L. Fielder, K. Akagiri, H. Fuchs, and M. Dietz, "MPEG-2 advanced audio coding," in *Proc. 101st Conv. Aud. Eng. Soc.*, 1996, preprint. Cited on page(s) [75](#)
- [116] T. Uzelac, M. Kovac, "A fast MPEG audio layer III software decoder," *Southeastcon '98. Proceedings. IEEE*, pp.43–46, 24–26 Apr 1998. DOI: [10.1109/SECON.1998.673287](#) Cited on page(s) [97](#)
- [117] T. Tsai, T. Chen; L. Chen, "An MPEG audio decoder chip," *Consumer Electronics, IEEE Transactions on*, vol.41, no.1, pp.89–96, Feb 1995. DOI: [10.1109/30.370314](#) Cited on page(s) [97](#)
- [118] G. Maturi, "Single chip MPEG audio decoder," *Consumer Electronics*. DOI: [10.1109/30.156706](#) Cited on page(s) [97](#)

Authors' Biographies

JAYARAMAN J. THIAGARAJAN

Jayaraman J. Thiagarajan completed his MS in Electrical Engineering and is currently a PhD candidate in the School of Electrical, Computer, and Energy Engineering at Arizona State University. His research interests are in the areas of DSP, audio processing and coding, sparse representations and computer vision. His works in sparse representations have been nominated for awards at IEEE DSP Workshop 2011 and Asilomar 2011. He has worked extensively with the MPEG-1 Layer-III audio coding standard and has created MATLAB modules for the various functions of the algorithm. In addition, he has developed LabVIEW interfaces for teaching speech coding algorithms and contributed to the Java-DSP software package.

ANDREAS SPANIAS

Andreas Spanias is a Professor in the School of Electrical, Computer, and Energy Engineering at Arizona State University (ASU). He is also the founder and director of the SenSIP Center and NSF NCSS I/UCRC site. His research interests are in the areas of adaptive signal processing, speech processing, and audio sensing. He and his student team developed the computer simulation software Java-DSP (JDSP– ISBN 0-9724984-0-0). He is author of two text books: *Audio Processing and Coding* by Wiley and *DSP: An Interactive Approach*. He served as Associate Editor of the IEEE Transactions on Signal Processing and as General Co-chair of IEEE ICASSP-99. He also served as the IEEE Signal Processing Vice-President for Conferences. Andreas Spanias is co-recipient of the 2002 IEEE Donald G. Fink paper prize award and was elected Fellow of the IEEE in 2003. He served as Distinguished lecturer for the IEEE Signal processing society in 2004.