UNIVERSITY OF MIAMI

REDUCING PERCEPTUAL AUDIO QUALITY LOSS IN TANDEM CODECS

By

Robert Steven Burke

A Research Project

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Master of Science

Coral Gables, Florida

May 2005

UNIVERSITY OF MIAMI

A research project submitted in partial fulfillment of
the requirements for the degree of
Master of Science

REDUCING PERCEPTUAL AUDIO QUALITY LOSS IN TANDEM CODECS

Robert Steven Burke

Approved:

_____	_____
Ken Pohlmann	Dr. Edward Asmus
Professor of Music Engineering	Associate Dean of Graduate Studies
_____	_____
Colby Leider	Dr. Manohar Murthi
Assistant Professor of Music Engineering	Professor of Electrical Engineering

BURKE, ROBERT (M.S., Music Engineering Technology)

Reducing Perceptual Audio Quality Loss in Tandem Codecs (May 2005)

Abstract of a Master's Research Project at the University of Miami.

Research project supervised by Professor Ken Pohlmann.
No. of pages in text: 101

Audio signals are often encoded in low bit-rate form. In many cases, the original uncompressed version is not available due to storage costs and/or bandwidth. Subsequent use of the low bit-rate source material then poses potential problems. Re-encoding of a previously encoded signal due to an even smaller bandwidth constraint, such as the consumer re-encoding the file for personal use, can increase the severity of audible artifacts that are inherent in all audio coders. Quality losses may be unavoidable in encoding algorithms that are optimized for a one-time pass of the original material. This research outlines a strategy and method for developing a means to mitigate this cascading error problem by using an "enabled" or "aware" coder that allows multiple-generational encoding with decreased accumulation of audible artifacts.

DEDICATION

To my parents, I love you both more than you will ever know.

ACKNOWLEDGMENT

To all the members of my committee, thank you for your unlimited patience, understanding, and thoughtfulness. I would like to especially thank Ken Pohlmann for your input and guidance throughout this journey. And to Dr. Asmus, thank you for all your help you have provided to me throughout my time at the University of Miami.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Perceptual audio coding, or low bit-rate coding, is a core technology for many applications such as broadcasting, DVD production, and consumer electronic devices such as personal MP3 players. As the laws of economics dictate, bandwidth efficiency will remain a concern in the audio industry.

Chapter 1 reviews the basics of audio compression technology, starting with some introductory concepts. Chapter 2 introduces the problem of the tandem coding errors, or audible artifacts that appear in audio material after many generations of coding, and discusses some more relevant details of audio coders which are of concern to the problem. Chapter 3 reviews some previous attempts to remedy this problem. In Chapter 4, a proposal to mitigate the cascading error problem is presented, and the experimental procedures are explained. In Chapter 5, the results will be presented and discussed. Chapter 6 concludes this research effort.

1.1 Analog Signals

The sounds heard in all of nature are analog in nature. That is, they are continuous in time. Our auditory systems process these waveforms in their natural analog forms. Figure 1.1 shows a continuous signal in the form of a sine wave. An audio signal is usually not stored in an analog format, however. In modern audio systems, there is an increasing use of digital technology. Once it is in digital format, an audio signal can be operated on in a digital sense; computing power can be brought to bear on certain tasks (e.g. audio coding). For these reasons, it is advantageous to store, retrieve, and process an audio signal in the digital domain.

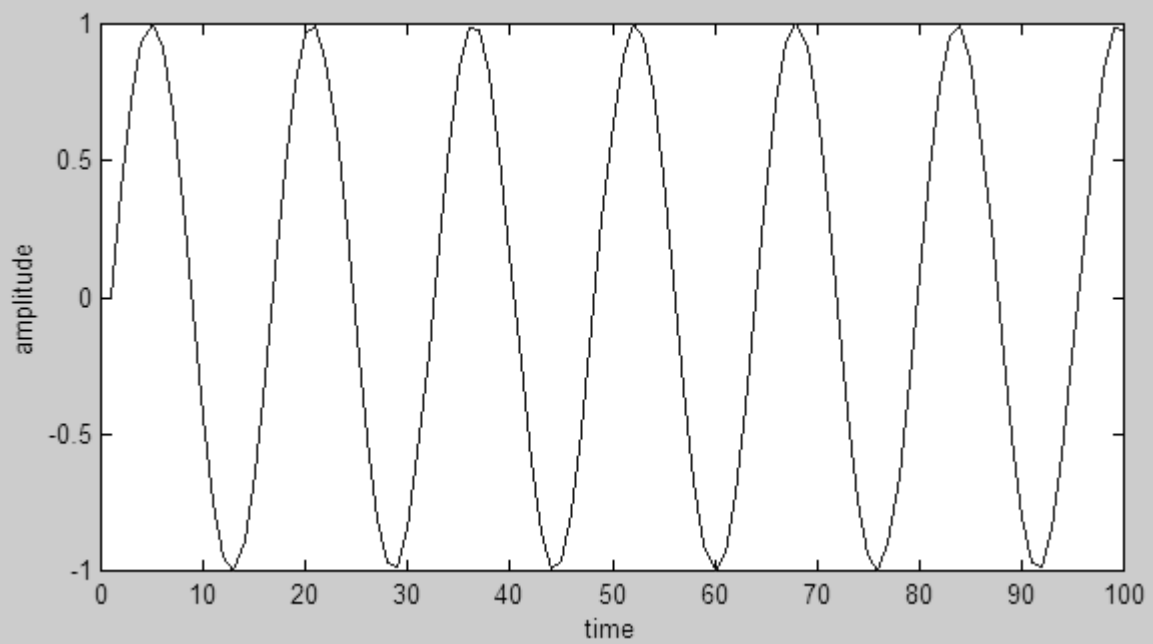

Figure 1.1 Analog waveform (sine wave)

1.2 Digital Representation – PCM data

Audio coding is a digital operation. It must therefore exist in the digital domain first before it can be processed. The standard digital coding format which has gained acceptance is the pulse code modulation (PCM) format. For a good description of PCM digital audio storage, see [19], but a brief description follows here.

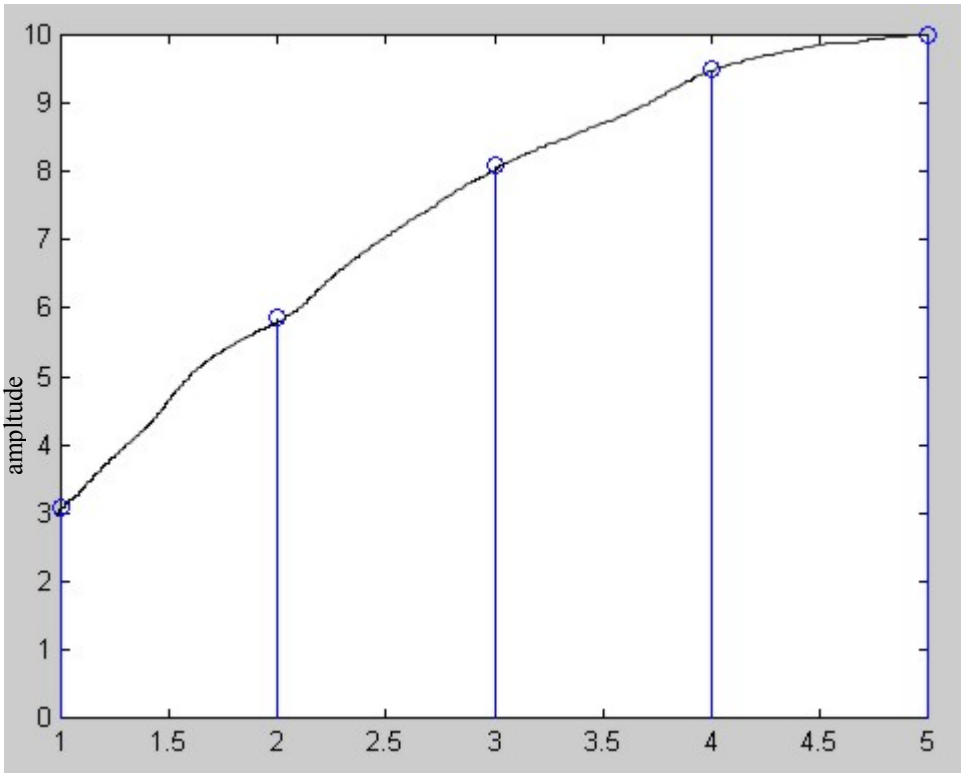

Figure 1.2 Sampling a waveform

An analog signal is sampled at regular time intervals such as in Figure 1.2. An amplitude measurement that could take on an infinite number of values exists at regularly spaced sample times; the sampling time is fixed. The next step is to approximate each amplitude by selecting the nearest of two digital approximations according to the 2^n levels available, with n being the number of bits chosen in the system (for example, n=16 for compact disc storage). The amplitude is said to be "quantized", and the audio signal will now contain what is called "quantization noise" – noise added to the signal due to the process of quantizing the amplitude at each sample position. The final step in PCM conversion is to convert the quantized amplitude into a binary format using only 1's and 0's to transmit the information. For example, if n=4, (allowing 16 different quantization levels), the amplitude of 8 at sample time t=3 in Figure 1.2 might be coded as 1000 (assuming an unsigned number format is used) as shown in Figure 1.3.

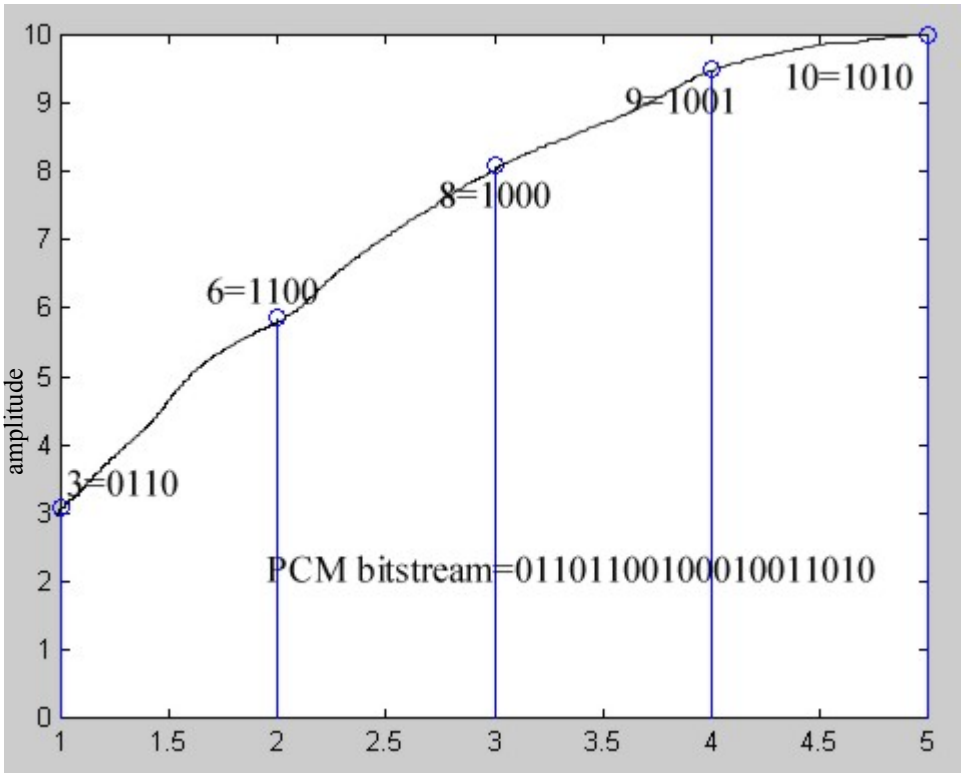

Figure 1.3 PCM bit stream formatting (4 bits/sample)

1.3 PCM Limitations

The PCM format has served its purpose well. It is capable of reasonably high fidelity. It is robust in the presence of channel noise and it is simple to transmit, having to transmit only one of two values for each symbol. For these reason and others, it has become the de facto standard for digital audio recording and use.

However, it is also a very greedy format in terms of bandwidth. Data rate requirements for stereo are PCM very high. Using the CD-Audio format for example, requires 16 bits/sample × 44,100 samples/sec × 2 channels = 1.41 Mbits/sec. With multi-channel sources, it can be seen how the bandwidth increases if PCM audio is chosen as the format. Also, for many consumer devices, 1.41Mbits/sec or higher (when

considering error correction and other overhead) is overly consuming of the limited bandwidth available. It becomes clear that storage requirements and transmission problems can be improved if data reduction is used to minimize the size of the digital file.

1.4 Conventional Data Rate Reduction

Crude methods for reducing the data requirements for PCM data storage are apparent (assuming a CD-Audio format):

- Reduce the word length. For example, instead of using 16-bits to represent each audio sample, use 12-, 10-, or 8-bit samples.
- Reduce the sample rate. Instead of a 44.1 kHz sample rate, use a 22.05 or 11.025 kHz sampling rate. This will reduce the total number of values stored.

For example, if 8 bits per sample and a 22.05 kHz sampling rate is chosen, the overall data rate can be reduced by a factor of four. In practice, however, neither of these strategies is useful, since audio quality is impacted greatly when using either or both of these methods aggressively enough to have any benefits. If the word length is truncated, the quality of each digital sample is limited, which in turn limits the overall perceptual quality of the audio. If sampling frequency is lowered, higher frequencies are unable to be coded due to the Nyquist limit that states one can only represent frequencies equal to one-half of the sampling rate chosen. It is desired to reduce the data rate without impacting audio quality, so other options must be looked to for reducing data rate.

1.5 Data Compression Techniques - Lossless vs. Lossy

In general, there are two types of data reduction: lossless and lossy compression [21]. Lossless compression retains all the information in a given signal, i.e. no

information is lost at all upon reconstruction of the signal. In contrast, a lossy scheme removes information from the original signal.

Lossless compression relies on analysis and removal of *statistical* redundancy of the signal. For instance, instead of coding a sequence of 95 consecutive 4-bit values of 1010, a lossless coder could send some code word such as (1010, 95) to indicate 95 consecutive running samples of the value 1010. This is called run length coding [21]. Lossless compression schemes based on this and other techniques such as pkzip, ARJ, and others are used quite extensively in compressing computer data. For audio data, lossless compression can take advantage of statistical redundancy as well. Because consecutive audio samples are relatively close to each other in value, a predictive model could use the last sample value as a metric to re-create the next sample. The difference between the predictor's output and the actual sample value may be encoded, saving bandwidth. Other techniques exist as well. A few of the more common lossless coding schemes for audio data are FLAC, WavPack, AudioZip, and Meridian Lossless Packing [1]. Lossless coding occupies a useful niche in the world of audio coding; however, on the whole, music signals rarely exhibit the type of structure necessary to achieve aggressive compression ratios using lossless techniques alone.

Lossy reduction schemes, on the other hand, do not achieve a perfect reconstruction of the original signal. As a rule, they are capable of more aggressive reduction ratios than lossless coders because of this degree of freedom, or relaxing the requirement of perfect reconstruction of the original signal. Lossy reduction relies on the principle "getting close enough" in order to achieve results. If the imperfections can go unnoticed by the detection mechanism (e.g., the ear), then a listener may not notice that

the signal is not exactly as it originally sounded. Lossy reduction can take advantage of *perceptual* redundancy. Lossy reduction techniques are the backbone of most low bit-rate encoders in use today. To exploit this *perceptual* method, an accurate model of how humans perceive sound is needed.

1.6 Description of the Human Auditory System

Perceptual audio coders are able to achieve high coding gain with minimal perceptual loss by accounting for inherent weaknesses of the human auditory system (HAS). Original recorded material such as an audio CD source bitstream has some redundant data; that redundant data can be removed by intelligently choosing not to code what the human ear cannot perceive. In this way, audio encoders deliver to the ear only the information that is needed in order to perceive the originally intended audio content without any extra information. There are three main mechanisms or weaknesses of the HAS that an audio coder can exploit: *Threshold Cutoff*, *Simultaneous Masking*, and *Temporal Masking*.

The human ear detects sounds as a local variation in air pressure called the sound pressure level (SPL). Longitudinal pressure waves strike the eardrum, which in turn transduces the incident pressure wave sending it along to be ultimately analyzed in frequency content by the ear's cochlea. Electrical impulses are then sent to the areas of the brain that process sound.

The ear is not perfect, however. Some sounds will be quiet enough to not be detected by the HAS. This lower threshold of hearing is not fixed for all frequencies. This minimum threshold of hearing in quiet has been empirically measured and is generally modeled by the following non-linear equation [17]:

$$(1)\ T_q(f) = 3.64(f/1000)^{-0.8} - 6.5e^{-0.6(f/1000-3.3)^2} + 10^{-3}(f/1000)^4$$

Figure 1.4 shows four tones, two of which are audible since they exceed the threshold of hearing in quiet, and two of which are not audible since they fall below the threshold. Any frequencies with power measured in dB_{SPL} that do not exceed the threshold cannot be heard and therefore do not need to be coded in the lossy coder.

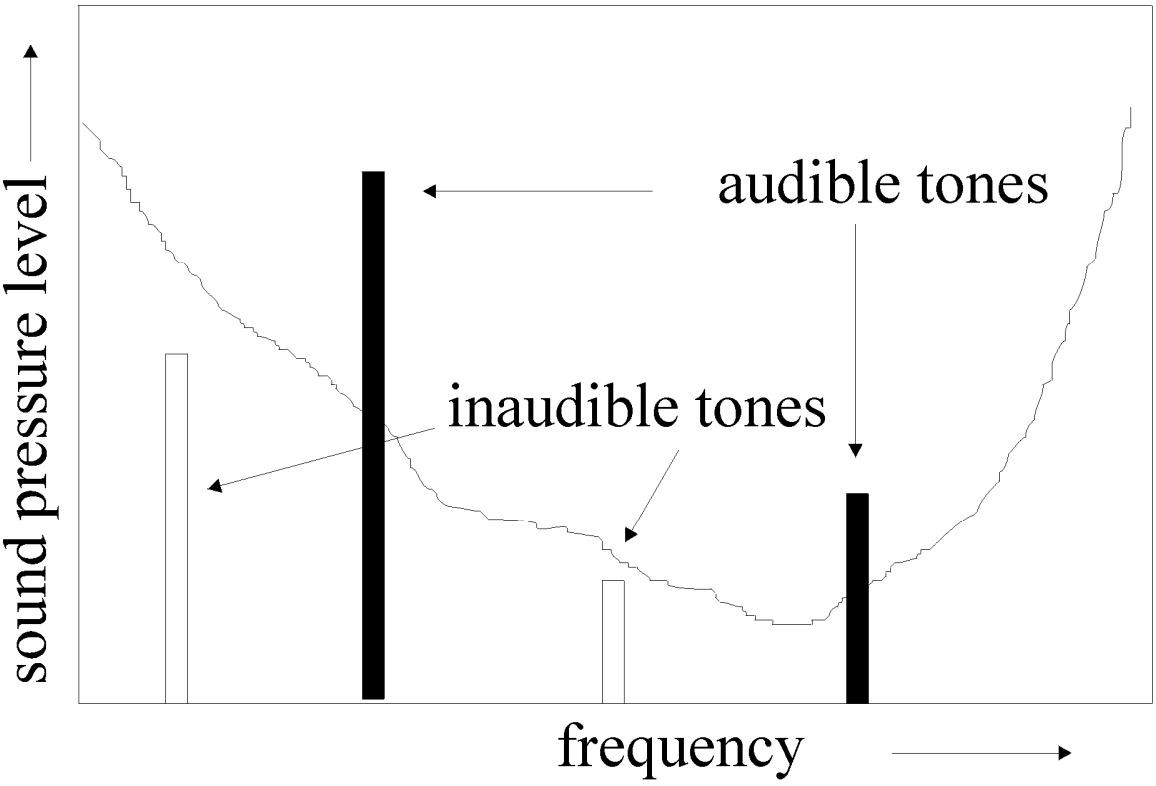

Figure 1.4 Minimum threshold of hearing in quiet

The threshold cutoff describes the minimum levels a signal or tone must reach before the HAS can detect it. However, even if a signal exceeds this threshold of audibility, there is a chance the HAS will not be able to detect it depending on what other components exist in the signal. This type of masking is called simultaneous masking. Two simultaneously occurring tones, audible when sounded separately, may not be detectable depending on how close they are in frequency. Signal strength or amplitude

also plays a role. If one sound is much louder than another, the weaker signal is not audible and is said to be "masked." For example, the sound of a clock ticking may only become audible after the television set is turned off in a room. In order for an audio coder to take advantage of this, a frequency analysis must be performed on the incoming audio signal to determine the incoming signal's frequency components. This is known as the psychoacoustic model. A good audio encoder uses an accurate psychoacoustic model to mimic our auditory system and thus is able to filter out these extraneous sounds and eliminate them from the output bitstream.

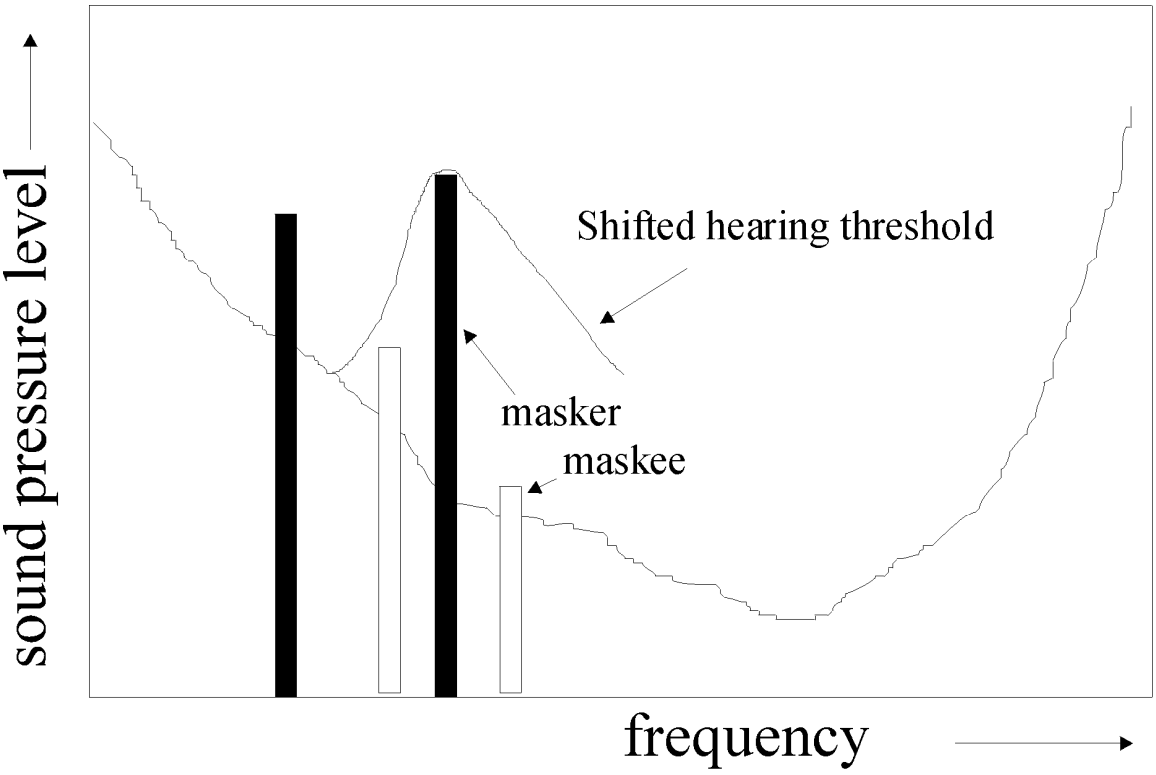

Figure 1.5 Simultaneous masking

The psychoacoustic model in the encoder does an analysis for each input block of audio data and provides information to the encoder's quantizer about each incoming block of incoming audio. If a particular frequency band (32 of them in many encoders) in that block is considered to be "audible" by the psychoacoustic model, it will ultimately

be coded by the encoder. The encoder may have a maximum of 16 bits to use to encode each frequency component or subsample. However, the coded value need not be represented by the full bit range of the original PCM audio (16-bits). By quantizing these coefficients with coarser levels, bits are saved, at the cost of "quantization noise" or noise introduced by such coarse quantization. If 4 bits instead of 16 bits are used, not as fine of an approximation can be represented, but bits will be saved and data rate will ultimately be reduced. This quantization noise introduced by the encoder should ideally be below the computed global "masking level" in each subband. Thus, the final coded output will be much smaller in size than the original with little or no perceptual loss depending on bit-rate and other factors.

It has been found through empirical evidence that sounds that are rather noise-like in quality have different masking properties than those that are tonal in quality. In general, it can be said that noise masks tones better than tones mask noise [17]. Also, tones more easily mask other tones and have more difficulty masking noise [3]. As part of a psychoacoustic analysis, peaks in the audio signal can be classified as tonal or non-tonal in nature. A tonal component will have a sharper peak characteristic than a non-tonal component in general. How an encoder computes or discriminates between tonal and non-components is important since this ultimately determines how accurately the psychoacoustic model portrays the audio signal.

In addition to simultaneous masking, which is a frequency-domain phenomenon, masking also occurs in the time domain. If a loud sound is followed or preceded by a softer sound, if the conditions are correct, this softer sound may be masked and go unnoticed by the ear. It is intuitive that a soft sound may go unnoticed after a loud sound,

this is known as “post-masking” or “forward masking.” It is the more prominent of the two temporal masking effects. The “pre-masking” or “backward masking” is somewhat counter-intuitive in nature but it has been shown empirically that a small amount of masking can occur prior to the onset of a masker. However, it is generally only effective for a much shorter time span. Figure 1.6 shows the concept of how a masker might provide some pre- and post- masking before and after it is sounded.

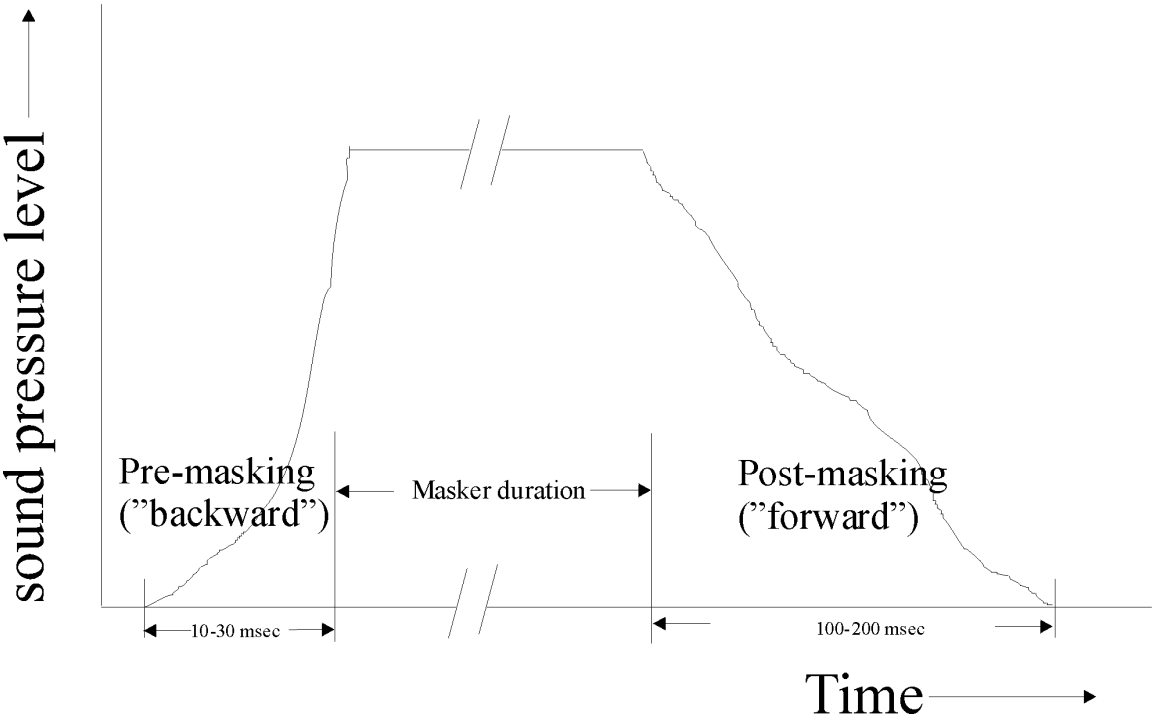

Figure 1.6. Forward and backward masking [5]

1.7 Spectral Analysis

With the description of the HAS, we have the essence of what is known as the “psychoacoustic model” of the encoder – a set of rules which determines what can and cannot be heard by a hypothetical listener. Two out of three of these phenomena are described in the frequency domain. Therefore, the audio signal is more efficiently

described in the frequency domain. Typically, an audio signal is analyzed using some form of transform. Coders were formally divided into "subband coders" and "transform coders," however, this distinction is now generally a deprecated or otherwise artificial term; with hybrid filter banks and such being used it is now becoming less descriptive of a term for modern coders [17]. Various methods exist to transform an audio signal from a time-domain representation into a frequency-domain representation, such as the fast fourier transform (FFT), discrete cosine transform (DCT), quadrature mirror filters (QMF), pseudo-QMF (PQMF), modified-DCT (MDCT), and the wavelet transform (WT). This list is by no means exhaustive but is representative of the methods available; different standards will select different transform methods based on particular criteria. The goal of the transform remains the same however, in that the signal must be mapped into some sort of frequency domain representation for the quantizing step that happens next. Figure 1.7 shows a high-level block description of an audio encoder with its four main tasks blocked and labeled.

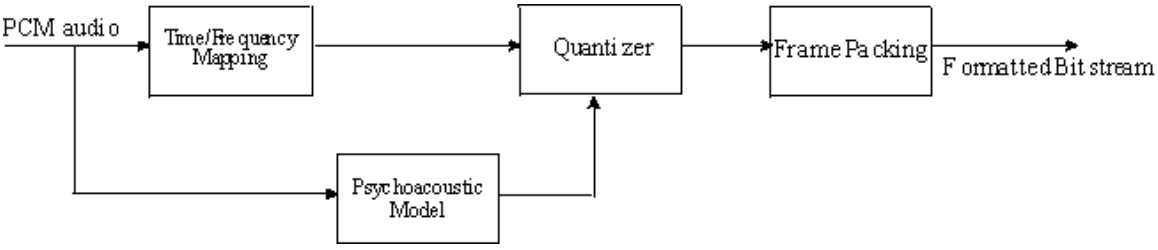

Figure 1.7 Basic operation of audio encoder

1.8 Quantization

The quantization step is the process of taking a large number of possible input values, and representing them with a more restricted subset of output values. In this way, less information is sent, and, therefore, bandwidth is conserved. A simple example of

this would be quantizing or rounding the value of "2.3" to just "2" – with no decimal place included. Similarly, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, and 2.4 would all be quantized to 2. Audio coders generally achieve most of their compressing ability by intelligently choosing to quantize some values and zeroing out others which are not audible according to its psychoacoustic model. Quantizing audio with fewer bits always causes some distortion or noise. Done properly, this added quantization noise will go unnoticed by the HAS due to the weaknesses inherent in the system. Once the signal is described in a frequency domain, a quantizer block can make decisions on which coefficients are most suitable for using fewer bits to quantize each value based the psychoacoustic model's evaluation of the audio signal block. This is the essence of perceptual audio coding. It is also apparent that this is one of the major areas in the encoder in which information is removed or left out. It is removed intelligently, but it is still removed.

1.9 Bit/Frame Packing

The final task in an audio encoder is to package data into a frame format that can be readily understood by a decoder. An encoding standard will usually specify the bit packing order that an encoder must obey to be deemed compatible. In this way, decoders can parse and understand the bitstream created by any encoder adhering to the standard in question. By specifying the bitstream format only, this allows for advances in technology such as filter improvements or psychoacoustic model improvements to be incorporated into newer encoders without the need for continually revising or specifying a new standard.

1.10 Summary

This chapter described the process of reducing the bit-rate of an audio signal. An analog signal is first converted into a digital PCM format if it not already digital. The PCM format is useful and robust, however is not optimal for Internet and consumer audio applications because of its overly large bandwidth. There are two types of data reduction generally, lossless compression and lossy reduction. Lossless compression perfectly recreates the input signal and relies on statistical redundancy to achieve compression; lossy reduction discards information in coding. Lossy coding is more appropriate for audio encoding due to its performance at higher compression ratios. Three basic mechanisms which perceptual coding of audio relies on are threshold cutoff, simultaneous masking, and temporal masking. An audio coder is generally comprised of the following basic building blocks:

- time/frequency mapping
- psychoacoustic analysis
- quantizer
- frame/bit packing

With the basics of audio encoders covered, Chapter 2 will look further into the specific causes of cascading audio coding error.

Chapter 2: Tandem Coding Problem

The need to re-encode previously encoded content with minimal audio artifacts becomes increasingly important. This problem is well-understood in the broadcast industry [8, 15, 23]. When multiple encoders are placed in a cascade configuration, the coded version of the audio is essentially the source. A problem arises with audible coding artifacts left by the original coder due to the inherently lossy process. Though perhaps not obvious upon one coding generation, these audible artifacts propagate through to any coded waveforms and can become more audible if multiple generations are considered.

The audio codec (coder-decoder) process is generally viewed as a one way or terminus operation, and rightly so. It is most beneficial for an audio encoder to be as efficient as possible upon one pass of the input material. This means the encoder will be throwing away as much signal information as possible in order to minimize the rate/distortion metric [21].

This chapter takes a closer look at the details of the tandem coding losses. In order to preserve the signal quality, it is necessary to understand exactly what information is being lost, and where, and to characterize this information loss so that a strategy can be developed to combat the problem across multiple generations.

2.1 Overall Loss in Normal Encoder

Figure 2.1 shows the generic scheme for an encode-decode process. For purposes here, optional encoder processes such as entropy coding, joint stereo, or other coding enhancements are not considered.

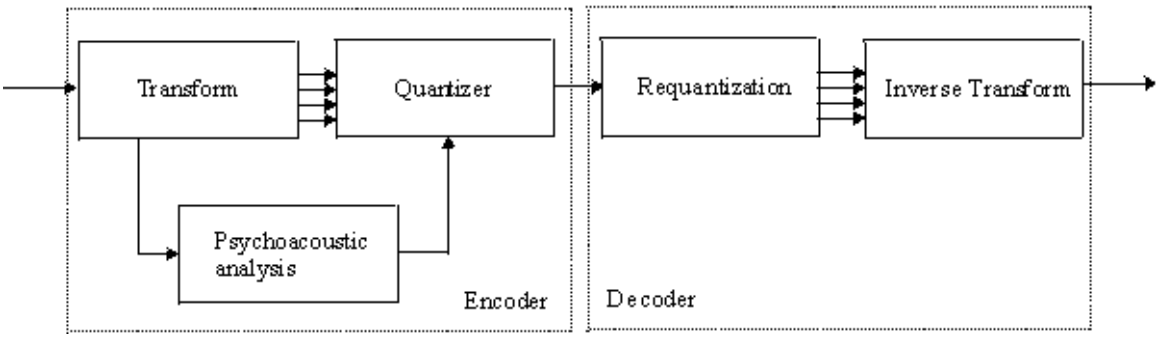

Figure 2.1 Generic encoder-decoder [13]

The encoder first performs a block transform Tx on the input signal x. For each input signal x, the encoder also performs a spectral analysis Sx of the input block in parallel to obtain parameters for a psychoacoustic model $\Psi(x)$. A bit allocation $b(\Psi(x), Tx, x)$ is then performed to determine a set of quantization rules which best fit the psychoacoustic model implemented [13].

Code words and side information are sent to the decoder. The decoder uses de-quantization, codebook lookup, or inverse scaling to reconstruct the subband samples sent to it by the encoder. It then performs a synthesis or inverse transform T^{-1} on this data to recreate the PCM audio signal [13].

Sources for error in this model were enumerated as follows by Kurth:

1. lossy quantization
2. round-off and arithmetic errors due to T, T^{-1}, and S operations
3. aliasing due to T (cancelled by T^{-1} only if no quantization occurs)
4. near perfect reconstruction errors (if T and T^{-1} are only pseudoinverses)
5. inaccuracy of the psychoacoustical model Ψ
6. non-suitable time-frequency resolution in spectral analysis
7. missing translation invariance of T and T^{-1} (in view of multiple coding)

In comparing the magnitude of each effect, the quantization error item 1 dominates all others and is most likely to contribute to signal degeneration. For example, in extremely low bit-rate encoding, many of the upper frequency (transform) coefficients may be set to zero to conserve bandwidth. It is this error that is most responsible for perceptual quality loss. Items 2 and 4 are each minimal but must be watched carefully dependent on the codec. Item 3 is usually dealt with by techniques such as windowing [27]. The others are systemic errors which are not discussed at length in this research.

For these reasons, the lossy quantization step will be the prime source of error that this research will address in order to minimize cascading coder error.

2.2 Quantization Losses

Lossy codecs are able to achieve high coding gain with minimal perceptual loss by taking advantage of what is known as "course quantization." Eleven-to-one and higher reduction ratios can be achieved for transparent audio encoding. This equates to CD-quality audio originally coded at 16 bits/sample being represented by roughly 1.5 bits/sample (128kbps), saving bandwidth with minimal perceptual loss. Further techniques may be used to push the threshold of transparent coding to much lower bit-rates than this historical precedent of 128 kbps encoding. As discussed earlier, the quantization noise introduced by this procedure is largely imperceptible because it is "masked" beneath the threshold of what is calculated by the psychoacoustic model.

Nevertheless, there will undoubtedly be some loss associated with the encoding step, no matter how minimally imperceptible, and therein lies the problem. Because it is here in the quantizer where most of the coding gain is achieved, it is also the quantizer which must be looked at first when searching for an answer to the problem.

Consider a tandem quantization process as two stages of encoding. For example, in the first step, the signal is quantized with a 5-level quantizer. This will introduce some quantization noise into the signal. In the next quantizer step, this previously quantized signal is presented as the input to a 7-level quantizer. The 7-level quantizer represents the second stage encoder. The question that must be answered is, "Does this introduce more quantization noise than already previously present? Or does it somehow decrease the amount of quantization noise?" One might be tempted to say "no" since a 7-level quantizer has finer resolution than a 5-level quantizer. This answer would be correct if both quantizers were compared to each other after quantizing the original signal; however, because they are in a cascade configuration, the output of the first is fed to the second quantizer as its input. Under these conditions, the second quantizer will still introduce its share of quantization noise despite its increased accuracy, which is additive in nature. It will therefore increase the noise of the overall system.

An example will further demonstrate this. Consider the following series of 12 subband samples quantized with a uniform 5-level quantization scheme.

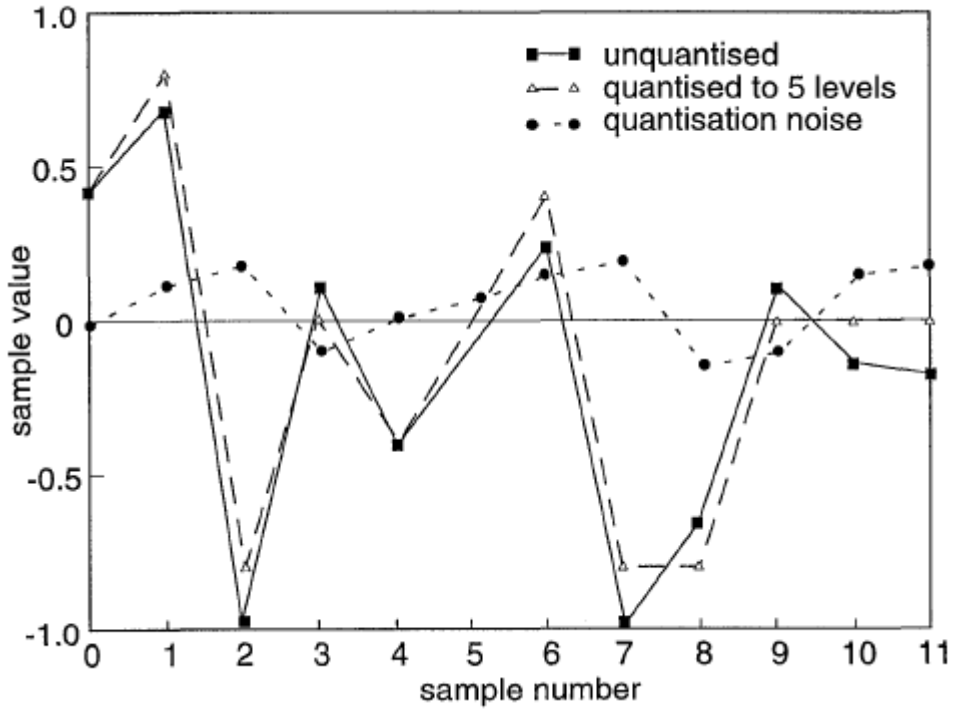

Figure 2.2 Subband sample quantization, 5 levels [7]

Figure 2.2 shows the quantization noise introduced by using a 5-level quantizer, obtained by taking the difference of the original sample and subtracting the 5-level quantized version. Figure 2.3 shows this same signal being re-quantized with a 7-level quantizer and the resulting quantization noise introduced (tandem or cascaded coding stages).

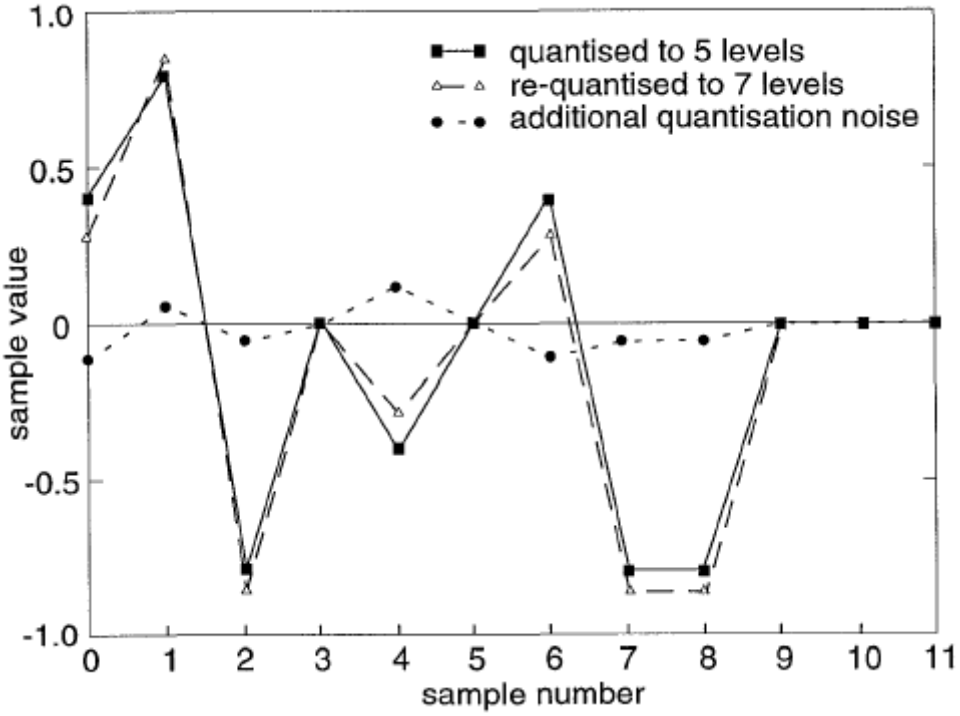

Figure 2.3 Subband sample quantization, 7 levels [7]

This second encoding stage introduces even more quantization noise on top of what the first stage did, even though it is finer in resolution. It is also interesting to note that, had a 5-level quantizer been used instead, in this hypothetical example, no further quantization noise would have been introduced since the samples would be re-quantized to their original values. This demonstrates the importance of matching the quantization scheme of future encoding stages with the original scheme used to create the encoded bitstream.

2.3 Filterbank delays

Another potential cause of concern in the general case is that during the transform stage of the encoder, T, there could be a filterbank delay of a certain number of samples introduced by the encode-decode process. In the general case, even if a signal is passed through the same exact encoder, if the alignment of the input signal is offset by any

number of samples (even one), different subband samples will be created by the filterbank and hence, different quantization values will also be generated [7].

The delay is not consistent across all encoders, it is entirely dependent on the codec in question. It perhaps may not exist at all. If there is a sample delay, it usually manifests itself in the output PCM samples created by the decoder in the form of a string of extra values (some or many of which may be 0-values) being written to the decoded output file at the beginning and/or the end of the file. Figures 2.4 and 2.5 show the beginning and the end of an audio file (excerpt.wav) juxtaposed with its .mp3 encoded-decoded version below it. It can be seen that after one encode-decode stage, the operation has written extra sample values to the beginning and end of the audio file that weren't in the original. The encoded-decoded file has 66,816 samples; the original has 64,356 samples total. In this case, the process has appended 2460 audio samples to the original uncompressed waveform. This padding becomes a problem for the general case tandem audio coding problem; any successive encodes will see entirely different sets of input audio blocks than existed for the first generation encode cycle. From a user's standpoint, this problem is difficult to get around. It seems highly unrealistic to ask a user to have to use low level audio editing software to prepare audio material before encoding. This issue of "framing synchronization" will be revisited in greater depth in Chapter 3.

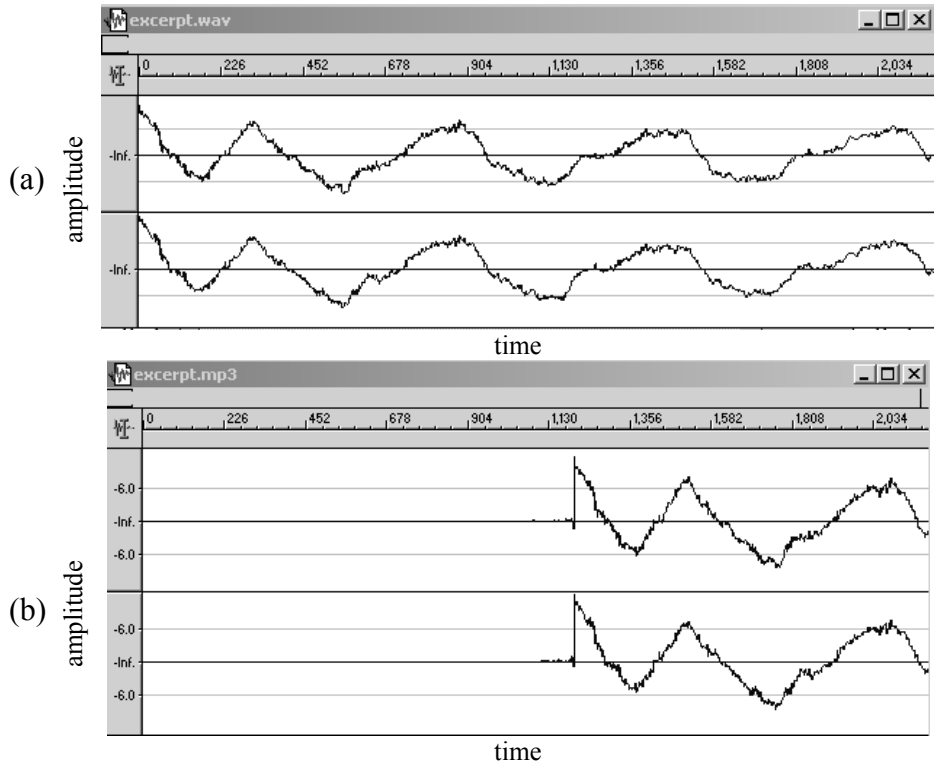

Figure 2.4. A sample-aligned juxtaposition comparing the beginning of an original and encoded file
(a) The original file and (b) The mp3 encoded file

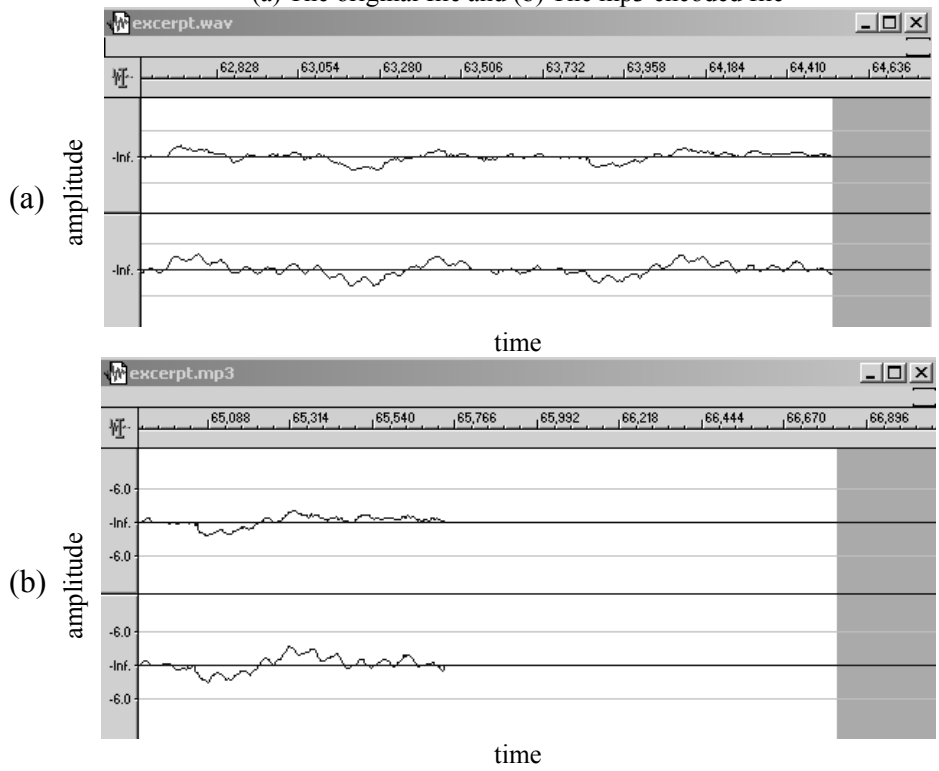

Figure 2.5. A comparison of the end of an uncompressed file and its encoded sibling
(a) The original uncompressed .wav and (b) the mp3 encoded file

It can be seen in the figure that the codec operation has written some extra 1200 samples to the beginning of the decoded waveform and a similar amount to the end of the waveform. The above example was performed using the mp3 plug-in with Sonic Foundry's Sound Forge 6.0. As can be seen, this problem is difficult to quantify in the general case, since any number of codecs can be used which will result in a different number of extra samples, or perhaps no extra samples at all. For instance, the above experiment was repeated using Windows Media Audio V8 plug-in and 18 total fewer samples were present in the decoded file after the codec operation. Table 2.1 summarizes some various offsets off common encoders which exist in the public domain. Some encoders are specifically designed to output a number of samples equal to the input samples.

Table 2.1 Offsets for various codecs [24]

Encoded with:	**Decoded with:**	**Offset, in msec**
Lame 3.92 – alt-preset standard	Lame 3.92 –decode	0
Blade 0.94.2, 256kbps	Lame 3.92 –decode	0
Thompson mp3pro 1.04, 64 kbps	Thompson mp3pro, 1.04	105
MMJB 7.2 mp3pro, 64 kbps	MMJB 7.2	79
Wm8eutil 8.0.0.0343, 64 kbps	In wm.dll from Winamp 2.6	46
Oggenc 0.9 (rc3), (-q 3)	In vorbis.dll from Winamp 2.80	0
Mppenc 1.04 (--standard)	Mppdec 1.04	0
Quicktime 6 AAC, 64 kbps	Quicktime 6	66

This mismatch of filterbank lengths and output file lengths contributes to the problem of cascading coding losses since even if quantization parameters could be retrieved, they are somewhat meaningless without identical filterbank lengths. And many audio codecs do not share identical analysis filterbank sections. Table 2 shows the differences in filterbank properties among several audio codecs [19]. Also, in [4], Brandenburg discusses various filter types used in audio coders.

Table 2.2 Comparison of filter bank properties [19]

Feature	MPEG L I	MPEG L II	MPEG L III	Dolby AC-2	Dolby AC-3	ATRAC*	PAC/MPAC
F resolution at 48 kHz	PQMF	PQMF	Hybrid PQMF/ MDCT	MDCT / MDST	MDCT	Hybrid QMF/ MDCT	MDCT
Time resolution at 48 kHz	0.66 ms	0.66 ms	4 ms	1.3 ms	2.66 ms	1.3 ms	2.66 ms
Frame length at 48 kHz	8 ms	24 ms	24 ms	32 ms	32 ms	10.66 ms	23 ms

*ATRAC operates 44.1kHz. Values given are for a 48kHz operating f_s

2.4 Blocking of Data

Most modern audio coders are blocking algorithms by nature. That is, they parse an incoming audio signal into smaller block-lengths and operate on these smaller chunks of data. This methodology is partly responsible for the above problem of differing frame output lengths, but is necessary for processing time and efficiency. This blocking of input data typically ranges from 400 to 2048 samples at a time, and blocks are fed simultaneously to the time-frequency transform and the psychoacoustic model to create the masking threshold necessary to quantize the subband samples or frequency coefficients [18].

It is this blocking structure which is responsible for what is known as “pre-echo” distortions. It is desirous to use long block lengths in order to obtain better frequency resolution, however time resolution is decreased. If a transient signal such as a cymbal hit occurs in the middle of an audio block, the energy in the transient may prompt relatively coarse quantization. However, the transient cannot adequately mask the quantization noise. After quantization in the encoder and inverse transform in the decoder, the quantization noise will be spread across the block duration. This pre-echo

noise therefore announces the transient. It is an undesirable artifact. Figure 2.6 shows an example of a transient occurring in the middle of a input block, the pre-echo occurring in the reconstructed signal, and the resulting difference signal (shown on a smaller scale).

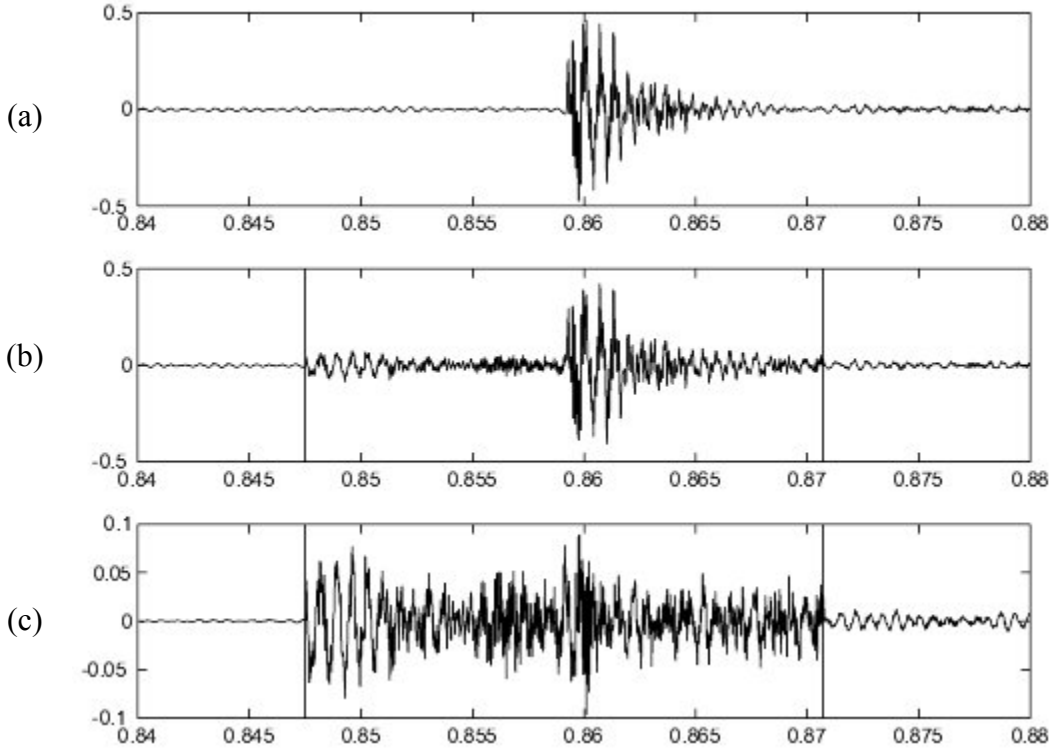

Figure 2.6 Pre-echo.
(a) the transient occurs in the middle of an input block of data
(b) upon re-structuring, quantization noise is introduced prior to onset of the transient
(c) the difference signal showing the error introduced into the output [18]

Many coders address this problem by detecting transients in the input audio signal and switching to a shorter input block length. This shorter block will have better time resolution at the expense of frequency resolution. However, since it is only used during heavily transient attacks, the pre-echo errors are minimized and a decrease in frequency resolution is experienced only momentarily by the coder. A short window is not used for the entire length of the signal since it is inefficient from a coding standpoint (overhead from large amount of side information created for each block). As soon as a stationary signal is again detected, long windows are used again to once again obtain better

frequency resolution. In order to make this scheme work, transition windows also have to be specified to cancel the aliasing distortions that would result. For a more in-depth discussion on the use of window-switching in audio coders (MPEG and otherwise), see [4] and [10].

Due to the blocking nature of all modern codecs, if the input samples presented to a second stage encoder do not coincide on a sample-by-sample basis as the first step, the quantizer will "see" different subband samples or frequency coefficients as a result of the offset inherent in the encoder's filter bank. This is the case for many codecs, as there is no guarantee that all codecs will generate a 0-sample delay length. The blocking nature of audio coders makes it difficult to obtain identical quantization parameters in the encoder unless a sample by sample alignment is achieved, which is difficult to specify or dangerous to assume in a typical cascading audio scenario, without significant knowledge of the nature of the codec.

2.5 Summary

Most of the overall loss in a one-stage encoder occurs at the quantization step. Among encoding standards, differences in filterbank types and lengths lead to a cacophony of delay lengths and file size differences. Codec operations do not as a rule guarantee a 1:1 correspondence from uncompressed input samples to decompressed output samples. There is little or no motivation to resolve these discrepancies. This presents a dilemma when attempting to characterize the cascading coder problem in the general sense. These are important issues that should be kept in mind, if not addressed directly, if a multiple-generational encoder is to address the problem of accumulating encoder error.

Chapter 3: Previous Efforts

The problems discussed in Chapter 2 have not gone unnoticed by audio engineers. The cascading coder problem has been addressed by researchers and is receiving more attention as perceptual encoding gains a stronger foothold in the audio industry. This chapter will take a closer look at the previous research methods aimed to answer the question: "Is it possible to retain as much as possible of the original PCM representation of an encoded format?" A review of the literature indicates that such a process is at least partially achievable, although perhaps not fully optimal at this time. Each method has its limitations. Even if not perfect, however, it is highly desirable to obtain an accurate as possible reconstruction of the original signal for the purposes of re-encoding or multiple generation encoding. Three particular strategies will be discussed in this chapter:

- The "MOLE signal" from Atlantic Audio/BBC.
- Frank Kurth's method (there is no formal name for this codec scheme).
- Fraunhofer Institute's "inverse decoder".

3.1 Atlantic Audio – the MOLE signal

The ATLANTIC consortium (Advanced Television at Low bitrate And Networked Transmission over Integrated Communication systems) is a research and development group funded by the BBC. This group observed the progressive deterioration in quality resulting from cascaded coding and decoding processes in their work [8]. In order to overcome this limitation, they realized that this technological hurdle must be overcome to achieve their objective to assemble and demonstrate the essential components of a television program distribution chain based on compressed signals. It was primarily a video problem, but the audio portion of that project will be discussed in this section.

John Fletcher proposed a solution to this problem known as the MOLE signal in order to reduce the problem of tandem coding artifacts [7]. The essential premise behind the MOLE signal is to send extra information along with the encoded audio signal to help in the optimal reconstruction of the original PCM input bitstream. This data itself is what is referred to as the MOLE signal. A decoder or encoder which is able to use (decoders) or create (encoders) MOLE signals will show better perceptual quality when in the presence of multiple generations of audio encoding stages. A MOLE signal must be created by an encoder at encode time and included in the bitstream. Such MOLE-assisted codecs are important to investigate further as they lay some important theoretical groundwork for the reduction of tandem coding artifacts.

A critical piece of the encoding puzzle is the analysis filter used to extract the subbands from the input PCM samples. This comes prior to any quantization in the typical encoder, since the quantizer must make decisions based the subband analysis provided by the filterbank separation. In the previous section, it was shown how a second stage quantizer could eliminate added quantization noise if the quantizer step size is known exactly a priori. This assumes however, that the set of input samples presented to the second stage quantizer is the exact same sequence as what the first encoder saw (framing synchronization). In practice, this would be difficult to achieve because encoders introduce delays, and in general it is not obvious to discover these delays without experimentation or knowledge of the original encoding scheme (refer to Table 2.1).

Using the example of an MPEG-Layer 2 encoder (the standard selected by ATLANTIC Audio), incoming PCM samples are formed into blocks of 32 when fed to

the analysis filterbank. For each block of 32 input samples, one sample in each of the 32 subbands is created. This is repeated 36 times, until each of the 32 subbands "fills up" with 36 samples each, for an overall structure of 1152 samples/frame. In this way, the incoming audio is broken up into the subbands each having bandlimited content. If the incoming audio is offset by one sample, however, the analysis filterbank will produce different output subband samples. Figures 3.1 and 3.2 show the possible blocking boundaries; Figure 3.3 shows how these would create entirely different subband samples in one subband for the same input.

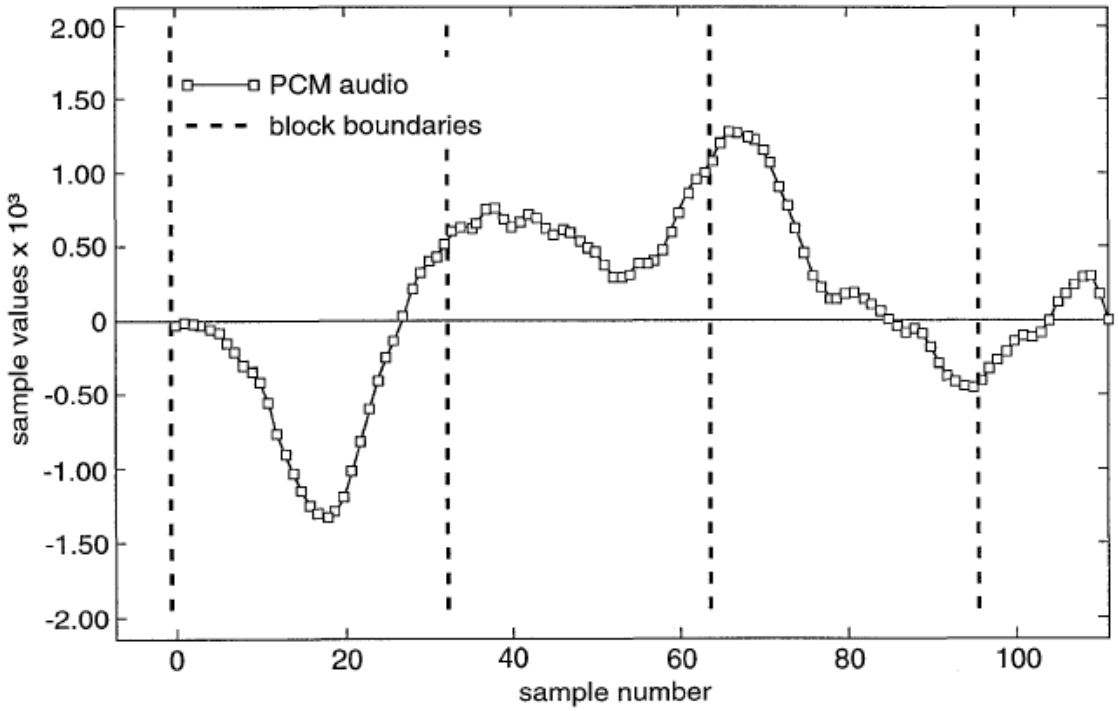

Figure 3.1 32-Sample block boundary [7]

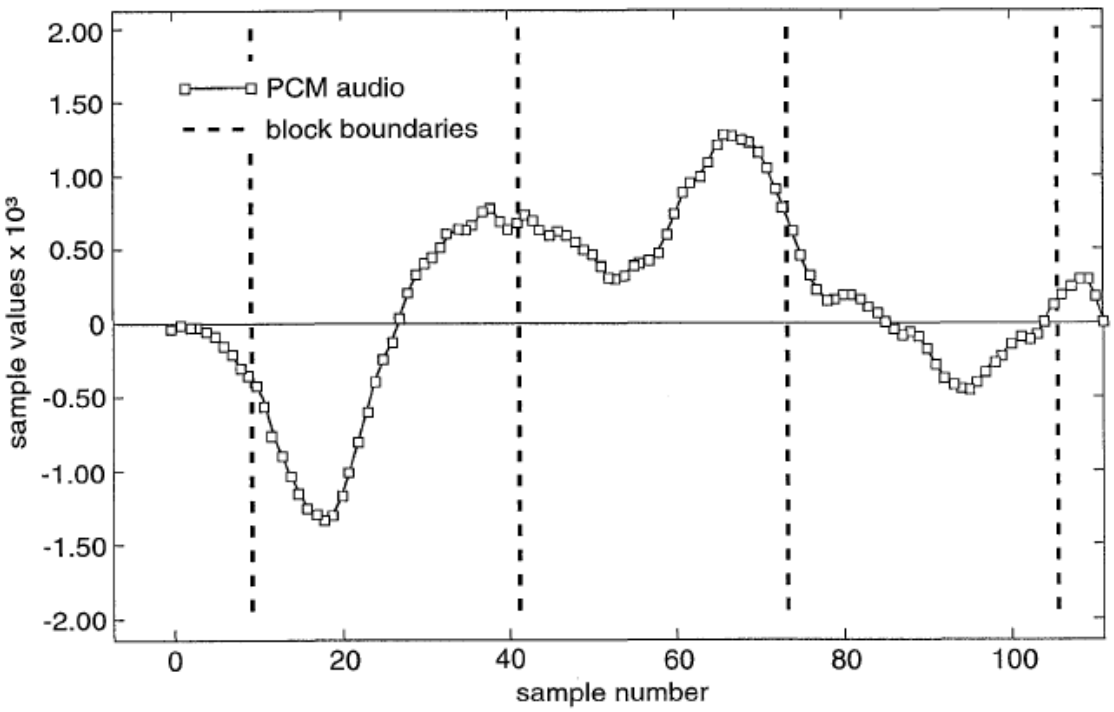

Figure 3.2 A different 32-sample block boundary [7]

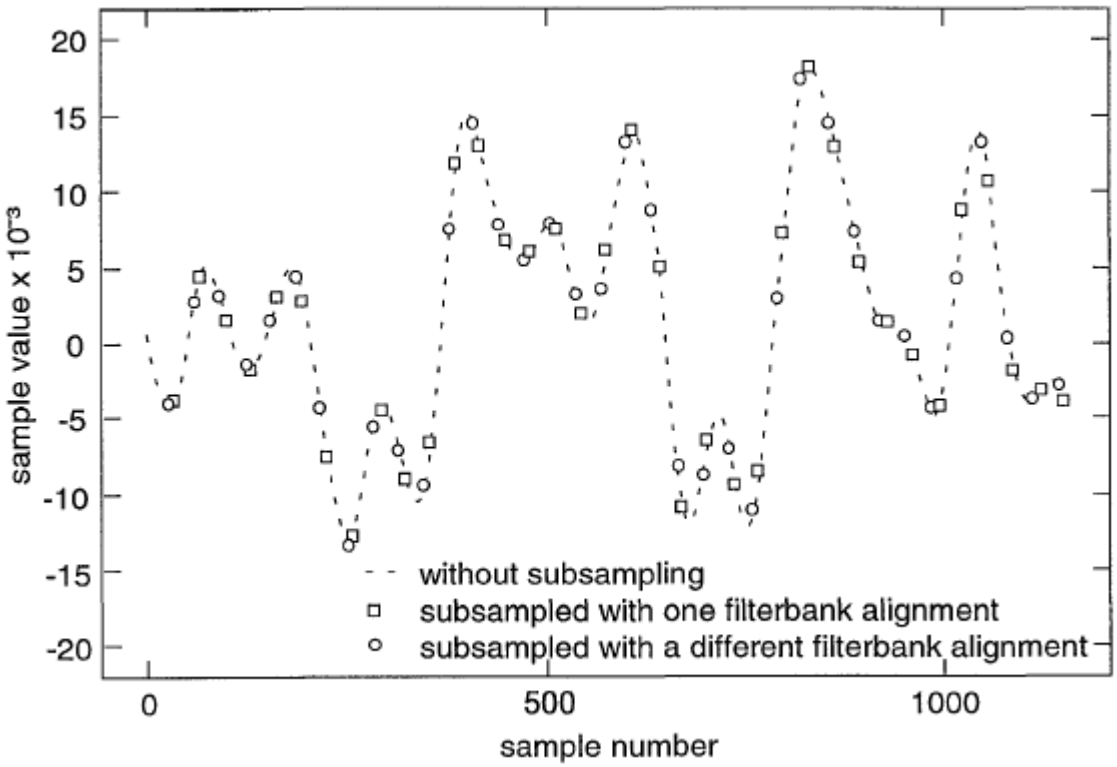

Figure 3.3 Different subband samples produced in the analysis filter bank
by offsetting the 32 sample input blocks [7]

If applied to a second stage encoder, these different subband samples will provide no help in reducing quantization noise in any potential second stage since they will produce completely different samples. In order for the optimum reconstruction to occur, the overall blocking strategy of the encoder must be recovered.

The effect of alignment can be accounted for in tandem coding even if no quantization is forwarded to the second stage. The following experiment was performed by the ATLANTIC Audio group. The results are indicative of the importance of the filter alignment. The experiment setup is as follows.

An uncompressed audio file was first passed through one stage of encode-decode stage. This “one pass” file was then compared to several “two-pass” files. In the “two pass” files, the alignment of the signal presented to the second encoder-decoder stage was varied over a range of 0-1152 samples (one MPEG-2 frame) by inserting silence in the one-pass file prior to the encoding to introduce the offsets. These two files were then aligned (adjusting for the 481-sample delay of the encode process) and subtracted from each other to produce a residue file. This residue file contained the “noise” of the aggregate two-stage encode-decode process. Then, the ratio of the original one-pass file (signal) was compared to the residue file created by the second stage (noise) to determine an SNR ratio for the second stage of the encode-decode process [7]. Figure 3.4 shows the resultant SNR vs. sample offset plot.

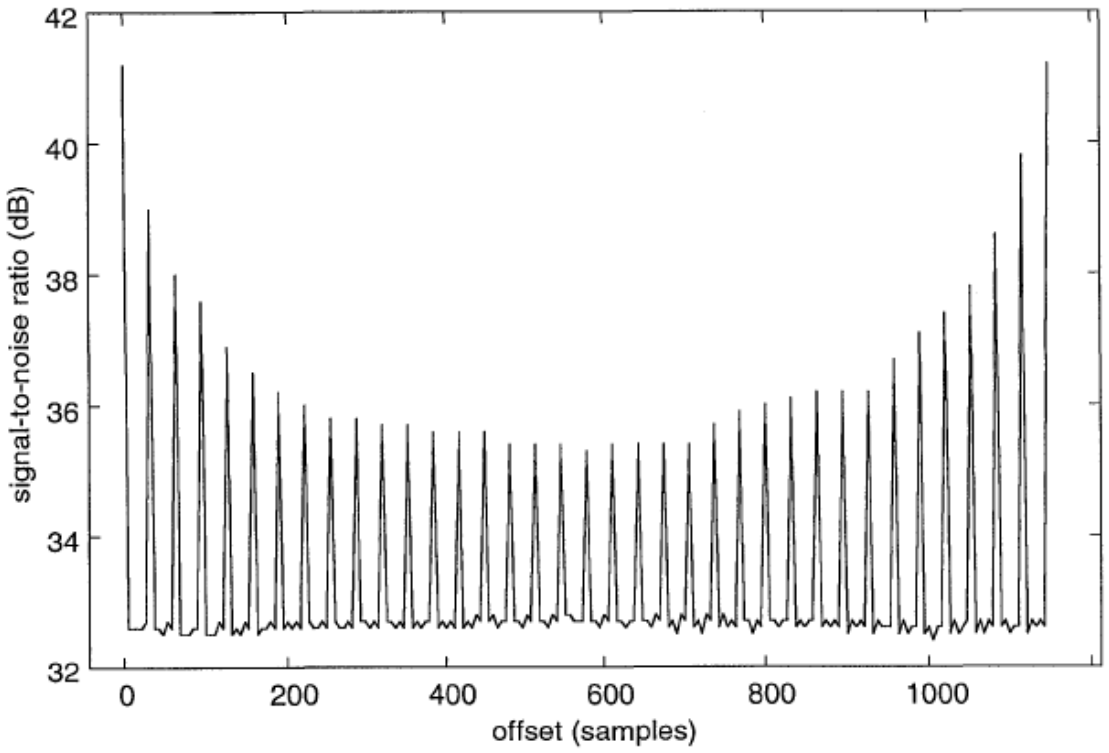

Figure 3.4 SNR vs. sample offset [7]

In this plot, the structure of the 32 sample filterbank is visible, as well as the larger frame structure of 1152 samples. If the sample offset presented to the second encoder is a multiple of 32 sample-offset (0, 32, etc. from the original), the SNR shows a spike at that particular offset, and is relatively a constant value everywhere else. It was shown through informal listening tests that second stage encoded bitstreams that aligned to the 32-sample filterbank boundaries did in fact sound better than those which did not conform to the boundaries.

The designers of the MOLE signal used these experiments to determine that the following information should be included in the MOLE signal for optimum reconstruction:

- Quantization information for each subband sample.

- Alignment of PCM samples relative to the filterbank.
- Alignment of PCM samples relative to the MPEG frame (encompasses the above).
- Header information.
- ID sequence and error check word.

Refer to Appendix A for the complete MOLE signal format [7].

When the bitstream is encoded/re-encoded and decoded with MOLE-assisted codecs, a second generation encode can be almost identically recreated by the proposed structure [7] (see Appendix A). Since the MPEG frame structure is maintained, the same input samples will be presented to the analysis filterbank. The second generation coder then reads the scalefactors and bit allocations from the MOLE signal instead of generating a new set. In this way, no further quantization noise is added to the signal. Header information is also read from the MOLE signal. A coded bitstream is then created by assembling this information. The resulting bitstream would be very nearly identical to its original. The only source of error would be due to the finite precision arithmetic of the filterbank. Perhaps some subband values would take on slightly different values, but it would be likely that the quantizer would re-quantize these values to their previous values. Only in the case of very low energy subbands might these errors manifest in the output stream.

The ATLANTIC Audio MOLE signal was a key proof-of-concept idea which showed that optimum reconstruction can be approached if information is known in advance of the quantization scheme and if filterbank alignment is maintained.

3.2 Frank Kurth's Codec

At around the same time that the ATLANTIC Audio project was experimenting with the idea of a codec impervious to multiple generational losses, Frank Kurth also had an interesting contribution [13]. Kurth had the idea to build on the idea of the MOLE signal but instead of sending the necessary information in a piggyback configuration which added coding overhead, he proposed a steganographic technique for burying the necessary sideinfo (the encoding parameters) into the transform domain signal without presenting any perceptual quality loss. Kurth worked with an MPEG 1 – layer II codec.

Kurth corroborated that quantization error was the main cause of perceptual quality loss for multiple generational loss, just as the ATLANTIC Audio project had determined. However, Kurth pointed out that recovering framing synchronization alone is not sufficient alone to solve the cascading problem wholly. Even for fixed frames, the energy per subband degrades due to the action of the encoding process. Kurth noted that there are two types of signal degradation depending on whether the signal was:

- Applied to subsequent codecs without any synchronization, or
- Fully synchronized to the original input and then fed into the codec.

Kurth's work showed that unsynchronized signals tend to produce audible artifacts and noise-like components, while the synchronized versions tended to attenuate frequency bands. Either of these signal degradations could be perceived to be more severe depending on the signal type and listener. Kurth contends that in successive generations energy is lost in upper bands primarily due to energy leakage from the subbands. [13]

Kurth's proposed codec used an embedding algorithm in the decoder to psychoacoustically bury the quantization information prior to the inverse transform step.

For details on the steganographic procedure, refer to [13]. Correspondingly, a detector in the encoder recovered the quantization information. If the signal was successfully recovered at the encoder, successive generations would re-use this quantization information instead of re-quantizing. If no embedded signal was detected or the embedded signal was detected as corrupted, the encoder would operate as a normal encoder and generate new quantization and bit allocation information. Also, embedding limits had to be computed and determined where information could best be perceptually buried in the data by the decoder. Since he had selected MPEG 1 – layer II, each 12-sample sub-block was chosen as the embedding blocks. The information Kurth embedded in the signal was the following:

- Header information (# of channels, bitrate, stereo encoding options),
- Bit allocation (2-4 bits for each subband's allocation),
- Scalefactor selection (2 bits indicating which of the 1-3 subblocks to apply scalefactors to),
- The scalefactors themselves (6 bits for each used).

Kurth's idea to re-use previous generation's encoding parameters in all subsequent codecs in cascade was similar to the idea of the MOLE signal, however by using steganographic techniques, he was able to reduce the overhead. Much of his work involved the particulars for reliable perceptually transparent embedding of data. It was nonetheless an important step in the right direction since he provided confirmation that it was possible to provide a codec resistant to tandem coding loss [13].

3.3 The Inverse Decoder

In some cases, quantization and framing information is not available. Is there some way to extract this information from the encoded stream itself? These are the questions which the "inverse decoder" effort from the Fraunhofer Institut concerned itself with. The inverse decoder recognizes the fact that encoders impose some sort of structure onto the bitstream that they create. One can take advantage of this to re-create a better reconstruction.

In [9], it was proposed that, through the decoder architecture, it was possible to determine the relevant bitstream information from the decoded PCM signal in a step-by-step process:

- Determine the decoder framing grid.
- Determine filterbank parameters, if there are multiple parameters available to the specific codec.
- Estimate the quantization information from the quantized spectral values.
- Recover any miscellaneous coding option used (joint stereo, etc.).

Note that the above procedure is applied to a full encoded-decoded bit stream. Figure 3.5 shows conceptually how the inverse decoder relates to other portions of the codec architecture family.

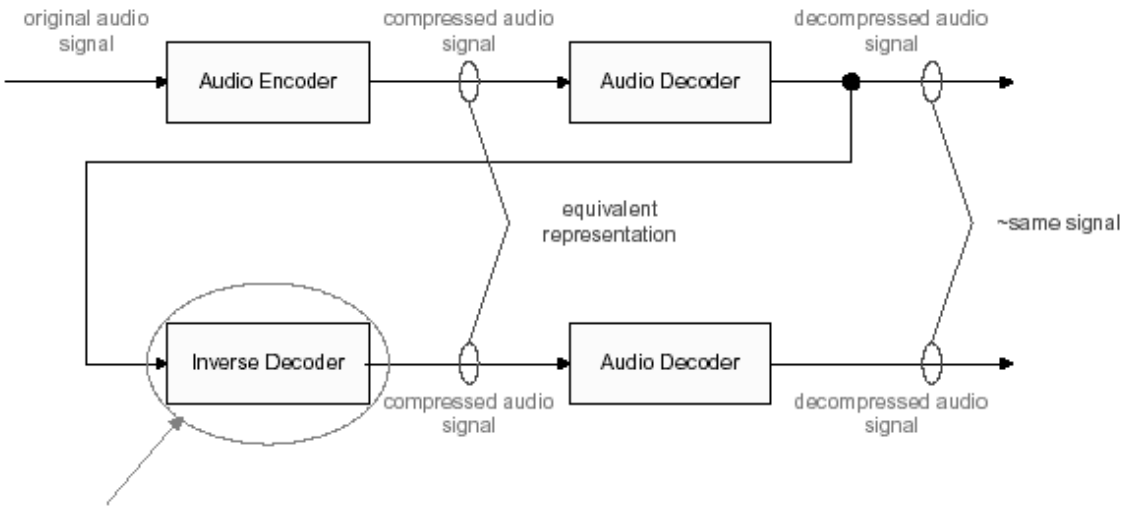

Figure 3.5 An inverse decoder [10]

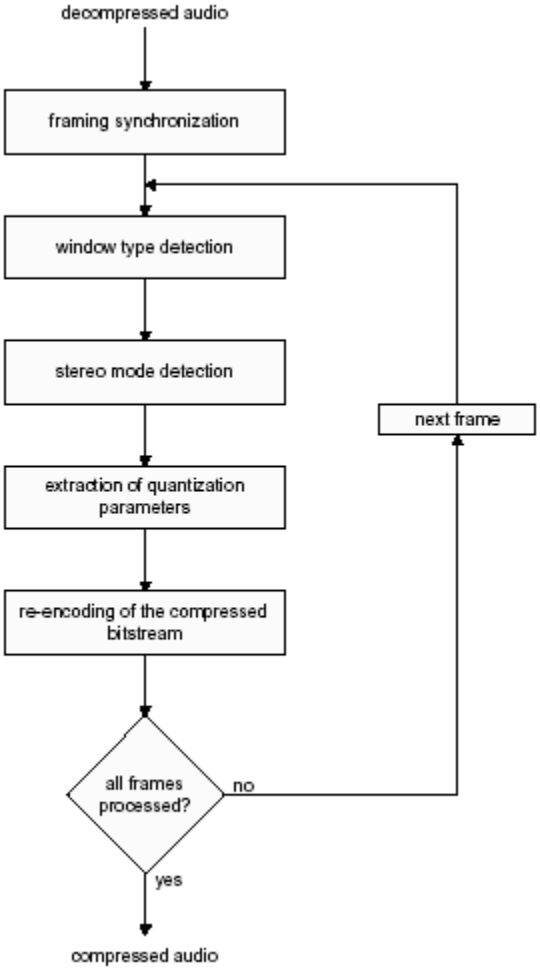

Figure 3.6 Inverse decoder algorithm [10]

Figure 3.6 shows the inverse decoder algorithm in detail. The following is a brief discussion of the major portions involved in the inverse decoding algorithm. For a discussion of the inverse decoder in detail, see [9] and [10].

As seen in the ATLANTIC audio project, recovering the coarse framing boundaries used by the codec is crucial to recreating an optimum reconstruction. In the case of MP3 coders, 576 samples will be decomposed into 576 spectral coefficients. The key is to correctly obtain these spectral coefficients according to their frequency bins.

An important consideration is that, at encode time, usually several spectral coefficients are quantized to zero due to masking effects of nearby spectral components. In order to uncover these zeroed coefficients, it is necessary to find the appropriate analysis conditions, as discussed previously in this document and in [9] and [10]. These become apparent only when:

- The framing offset used is identical to the one used in the encode process.
- The analysis filterbank of the inverse decoder is similar to the one used during the encode stage.

Figure 3.7 shows such a decoded audio signal which has been spectrally decomposed with a correct framing (top) and the same decomposition if a one sample delay is introduced (bottom).

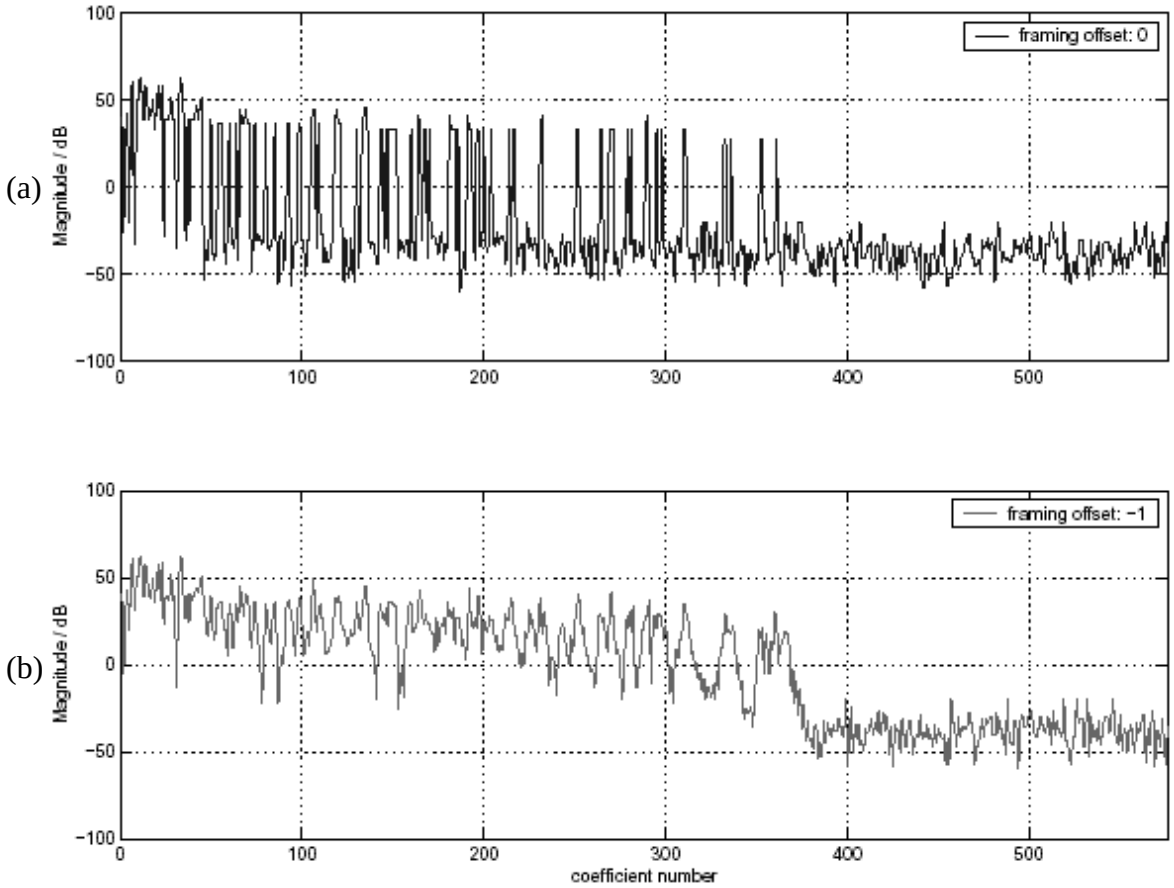

Figure 3.7 Spectral decomposition step for the inverse decoder. Difference between (a) an identical alignment and (b) a 1-sample offset framing for the spectral decomposition step [10]

In part (a) of this figure, it is apparent that in the first case, the spectral information is a much more compact representation since many of the spectral lines exhibit troughs or zeroes. Part (b) of the figure shows how even a one-sample delay will ruin this tight spectral representation of the encoder and smear the spectral content across the bins. Using this as a template, this concept is extended to recover the correct filterbank analysis parameters:

- A switching state of the analysis filterbank is assumed (for MP3: long, short, start, stop window types).

- The audio spectrum is decomposed using a range of sample offsets over the bitstream until the “compact” spectrum of the encoder is reliably recovered (a brute force technique).
- The detected framing grid is extrapolated over any part of the audio signal, if possible.

In this way, the correct framing grid can be inferred based on how much the spectral lines differ at each sample offset.

In Figure 3.8, we see the how the periodicity of the “fluctuation strength” or differences in spectral line computation is maximum at frame boundaries. This is due to the compact nature of the correct alignment compared to the relatively flat response for an incorrect alignment. The three insets show the frame grid detection for 48 kbps, 64 kbps, and 96 kbps. Notice that as bitrate is increased, the characteristic peaks grow ever smaller, meaning that it is more difficult to extract this framing synchronization. However, even at 96 kbps they are still detectable [10].

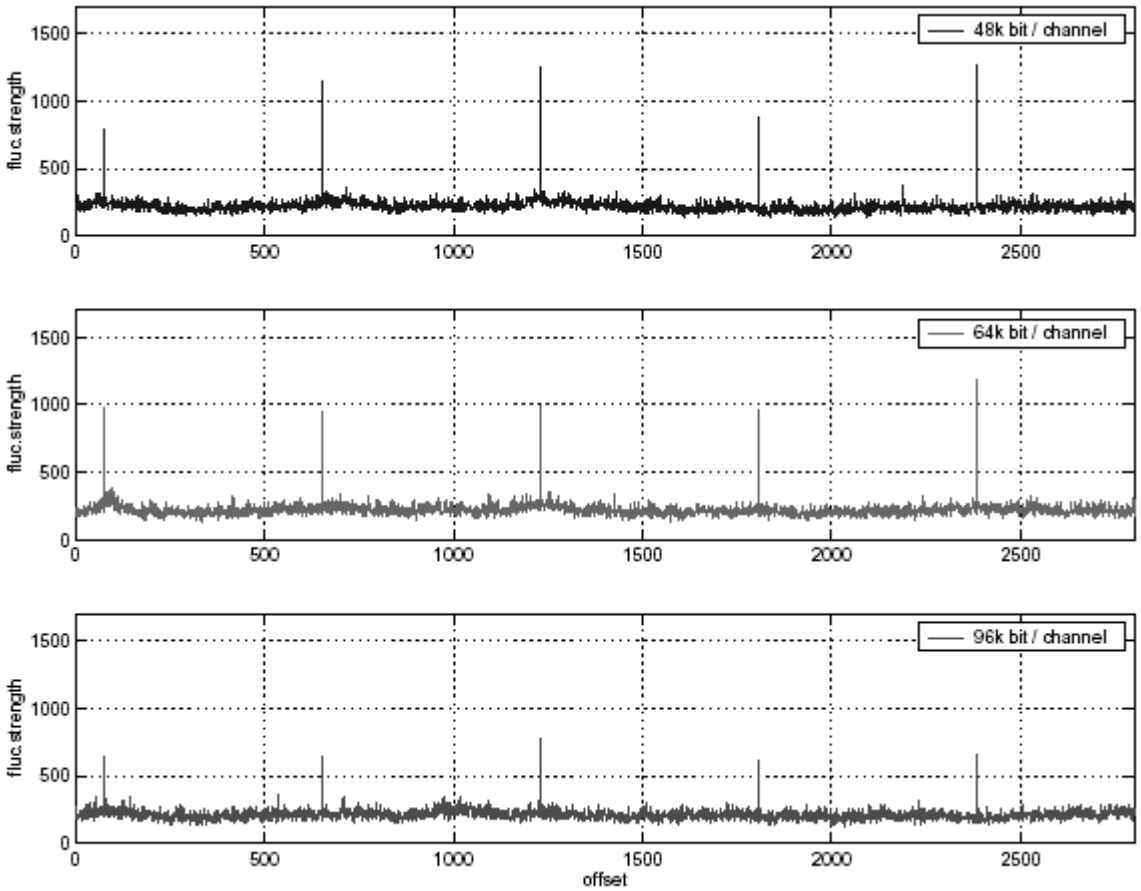

Figure 3.8 Framing recovery [10]

Once the correct framing grid is recovered, the inverse decoder attempts recovery of specific analysis filterbank parameters such as window types, sizes, and so on. An MP3 coder, as is typical for most modern codecs, has built-in adaptive windowing methods to reduce unwanted pre-echoes that arise from the time/frequency resolution tradeoff which exists due to block size choice. To proceed, the correct window type must be deduced, using a similar procedure as before. If each of the window types are applied to the spectral information, only in the correct window type will the characteristic trough template be retained.

In Figure 3.9, the analysis is carried out for each of the four window types. Visually, notice how only the “start” window (second from top) spectral pattern matches

the characteristic trough pattern obtained when correct framing synchronization is achieved. The algorithm itself would have to perform some sort of pattern matching technique in order to automatically recognize which is the correct decomposition.

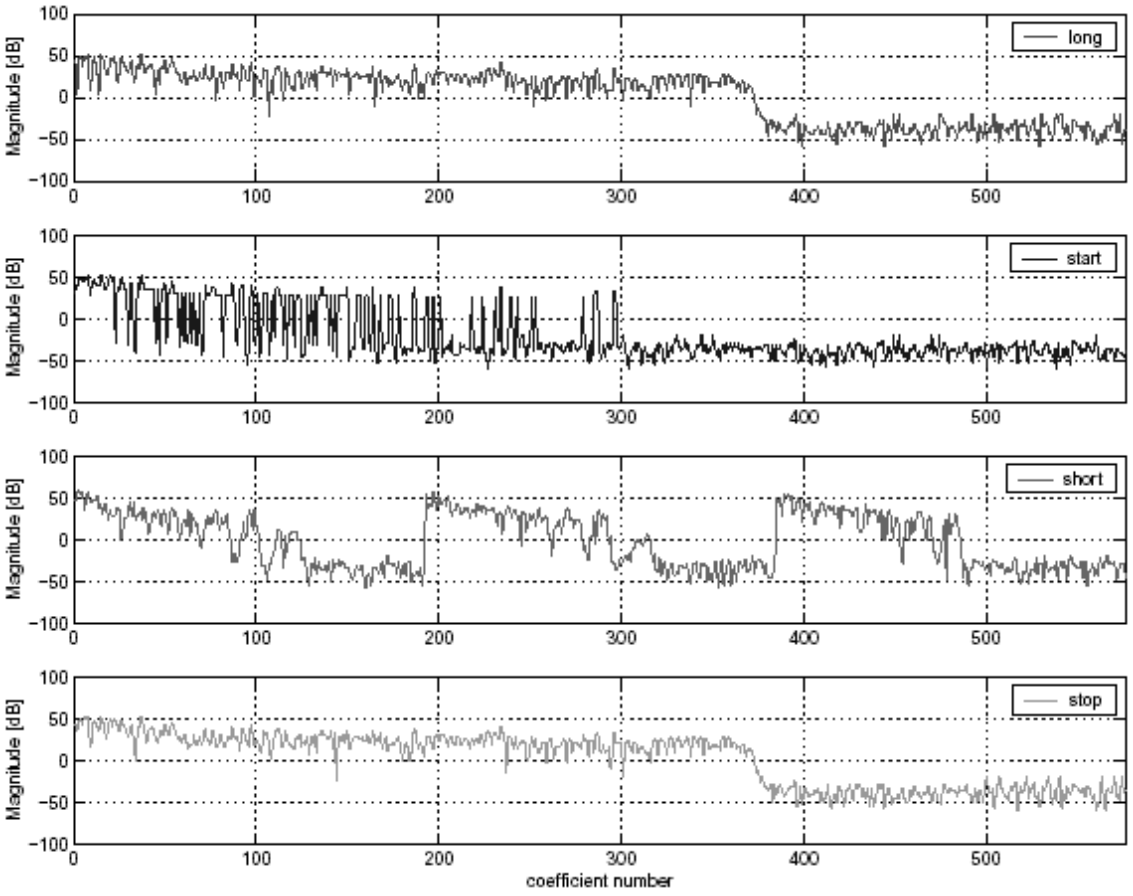

Figure 3.9 Long, Start, Short, Stop windows applied to spectral decomposition [10]

At this point, the correct spectral information is available to the algorithm, but the quantization information must be extracted somehow from the signal itself. To do this, the inverse decoder first estimates the step size of the quantizer. It does this in one of two ways which are applied according to how much energy is contained in the current block compared to how much white noise was introduced to the signal due to round-off error when presented to the inverse decoder. This separates each block into high energy content blocks and low energy content blocks. For the high energy case, the inverse

decoder uses a “greatest common denominator” method to retrieve the quantization step size; for the low energy case, a “cost minimization” technique is used. Figures 3.10 and 3.11 demonstrate the difference between these so-called high energy and low energy signals. In Figure 3.10, there is high energy in the signal and therefore less relative error compared to the low energy case as seen in Figure 3.11. The peaks and troughs are more prominent in the high energy case, which can use a less-complex technique to estimate the quantizer step size.

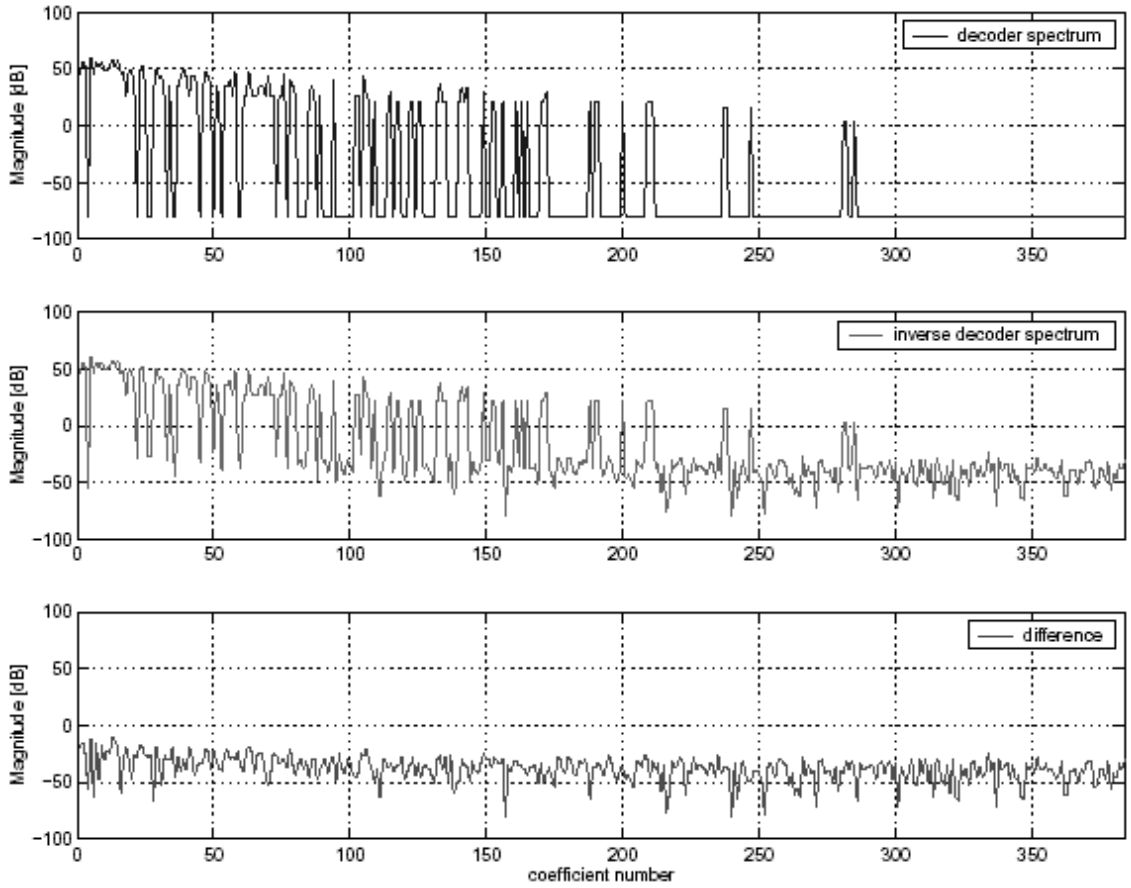

Figure 3.10 High energy frame [10]

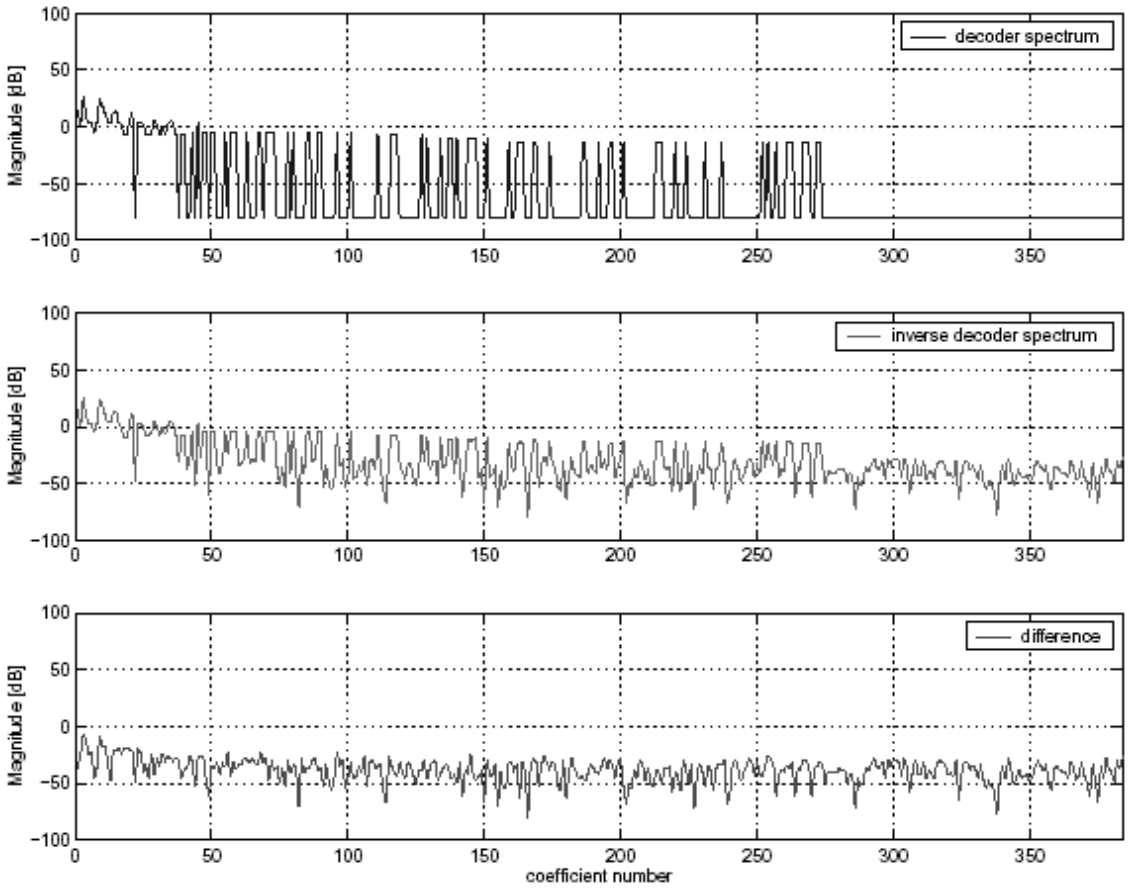

Figure 3.11 Low energy frame [10]

If a block has been classified as high energy, the inverse decoder more readily is able to estimate the quantization information. It makes an informed guess as to the quantizer step size by assuming the greatest common divisor (gcd) for one scalefactor band for most cases. In low energy blocks, there is more relative error and a different method is used which is more costly. These two different methods are applied to accurately recover the step size information – the 'greatest common divisor' method for high energy signals and the 'cost minimization' method for the low energy signals. For details of these methods, see [10].

Herre has shown that such an inverse quantization is possible [10]. The inverse decoder relies on recovering as much as possible from the decoded PCM samples based upon the presumption that an encoder enforces a certain structure on the output bitstream.

Herre shows that by using the proposed inverse decoder, multiple generations of audio coding stages are inherently more resistant to tandem coding losses. See Appendix B for tables summarizing the results of the inverse decoder, and a short discussion.

In [10], Herre showed that by using an inverse decoder, even after three encode-decode generations, there is little considerable loss in perceptual quality compared to a first generation encode-decode stage. This method relies on a method that recovers the original quantization information from the decoded signal only.

3.4 Summary

These research efforts underscore the following tenets of maintaining audio quality through multiple generations of audio coding:

- Recovery of framing synchronization.
- Recovery of quantizing information.

Each of these studies has arrived at these basic principles through separate means. It seems necessary that any coder wishing to retain perceptual quality should obey these principles.

Chapter 4: Proposed Method

In Chapter 3 it was shown that if framing synchronization and quantization information could be sent along with the encoded bitstream, subsequent encoding generations would be improved in perceptual quality compared to a naïve encoder. Each previous attempt has dealt with this problem in various ways. The MOLE signal proposes sending all of this information along in side information. Kurth's method buries this information in the decoded PCM samples and detects the embedded data in the encoder. The inverse decoder attempts to deduce this information from the decoded PCM bitstream itself. All of these methods rely on sending or extracting all of the quantization information to achieve better sounding future generations of coded audio.

Perhaps there exists a means of achieving the same goal of perceptual quality retention while sending less information. Might a coder which does not require the need for carrying the entire explicit quantization data also be able to achieve improvement? In this chapter, a method will be proposed to do that.

This project attempts to determine if perhaps a similar improvement in the perceptual quality of cascaded audio coders can be achieved by using a trimmed down version of earlier efforts. While very useful for large-scale projects, these methods are complex, and therefore, costly. They involve transmission or burying of monolithic side information, or require the need for reverse engineering that same monolithic information out of the decoded file. There is niche that has not been filled by any of these other means. A method using a slimmed down version of the encoder information can be more easily adapted or built into the design of an encoder which is resistant to multiple generation artifacts.

4.1 Description of the Baseline Codec

As noted in Chapter 3, it is difficult to discuss all-encompassing properties of audio coders due to the abundant types and various options of codecs available. Block sizes, filterbank types, and psychoacoustic models are not standard. Therefore, in this section, some specific properties of the baseline codec used will be briefly described before describing the techniques used to limit tandem coding errors.

The basic codec chosen was one developed in MATLAB by Jon Boley of the University of Miami [1]. A full MATLAB codec was useful for this research since MATLAB's development environment and visualization tools would be a necessity. Hence, the MATLAB codec was chosen over freely available, C-language audio coders such as Ogg Vorbis or LAME. The codec is briefly described as follows:

- Mono channel.
- Uses an MDCT length of N=2048 (fixed), with a sine window.

$$(2)\ w(n) = \sin\left[\pi \cdot \frac{n + 1/2}{2N}\right], \text{ for } 0 \le n \le N-1 \quad [27]$$

- At 44.1kHz, each audio frame of 2048 samples ~ 46 ms.
- Based on psychoacoustic model 1 from MPEG.
- Uses Schroeder spreading function to build the masking curves.
- Normalizes amplitude by using a 1kHz full amplitude tone to equal 96 dB SPL.
- Uses a 'water fill' bit allocation technique, each bit used reduces SMR in a particular subband by 6 dB until quantizing bits run out or SMR is completely covered.

Some modifications had to be made to the coder. The codec (originally mono) was modified to be a dual channel mono coder – each L-R stereo pair creating its own output stream so that stereo listening tests could be performed. Other inner workings of the encoder had to be modified on an as-needed basis. For instance, internal data passing, creation of new variables, and overall operation of the codec were all modified. There were several other code modifications which need not be discussed at length here. Also, various optimizations were performed on the codec to speed up execution time.

4.2 Early Experimentation / Enhanced Bit Allocation

An attempt was made to re-create the specific problems cited by Kurth, Fletcher, et al. – the problem of varying quantization coefficients (and possible decreasing energy according to Kurth) for each subband, even when framing synchronization is maintained across multiple coding generations [7, 13]. It should be noted that frame synchronization is inherent in the structure of this encoder over multiple encode cycles. An option to offset the incoming audio samples for each encode cycle was added to simulate a general case coder if needed. However, this option was not used in the following procedure described.

To see what happens to a typical encoder over multiple generations, a sound file was put through several orders of cascading. At each stage of the encode cycle, MATLAB was used to display the bit allocation for a fixed frame of audio data over the lifetime of the encode generations. It is the bit allocation which must be analyzed, since the proposed encoder will not have access to the full information of quantization parameters. Figures 4.1-4.5 show the bit allocations resulting from several cascading audio encoder stages of a sound file named Sound1.wav.

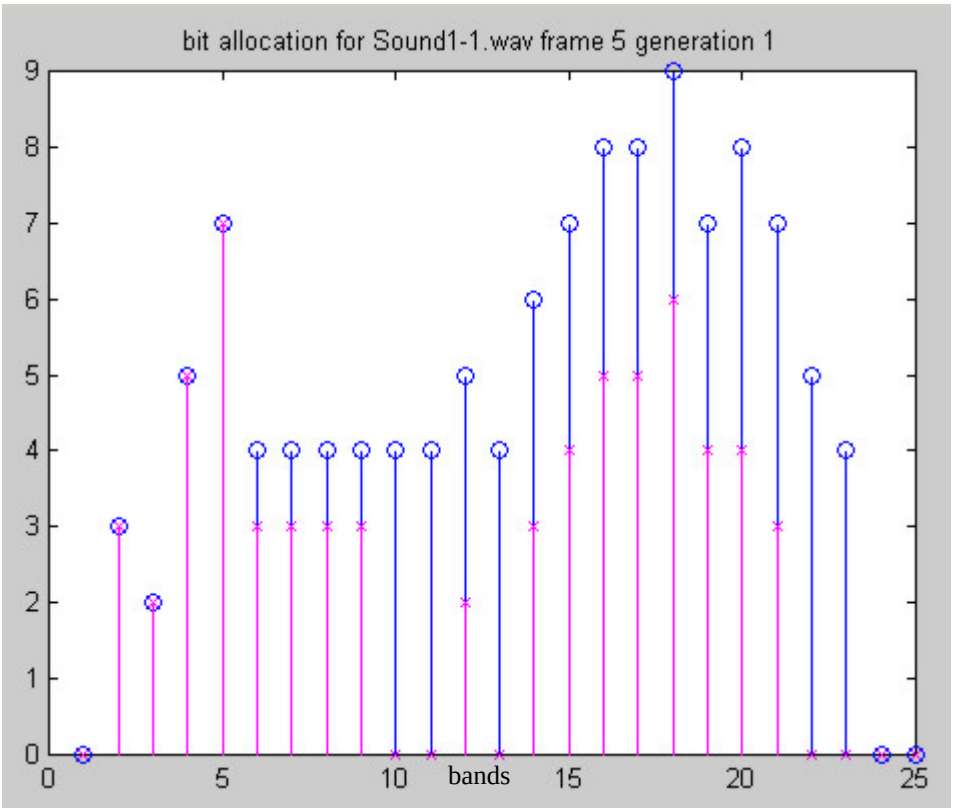

Figure 4.1. Bit allocation for first generation

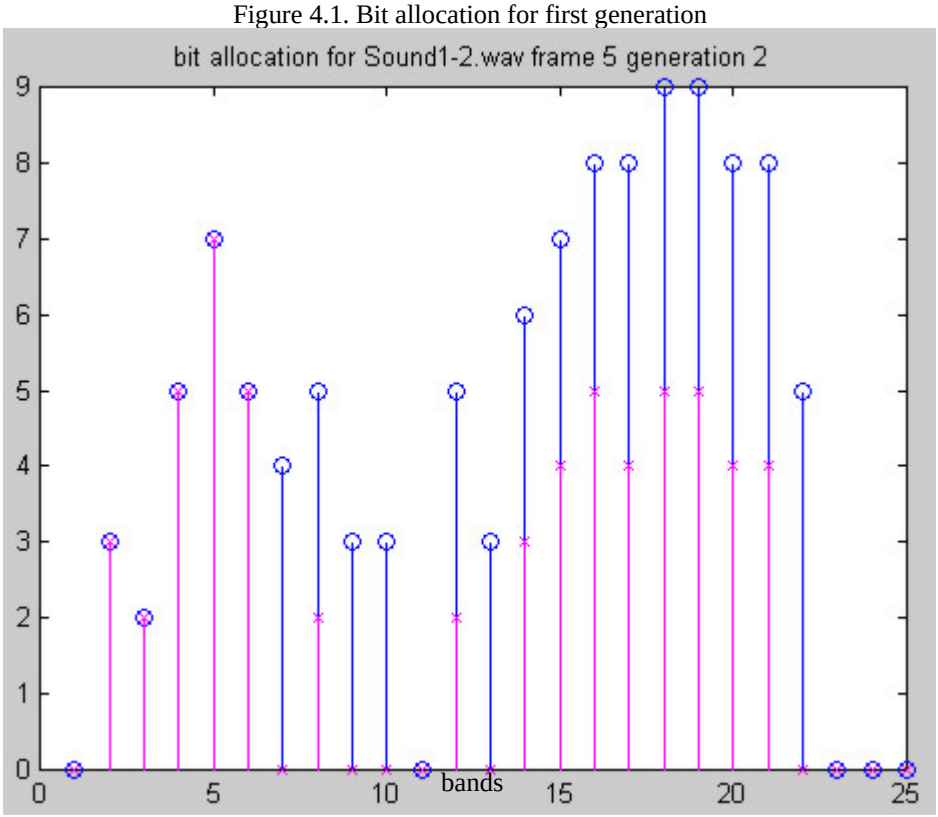

Figure 4.2 Bit allocation for second generation

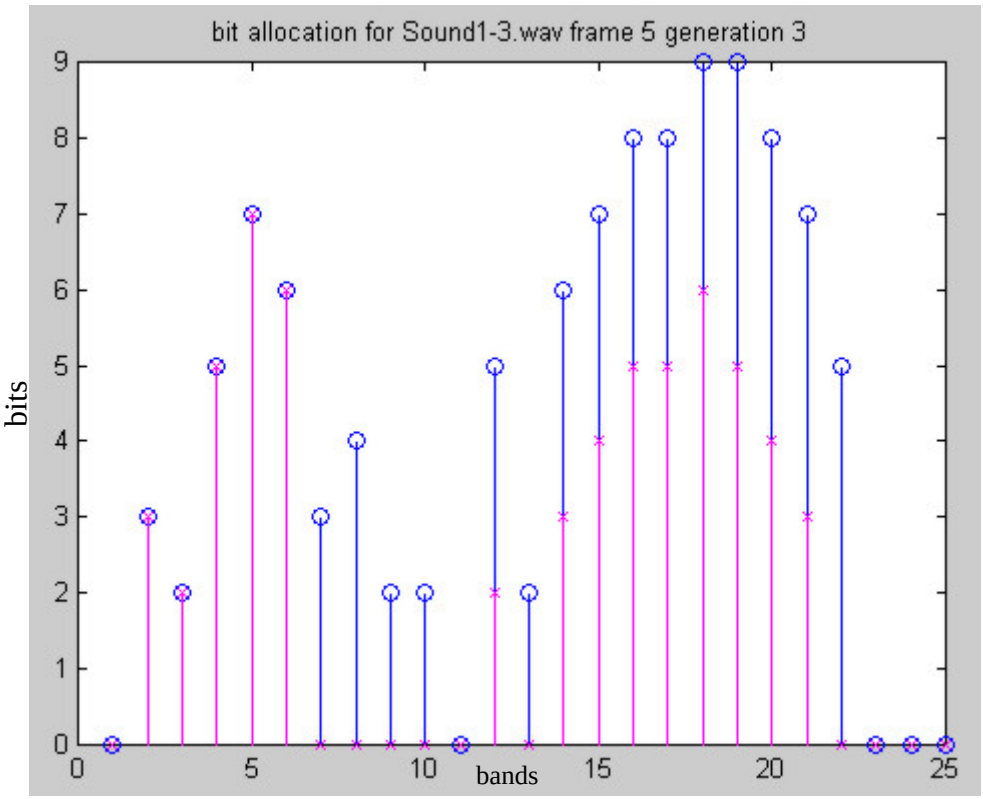

Figure 4.3 Bit allocation for third generation

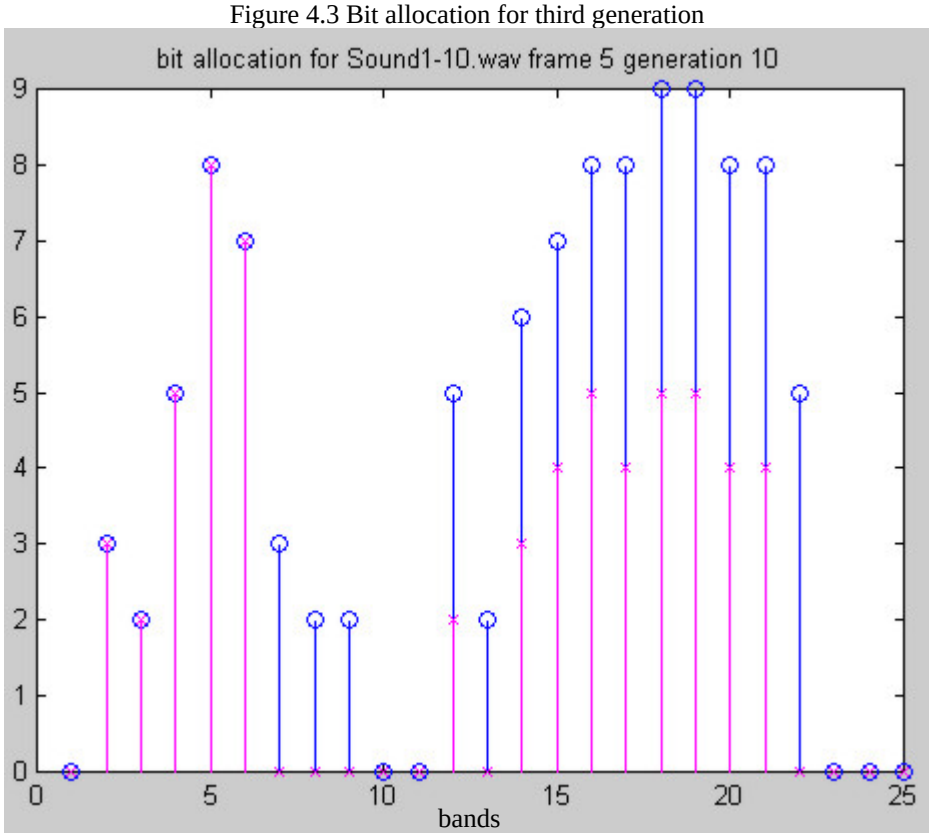

Figure 4.4 Bit allocation for tenth generation

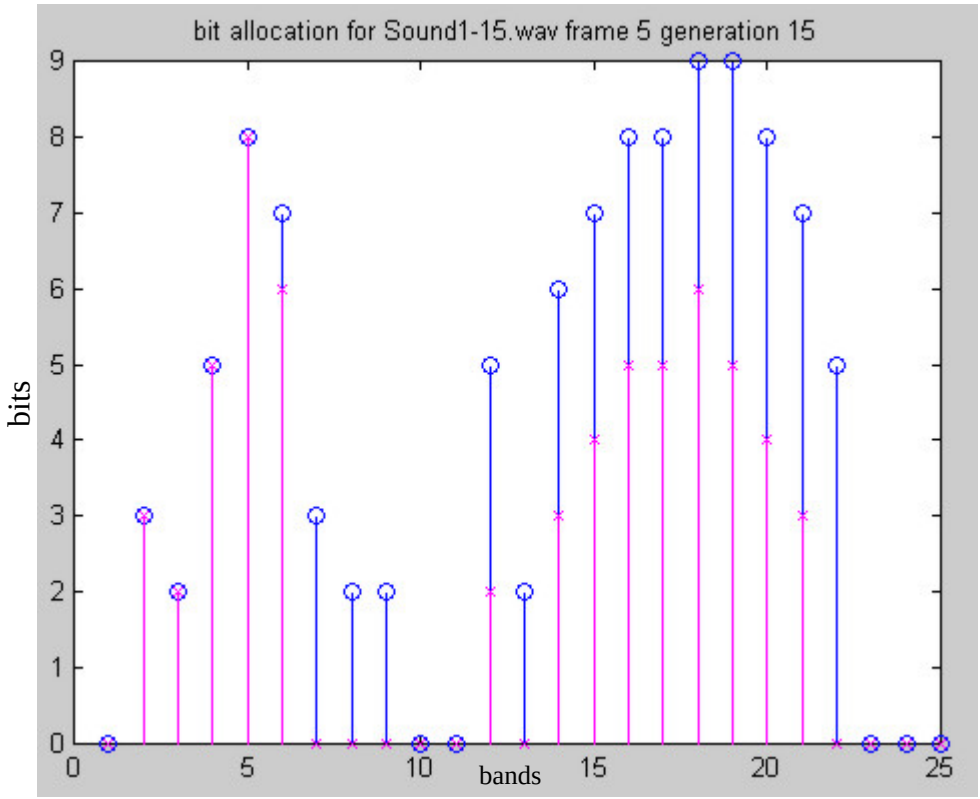

Figure 4.5 Bit allocation of fifteenth generation

In the figures, the circles represent the ideal bit allocation necessary to code the entire frame of audio data without perceptual error. The x's denote the actual bits calculated in the bit allocation routine used to represent that signal. The x-axis shows the perceptual subbands in a Bark scale; the y-axis shows the number of bits chosen by the encoder to represent the MDCT coefficients in each particular Bark subband or critical band. From these figures, it is demonstrable how the quantization parameters can change over time, even when framing synchronization is maintained. For instance, from generation 1 to 2 it is apparent that subband 7 which was originally coded with three bits was chosen to not be coded at all in the second stage, despite the need for bits. There are other anomalies as well in other Bark subbands, and all of these discrepancies will cause distortions which cannot be reversed down the line of generations. Even if the quantizer returns to the

original amount of bits used to encode as the second generation encode, it will still cause error if, in between generations, a different bit allocation was chosen in a preceding stage. Table 4.1 summarizes the results of all 15 generations.

Table 4.1 Bit allocations for all 15 generations of sound1.wav

		<- Bark subbands ->																								
Generations		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
	1	0	3	2	5	7	3	3	3	3	0	0	2	0	3	4	5	5	6	4	4	3	0	0	0	0
	2	0	3	2	5	7	5	0	2	0	0	0	2	0	3	4	5	4	5	5	4	4	0	0	0	0
	3	0	3	2	5	7	6	0	0	0	0	0	2	0	3	4	5	5	6	5	4	3	0	0	0	0
	4	0	3	2	5	7	6	0	0	0	0	0	2	0	3	4	5	4	5	5	4	4	0	0	0	0
	5	0	3	2	5	8	6	0	0	0	0	0	2	0	3	4	5	5	6	5	4	3	0	0	0	0
	6	0	3	2	5	8	7	0	0	0	0	0	2	0	3	4	5	4	5	5	4	4	0	0	0	0
	7	0	3	2	5	8	6	0	0	0	0	0	2	0	3	4	5	5	6	5	4	3	0	0	0	0
	8	0	3	2	5	7	6	0	0	0	0	0	2	0	3	4	5	4	5	5	4	4	0	0	0	0
	9	0	3	2	5	8	6	0	0	0	0	0	2	0	3	4	5	5	6	5	4	3	0	0	0	0
	10	0	3	2	5	8	7	0	0	0	0	0	2	0	3	4	5	4	5	5	4	4	0	0	0	0
	11	0	3	2	5	8	6	0	0	0	0	0	2	0	3	4	5	5	6	5	4	3	0	0	0	0
	12	0	3	2	5	8	7	0	0	0	0	0	2	0	3	4	5	4	5	5	4	4	0	0	0	0
	13	0	3	2	5	8	6	0	0	0	0	0	2	0	3	4	5	5	6	5	4	3	0	0	0	0
	14	0	3	2	5	8	7	0	0	0	0	0	2	0	3	4	5	4	5	5	4	4	0	0	0	0
	15	0	3	2	5	8	6	0	0	0	0	0	2	0	3	4	5	5	6	5	4	3	0	0	0	0
#errors		**0**	**0**	**0**	**0**	**3**	**10**	**1**	**2**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**14**	**14**	**0**	**0**	**14**	**0**	**0**	**0**	**0**

It is apparent that in subbands 6, 17, 18, and 21, several quantization errors have built up over the lifetime of the cascade. In other bands, there is no loss at all. This information pertains to one fixed frame of audio data. It may be dangerous to draw too many conclusions from this table alone, since the nature of the input signal may have some sort of effect. For example, Table 4.2 lists four generations of a different input file to demonstrate this point. Notice how in the table the errors take on a different structure than those in Table 4.1.

Table 4.2 Bit allocation for 4 generations of file excerpt.wav

		<- Bark subbands ->																								
Gens		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
	1	2	4	5	2	2	4	6	5	4	3	4	4	4	3	3	3	4	4	4	4	3	0	0	0	0
	2	3	3	5	3	3	0	3	2	2	3	3	3	3	3	3	3	3	5	5	5	3	0	0	0	0
	3	3	3	5	3	3	3	5	0	0	2	3	3	3	3	3	3	3	5	5	5	3	0	0	0	0
	4	3	3	5	3	3	3	5	0	0	3	3	3	3	3	3	3	3	5	5	5	3	0	0	0	0
#errors		**1**	**1**	**0**	**1**	**1**	**2**	**2**	**2**	**2**	**2**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**

It has also been noted by Kurth [13] that some signals display or settle into a “steady state” over multiple generations, which tends to minimize the cascading audio loss. If there are no changes in the amount of bits allocated to a particular subband (assuming frame synchronization), the reconstructed audio samples will be identical to their previous values. As demonstrated earlier, if a 5-level quantizer follows another 5-level quantizer, no further signal degradation occurs. When the quantizer chooses different values for each subband are errors introduced to the signal.

4.3 Constraining the Bit Allocation

From this experimentation, and using lessons learned from other research, it was determined that a possible way to combat these errors is to consider a bit allocation modification that could minimize the errors with a minimal amount of side information. The particular method chosen to convey the data was selected to send extra side information similar to the MOLE signal approach. This research’s goal, however, is to achieve similar or approachable results to the MOLE with less information or see some improvement in perceptual audio quality.

The particular bit allocation used in this codec iterates through each Bark scale band, and allocates a bit to the Bark scale subband which is determined “most needy” earlier via an SMR measurement. It then reduces the SMR appropriately (using a 6dB/bit assumption). Continuing on, the bit allocation iterates through all the Bark scale subbands in this way until it is either out of bits or the signal is completely brought under optimal SMR masking thresholds. Since it is useless to code a subband with one bit, any subbands left with one bit were freed up and filled into the lowest subbands first.

One idea considered was to send an extra “analysis word” of 25-bit length which would contain a flag for each Bark subband. This would add very little overhead to the overall side information bit stream when compared to a MOLE signal, for example, which adds approximately 726 bits per audio frame (varies). However, with merely one bit of flag information, there is little that can be done. Even if a band is earmarked in some way, it is difficult to make any judgments about what the new bit allocation should do with this earmarked subband. Since not all Bark scale bands get all the bits they need all the time, it would not help to simply insist on an earmarked band receiving all the bits needed for allocation.

With two “analysis word” flags, however, possibilities begin to manifest. There are a number of schemes to help constrain the bit allocation. Something noticed after experimentation is that on occasion the subbands which were allocated zero or two bits for their MDCT coefficients at times would receive varying bits due to the fact that if a subband were initially determined to receive one bit, it would be discarded and receive zero bits. However, in future encoding generations, perhaps errors have appeared in these same subbands and it would again receive two or more bits from the bit allocation routine. If these fluctuations could be fixed, improvement in future generations can be more controlled.

Also, it was realized that instead of interpreting each 25-bit analysis word as an independent word of earmarked flags per subband, each word could be considered as one bit of a two bit value able to encode four different options for each subband. At this point, more possibilities become available.

With this in mind, the following procedure was conceived to provide the basis of the encoder:

- One 25-bit word of bark subband flags which notify any bands receiving zero bits or three bits from the bit allocation routine.
- One 25-bit word of bark subband flags which notify any bands receiving two bits or three bits from the bit allocation routine.
- The two 25-bit words would be sent as side info along with each generation of encoders, to be written (generation one) or read (all others) as necessary by the encoder.
- At the decoder, each analysis word can be compared, and any subbands having both bits set in similar positions will be interpreted as a "3." The position can then be cleared so that the subbands receiving zero and two bits can be read directly.
- In this way, three different subband values can be fixed for future encode stages.
- All other subbands are considered "free." They will be allocated just the same amount of bits as they would be allocated by the normal bit allocation procedure, after all fixed bit allocations are accounted for.

Figure 4.6 illustrates the operation of the encoder with the analysis word. Note that in the figure, each successive generation of encoder reads and writes a new analysis word. The write operation that each encoder performs is writing a copy of the original. Each new analysis word will contain the same information as the last.

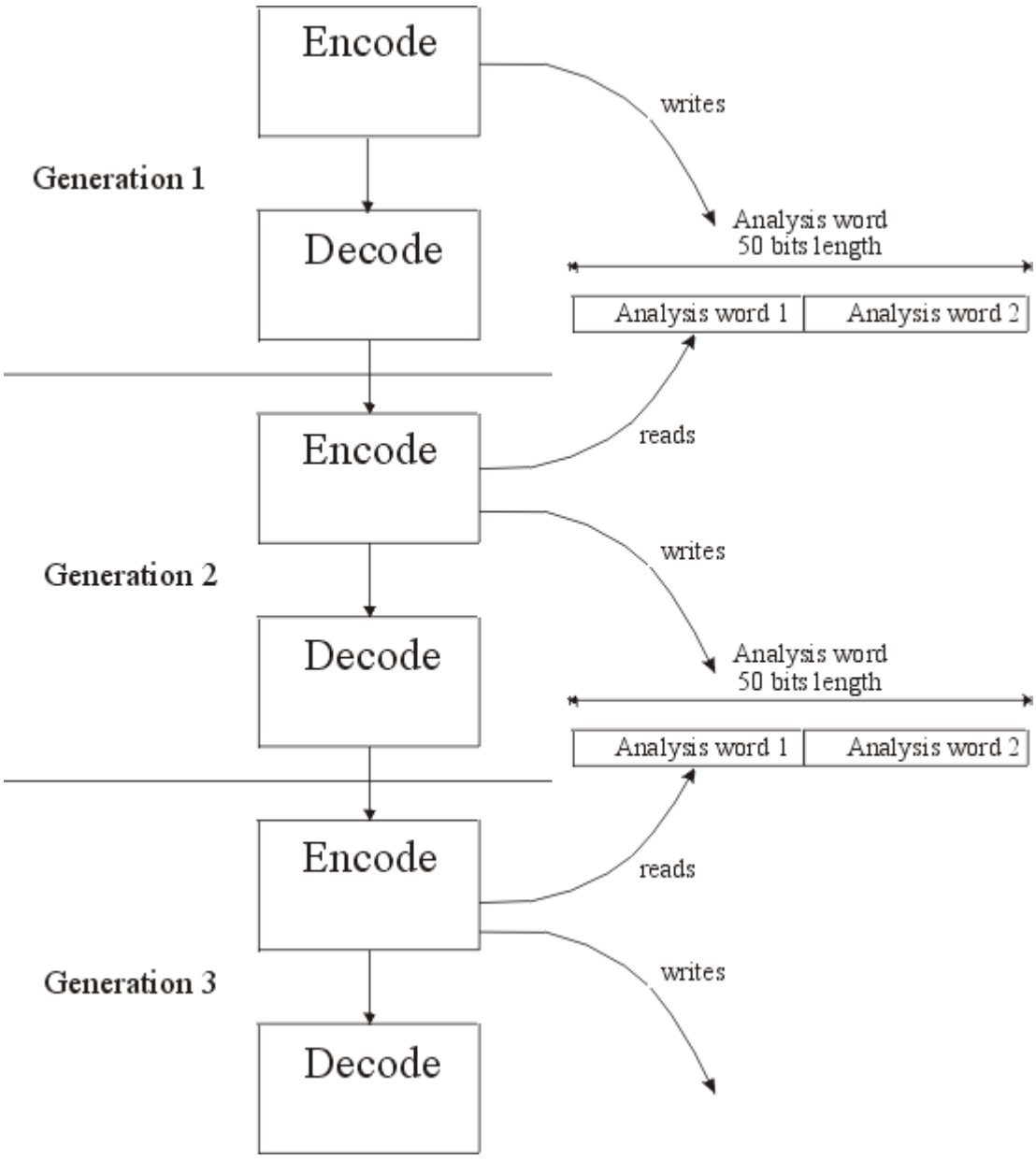

Figure 4.6 Operation of the codec

Table 4.3 displays the resulting code words created and how they are interpreted by the enhanced encoder for each subband. Z_word and T_word are the names of each 25-bit analysis word used by the encoder.

Table 4.3 Analysis word scheme

Z_word	T_word	Interpretation
1	0	Fix bit alloc at 0
0	1	Fix bit alloc at 2
1	1	Fix bit alloc at 3
0	0	Free allocation

The extra overhead is fairly small when compared to a MOLE signal. At a most conservative approach, this method uses at most half of the information that the MOLE uses, if only bit allocation information is considered. Although the MOLE is specified for MPEG-1 Layer 2 and must deal with 12-subsample granules, for the purposes of this comparison the MOLE data sends all explicit bit allocation. For each frame of incoming audio data, the proposed method creates an extra 50 bits of side information, compared to approximately 726 bits for the MOLE signal in its full use. It is perhaps not quite accurate to do a direct comparison since a portion of the MOLE signal is devoted to framing synchronization, but even if just bit allocation information is considered, this method uses one half the amount of information retain audio quality. The proposed method uses an analysis word of 50-bits width per audio frame of 2048 input samples, creating an overhead of 1.076 kbps at 44.1 kHz. By comparison, the MOLE signal creates (at 726 bits for an audio frame of 1152 input samples at 44.1 kHz) 27.792 kbps of overhead.

Chapter 5: Results and Discussion

In any type of audio coding research, it is ultimately left up to the subjective nature of human perception to determine the perceptual quality. An objective measure would be ideal and is being worked towards (the PEAQ tool – Perceptual Evaluation of Audio Quality), however, the most robust method to determine audio quality still remains the subjective listening test [12].

Subjective scores on listening tests are affected by the level of expertise of the subjects, the test methodology, the listening environment, the quantity of the training provided, and type and severity of the impairments to be assessed [2]. To test the proposed concept, the following factors were taken into consideration:

- A large number of generations were created. While the enhanced bit allocation appeared to be a solid approach in theory, there still remained a chance that only after several codec generations would an audible difference be noted.
- Discrimination between the naïve and aware encoders would be enhanced by choosing more aggressive (lower) bit-rates, due to the fact that the bit allocation would fix more subbands at 0, 2, and 3 bit decisions. At higher bitrates, it might be more likely to see fewer (on average) fixed bit allocations.
- Audible artifacts in the coder (for one stage of operation) would be tolerable, and perhaps even beneficial to providing or amplifying discriminations among codec generations. This perhaps would reduce the number of generations required to see improvements or differences.
- Audio material selected must be chosen carefully since listening tests are time consuming and difficult in general.

According to [1], the codec was tested subjectively and determined to be 'transparent' at 256 kbps mono with respect to CD-Audio. For the tests, the codec was chosen to operate at stereo (dual channel) bit-rates of 128 kbps and 96 kbps, corresponding to mono bitrates of 64 kbps and 48 kbps mono. It would be expected that subjective scores would be quite low at these bit-rates, however, it is not the overall quality of the subjective score that is of concern. It is more important to see a difference between the aware and naïve versions of each codec of generation *n*.

The Subjective Difference Grade (SDG) measure is obtained by having listeners score each file using a Mean Objective Score (MOS). The MOS scores were used to compare the results of files created through multiple generations of encoder stages. The MOS grades are as follows:

5.0 – Excellent

4.0 – Good

3.0 – Fair

2.0 – Poor

1.0 – Bad

After the MOS scores are obtained, the MOS score of the unencoded version of the audio clip is subtracted from the MOS score of each audio file under test to obtain a SDG score.

For each source file chosen at a given bitrate, three "trees" of encoder generations were compared. The original uncompressed file was also scored to provide a proper experiment control. Each "tree" was comprised of ten generations of cascaded audio cycles of the following configurations:

- A naïve encoder with no frame synchronization: referred to with the three letter abbreviation *off* (an offset configuration).
- A naïve encoder with frame synchronization and a standard bit allocation: referred to with the three letter abbreviation *nav* (a naïve configuration).
- An aware encoder with frame synchronization and an enhanced bit allocation as described in Chapter 4: referred to with the three letter abbreviation *awr* (an aware configuration).

The final result is that for each audio source selected at a fixed bitrate, 30 files must be created, ten for the case with offset, ten for case of naïve encoder with frame synchronization, and ten for the case of the aware encoder with frame synchronization and enhanced bit allocation.

For each tree or family of cascaded audio, only generations three, six, and ten were compared. This provides a distributed spacing in encoding generations. In addition to the source file, this resulted in ten files per set for a listener to grade and/or compare. Due to this relatively large number of samples, it was decided that four sets of ten would be a fair compromise among the competing priorities of obtaining accurate listening results (avoiding listener fatigue), and providing enough variety of source material in terms of both original material and bitrates.

There were four original source files of two different audio sources encoded at different bitrates. The stereo bitrates chosen were 128 kbps and 96 kbps. The two audio files chosen were a “castanets” clip for its transient content and a “French horn” clip for its harmonic content; each provides a different set of challenges for the audio encoder.

The decoded PCM files for each generation were written to a .wav format and listeners were asked to provide MOS scores for each set of ten audio clips.

5.1 Test Procedure

The test equipment was a Philips 8cm Expanium portable audio device capable of playing 8cm CD-Audio formatted disks. The headphones used were Sony MDR-CD380 headphones.

The audio tracks were ordered randomly within each data set. Table 5.1 shows the order of the tracks as presented to the listener.

Table 5.1 Track order for listening tests

French @96kbps	Castanets @96kbps	French @128 kbps	Castanets @128kbps
6nav	3off	10off	3off
10awr	3nav	6nav	-orig-
3awr	6off	-orig-	10nav
10off	6awr	3nav	3awr
10nav	6nav	3awr	10off
-orig-	3awr	10nav	6off
3off	-orig-	6awr	10awr
6awr	10off	3off	6nav
6off	10nav	10awr	3nav
3nav	10awr	6off	6awr

The listening test subjects were untrained subjects of varying ages. The range of ages of test subjects was approximately 22-70 years of age, with roughly one half to two thirds of all listeners under the age of 40. The remaining listeners were age 60 or over. The Subjective Difference Grade measures obtained for each condition (see Appendix D for full listening test results).

5.2 Analysis and Discussion – french horns @96kbps

The SDG measures for the french horns at 96kbps shown in Figure 5.1 display some interesting results. Surprisingly, listeners found the best overall case to be the tenth generation aware coder. This is contrary to expectations since it was expected that the third generation aware coder would achieve the best performance. Overall, the three best performing cases were the tenth, third, and sixth generation aware encoders, followed in order by sixth, tenth, and third generations of the naïve encoder. The three worst performers were the third, sixth and tenth generation offset. This is more in-line with expectations.

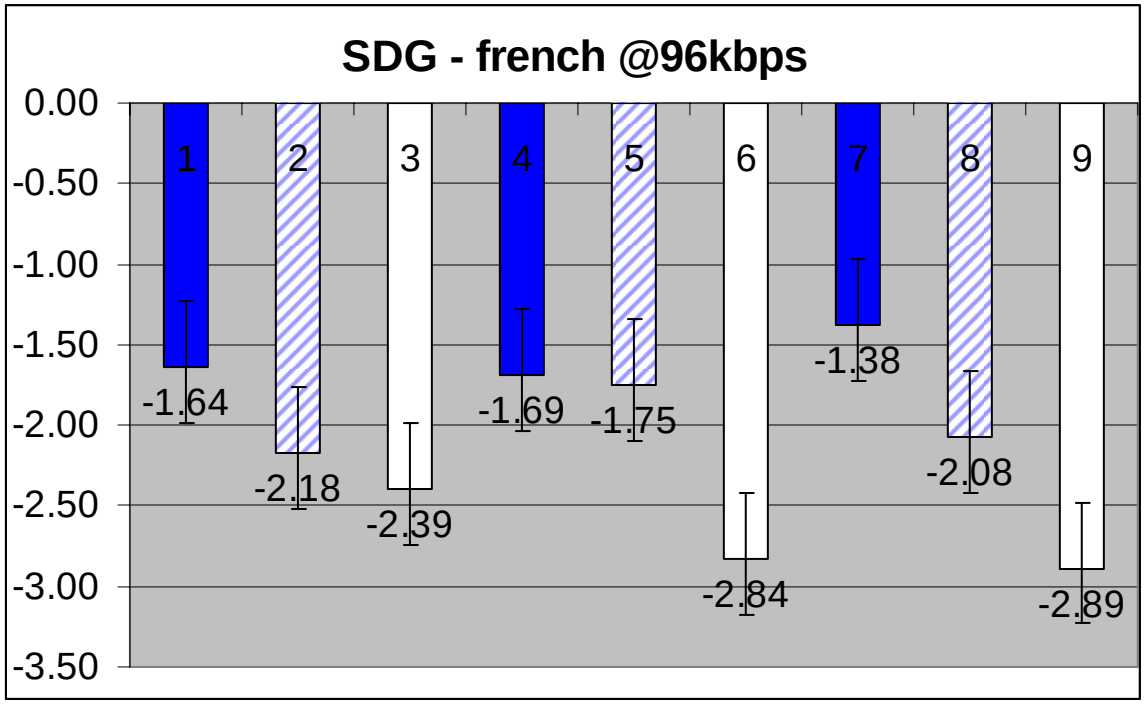

Figure 5.1 SDG Listening test chart for french horns @96kbps

Legend

1 – 3awr	2 – 3nav	3 – 3off
4 – 6awr	5 – 6nav	6 – 6off
7 – 10awr	8 – 10nav	9 – 10off

One possible explanation for the anomalous result is that with such an aggressive encoding, listeners found little difference between tenth and third generations of the aware encoder. A larger sample size (N=17 in all cases) might have alleviated this, since

there is moderate overlap of 95% confidence intervals among the best-performing encoders for this data set. Also, trained listeners may be better able to detect the presence or lack of artifacts. All of these or some combination of these factors could explain this result.

Comparing third generations only, the mean of the aware encoder outperformed the naïve encoder by a margin of 0.54 on the SDG scale, with 95% confidence limits slightly overlapping.

Comparing sixth generations only, the aware encoder and naïve encoder seemed to perform about the same, with an ever so slight advantage to the aware encoder.

Comparing tenth generations only, the mean of the aware encoder outperformed the naïve encoder by 0.70 on the SDG scale, with 95% confidence limits only ever so slightly overlapping.

Overall, all three aware encoded versions of the file occupied the top three scores.

This is a strong indication that the enhanced bit allocation is providing a benefit to the cascading audio configuration.

Table 5.2 shows the listening test results in numerical form. The mean of the MOS score, the standard deviation (spread), 95% confidence intervals, 90% confidence intervals, and mean of the SDG are listed for each case.

Table 5.2 Listening Test Results – french horns @96kbps

	3awr	3nav	3off	6awr	6nav	6off	10awr	10nav	10off
MOS score	3.15	2.61	2.39	3.10	3.04	1.95	3.41	2.71	1.90
Std dev	0.86	0.89	0.78	0.67	0.72	0.69	0.76	0.97	0.76
95% int	0.41	0.42	0.37	0.32	0.34	0.33	0.36	0.46	0.36
90% int	0.34	0.36	0.31	0.27	0.29	0.28	0.30	0.39	0.30
SDG	-1.64	-2.18	-2.39	-1.69	-1.75	-2.84	-1.38	-2.08	-2.89

Figure 5.2 shows a graphical representation of the bit allocation decisions through all ten generations for the naïve encoder. Figure 5.3 shows the same information for the aware encoder. Notice how in the aware encoder there are fewer changes to the bit allocation decisions over all the generations. This demonstrates the enhanced bit allocation fixing the quantization parameters as designed.

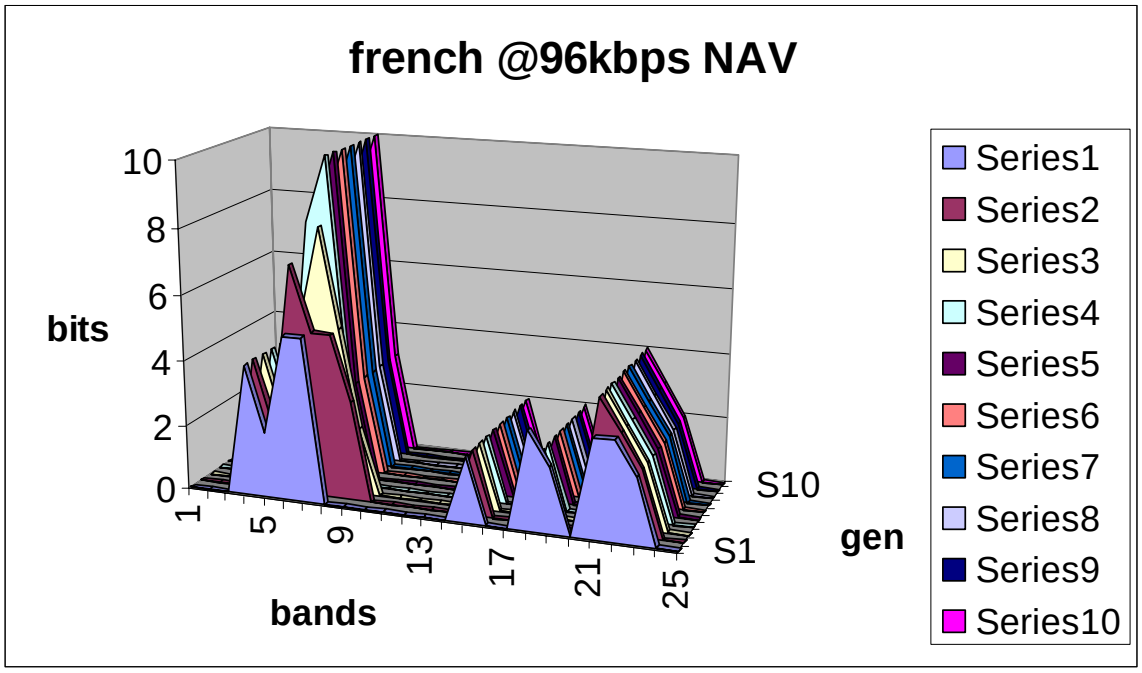

Figure 5.2 Bit allocations over all ten generations, naive french horns @96kbps

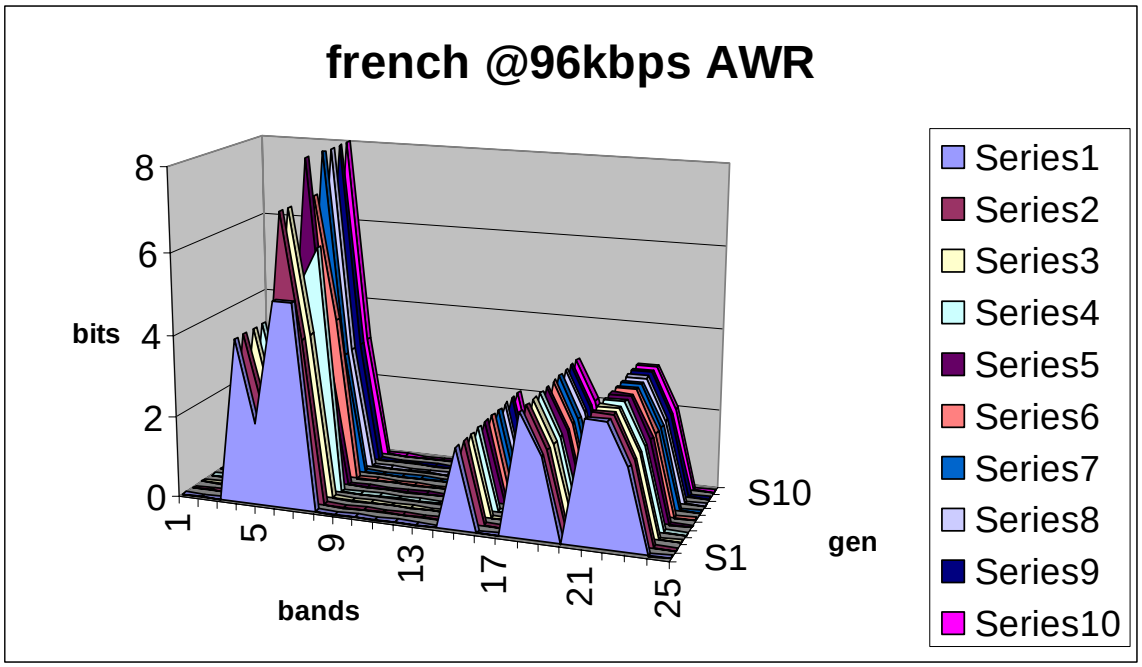

Figure 5.3 Bit allocations over all ten generations, aware french horns @96kbps

5.3 Analysis and Discussion – french horns @128kbps

For the case of french horns at 128kbps, the encoder exhibited fairly well-behaved listening data. Figure 5.4 displays the SDG scores for each cascade. The highest overall SDG was the third generation aware encoder, as expected. The best three performing cases were the third generation aware, third generation naïve, and tenth generation aware (followed closely by sixth generation aware placing fourth). The sixth and tenth generation naïve and third generation offset were the next logical grouping. The worst performing cases were the sixth, and tenth offset cases, as expected.

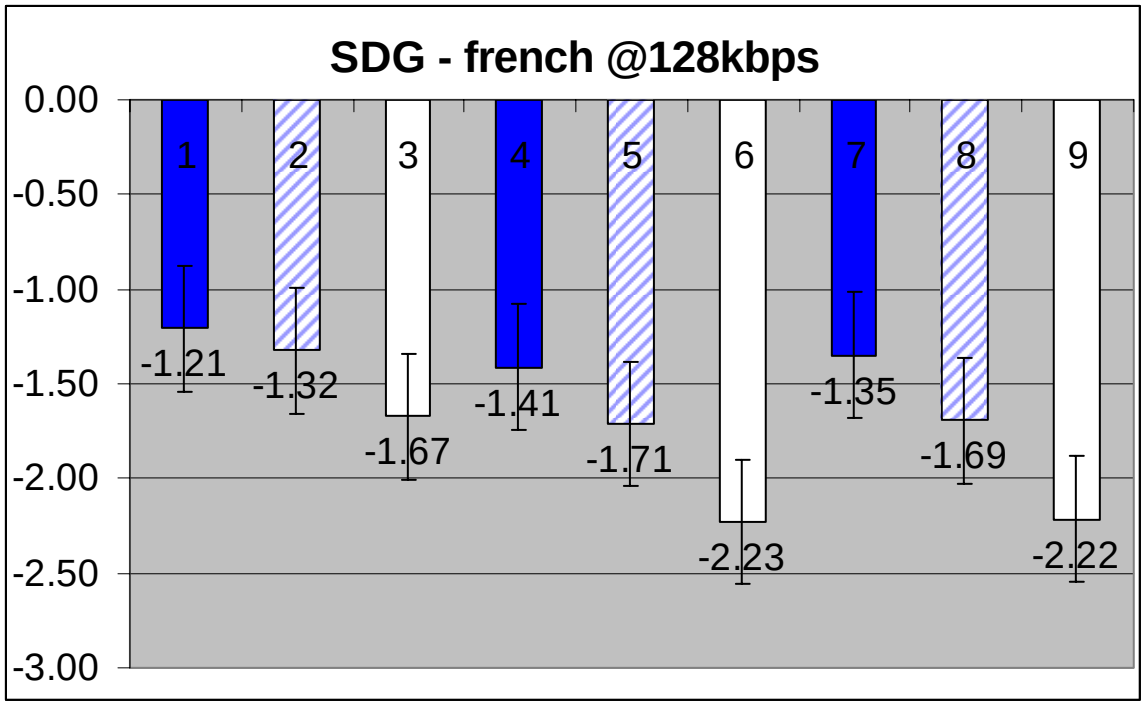

Figure 5.4 SDG Listening test chart for french horns @128kbps

Legend

1 – 3awr 2 – 3nav 3 – 3off
4 – 6awr 5 – 6nav 6 – 6off
7 – 10awr 8 – 10nav 9 – 10off

Comparing third generations only, the aware encoder mean SDG outperformed the naïve encoder by 0.11 SDG – this is a rather small increase, but an improvement nonetheless.

Comparing sixth generations only, the aware encoder SDG outperformed the naïve encoder by 0.30 SDG, with an approximate 50% overlap in 95% confidence intervals. This is a moderate indication that some improvement has been attained.

Comparing tenth generations only, the aware encoder outperformed the naïve encoder by 0.34 SDG, with another approximate 50% overlap in 95% confidence intervals.

Overall, the three aware encoded versions occupied three out of four of the top three scores.

This is a fairly strong indication that the aware encoder is providing a benefit in the case of a cascade configuration.

Table 5.3 shows the listening test results in numerical form. The mean of the MOS score, the standard deviation (spread), 95% confidence intervals, 90% confidence intervals, and mean of the SDG are listed for each case.

Table 5.3 Listening Test Results - french horns @128 kbps

	3awr	3nav	3off	6awr	6nav	6off	10awr	10nav	10off
MOS score	3.19	2.54	2.85	2.66	2.29	1.92	2.56	1.85	1.41
Std dev	1.15	0.99	1.12	1.01	0.98	0.79	1.07	0.81	0.71
95% int	0.53	0.47	0.38	0.48	0.46	0.55	0.14	0.34	0.39
90% int	0.45	0.40	0.32	0.40	0.39	0.46	0.12	0.28	0.32
SDG	-1.54	-2.18	-1.87	-2.06	-2.43	-2.81	-2.16	-2.87	-3.31

Figure 5.5 shows a graphical representation of the bit allocation decisions through all ten generations for the naïve encoder. Figure 5.6 shows the same information for the aware encoder. Notice how the aware encoder maintains more consistency through all ten generations of bit allocation decisions, especially through the later generations.

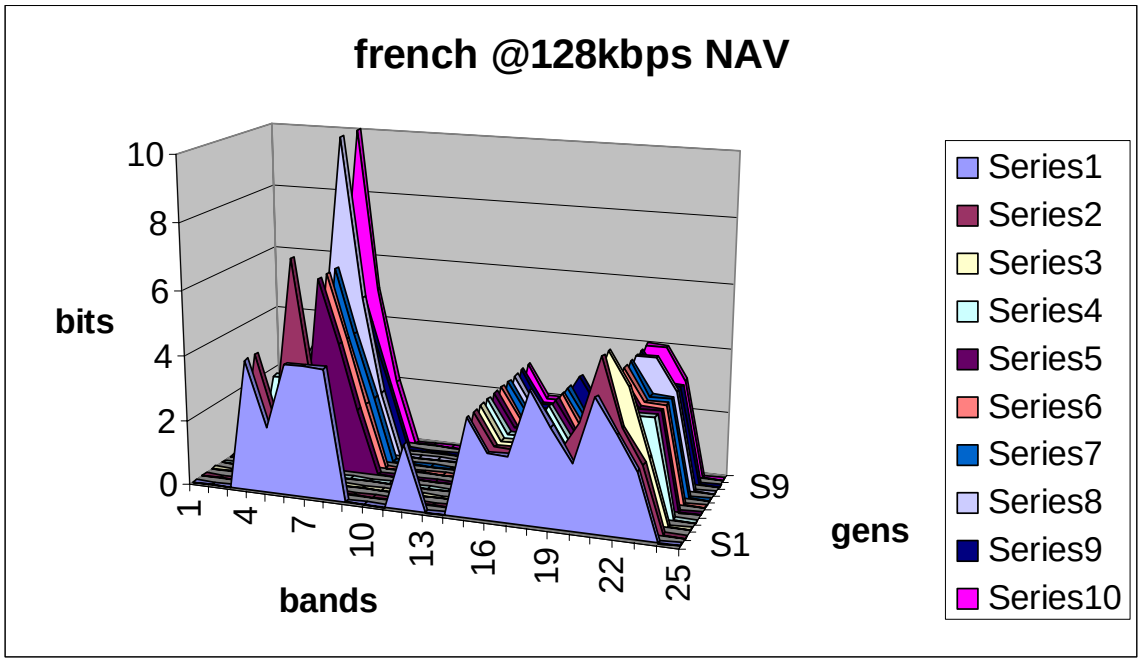

Figure 5.5 Bit allocations over all ten generations, naive french horns @128kbps

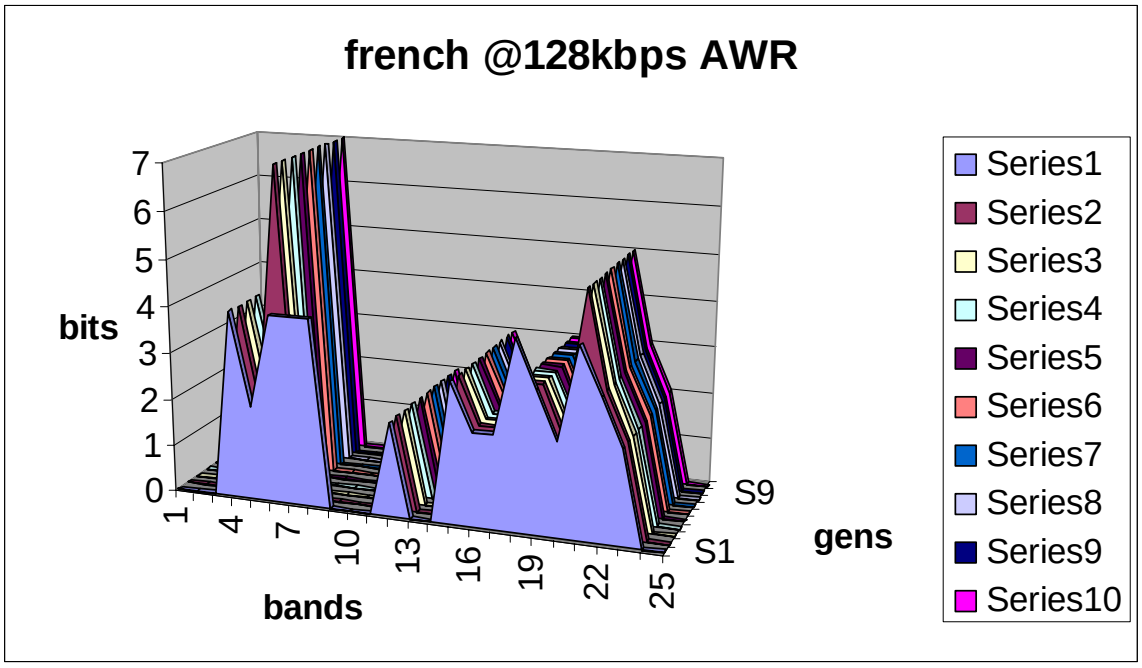

Figure 5.6 Bit allocations over all ten generations, aware french horns @128kbps

5.4 Analysis and Discussion – castanets @96kbps

This data set also exhibited fairly straightforward, expected results, with one minor exception. The top three performing cases were the third generation aware, third generation offset (the minor exception), and the sixth generation aware case. The tenth generation aware and third generation naïve encoder were ranked similarly. The bottom four scores in order were the sixth generation naïve and offset cases, and the tenth generation naïve and offset cases.

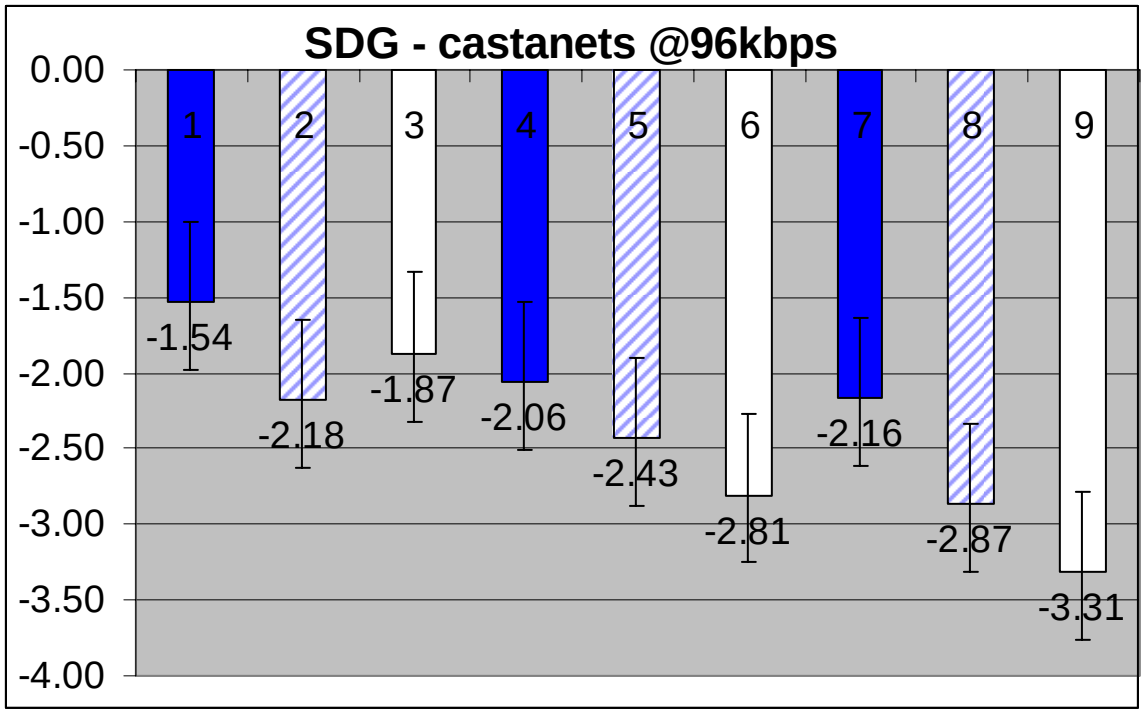

Figure 5.7 SDG Listening test chart for castanets @96kbps

Legend

1 – 3awr	2 – 3nav	3 – 3off
4 – 6awr	5 – 6nav	6 – 6off
7 – 10awr	8 – 10nav	9 – 10off

Considering only the third generations, the aware encoder outperformed the naïve encoder by 0.64 SDG, with approximately a 33% confidence interval overlap (of the entire 95% interval).

Considering only the sixth generations, the aware encoder outperformed the naïve encoder by 0.37 SDG, with approximately a 50% confidence interval overlap.

In the case of the tenth generations, the aware encoder outperformed the naïve encoder by a margin of 0.71 SDG with very little confidence interval overlap.

Overall, the three versions of the aware encoder were scored by the listeners as being in the top three out of four scores. The tenth generation aware encoder performed particularly well in this case.

Table 5.4 shows the listening test results for this case. The mean of the MOS score, the standard deviation (spread), 95% confidence intervals, 90% confidence intervals, and mean of the SDG are listed for each case.

Table 5.4 Listening Test Results - castanets @96 kbps

	3awr	3nav	3off	6awr	6nav	6off	10awr	10nav	10off
MOS score	3.19	2.54	2.85	2.66	2.29	1.92	2.56	1.85	1.41
Std dev	1.15	0.99	1.12	1.01	0.98	0.79	1.07	0.81	0.71
95% int	0.53	0.47	0.38	0.48	0.46	0.55	0.14	0.34	0.39
90% int	0.45	0.40	0.32	0.40	0.39	0.46	0.12	0.28	0.32
SDG	-1.54	-2.18	-1.87	-2.06	-2.43	-2.81	-2.16	-2.87	-3.31

Figure 5.8 shows a graphical representation of the bit allocation decisions through all ten generations for the naïve encoder. Figure 5.9 shows the same information for the aware encoder. Once again, notice how the aware encoder maintains more consistency through all ten generations of bit allocation decisions.

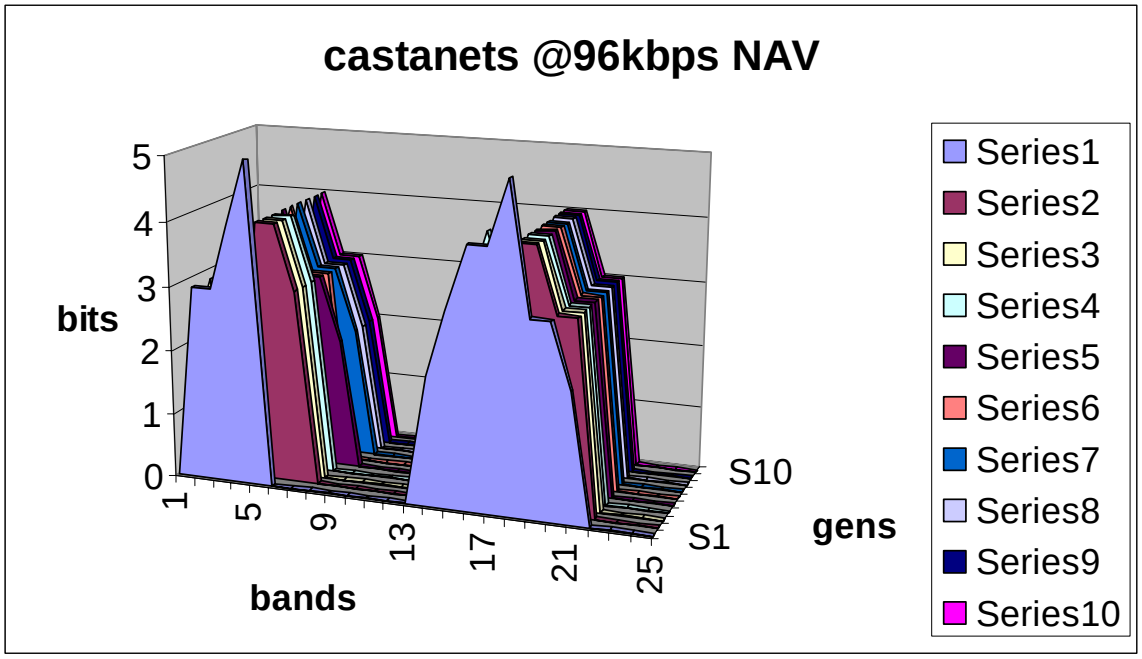

Figure 5.8 Bit allocations over all ten generations, naive castanets @96kbps

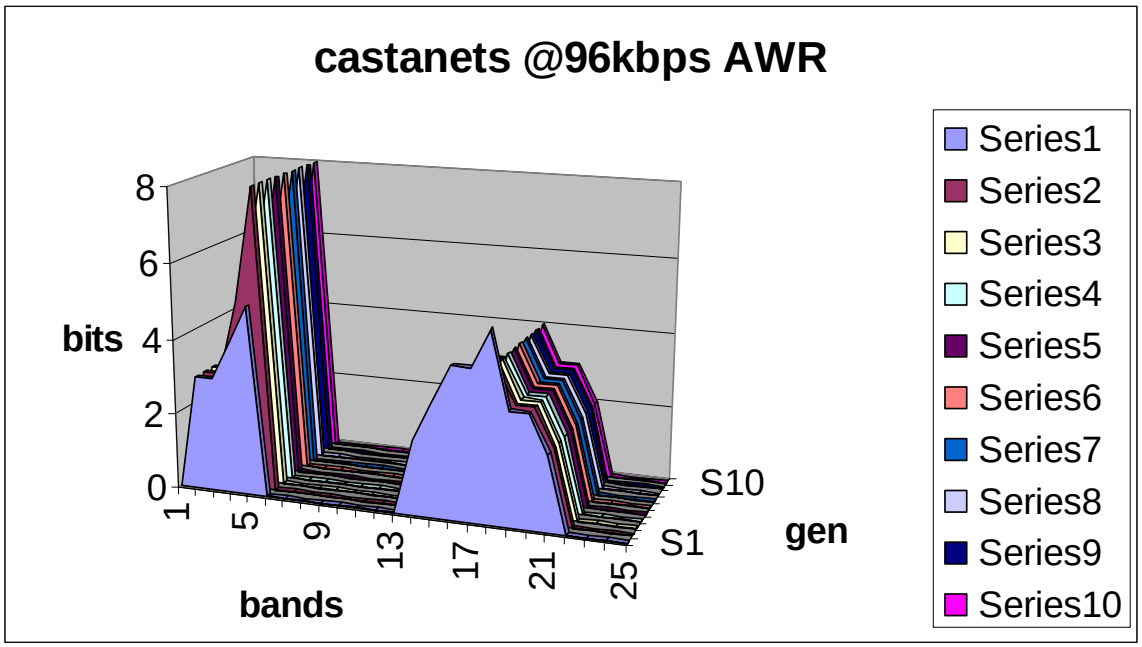

Figure 5.9 Bit allocations over all ten generations, aware castanets @96kbps

5.5 Analysis and Discussion – castanets @128kbps

The castanets file at 128kbps displayed some rather unusual trends. The top SDG scores were the sixth generation aware and third generation offset, followed closely by the third and tenth generation aware cases. The next five highest scores in order were the tenth generation naïve, sixth generation naïve, third generation naïve, and finally the sixth and tenth generation offset cases. These results are perhaps due to untrained listeners or the difficult program material of the castanets file. The naïve case scored progressively better through multiple generations, which is definitely not what was expected. Other than these anomalies, it appears that the other two 'trees' behaved as expected, showing a gradual degradation in the case of the offset tree and steady SDG measure in the case of the aware coder through ten generations.

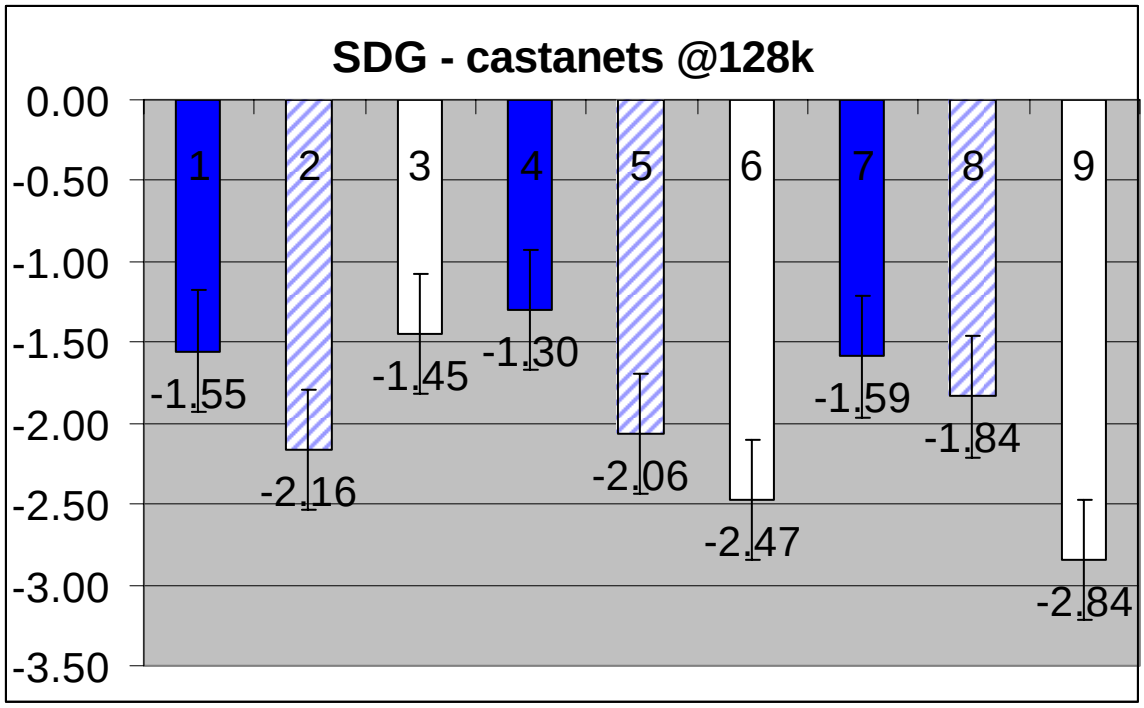

Figure 5.10 SDG Listening test chart for castanets @128kbps

Legend

1 – 3awr	2 – 3nav	3 – 3off
4 – 6awr	5 – 6nav	6 – 6off
7 – 10awr	8 – 10nav	9 – 10off

If third generations only are compared, the aware encoder outperformed the naïve encoder by an SDG score of 0.61.

If sixth generations only are compared, the aware encoder outperformed the naïve encoder by a margin of 0.76, with confidence intervals completely separated.

If only tenth generations are compared, a difference of 0.25 separates the aware from the naïve encoded generations. In this case, the aware encoder marginally outperformed the naïve encoder.

Once again, the aware encoded version of this file scored in the top three out of four SDG scores, making a fairly strong argument for the efficacy of the aware encoder. Although there are some strange trends in this data set, when viewed overall in light of the other data, it can still be said that the aware encoder outperforms the naïve encoder.

Table 5.5 shows the listening data for this case. The mean of the MOS score, the standard deviation (spread), 95% confidence intervals, 90% confidence intervals, and mean of the SDG are listed for each case.

Table 5.5 Listening Test Results - castanets @128 kbps

	3awr	3nav	3off	6awr	6nav	6off	10awr	10nav	10off
MOS score	3.00	2.39	3.11	3.25	2.49	2.08	2.96	2.72	1.71
Std dev	0.78	0.70	0.80	0.61	0.69	0.91	0.77	0.90	0.87
95% int	0.37	0.33	0.38	0.29	0.33	0.43	0.36	0.43	0.41
90% int	0.31	0.28	0.32	0.24	0.28	0.36	0.31	0.36	0.35
SDG	-1.55	-2.16	-1.45	-1.30	-2.06	-2.47	-1.59	-1.84	-2.84

Figure 5.11 shows a graphical representation of the bit allocation decisions through all ten generations for the naïve encoder. Figure 5.12 shows the same information for the aware encoder. Once again, notice how the aware encoder maintains more consistency through all ten generations of bit allocation decisions than the naïve encoder is able to achieve.

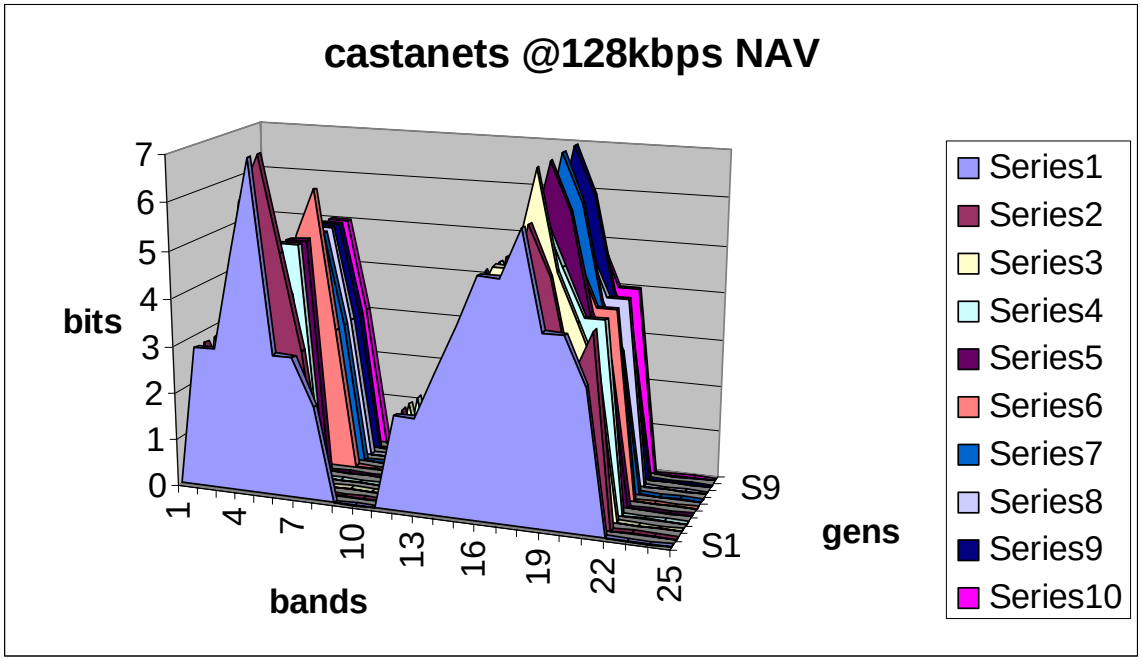

Figure 5.11 Bit allocations over all ten generations, naïve castanets @128kbps

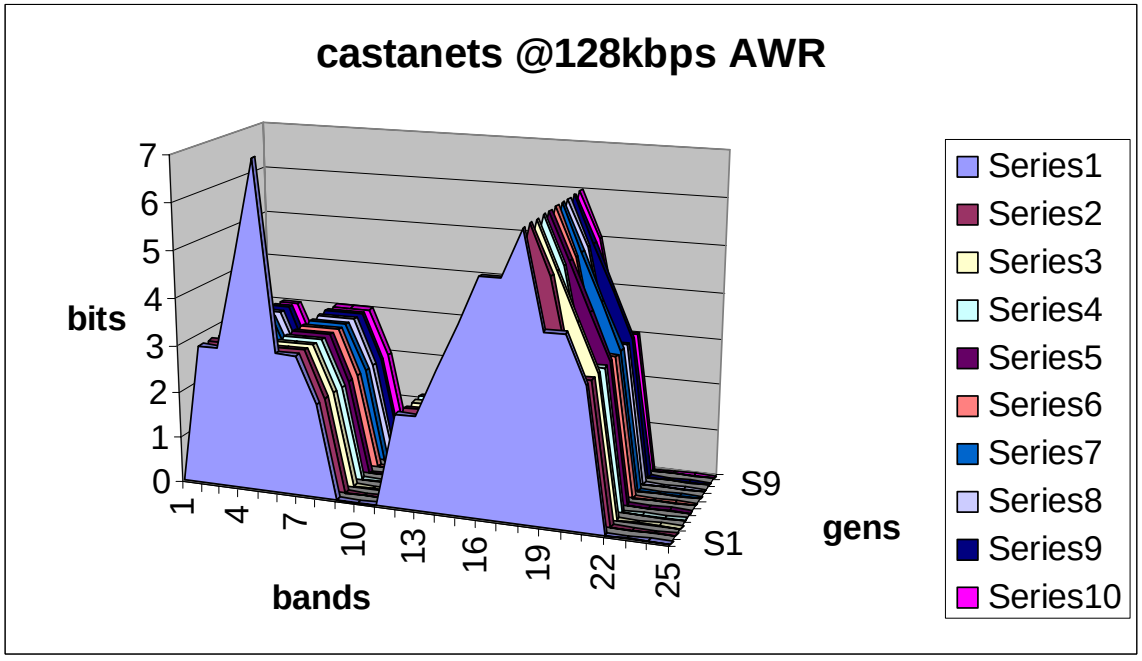

Figure 5.12 Bit allocations over all ten generations, aware castanets @128kbps

5.6 Further Discussion

Table 5.6 shows the number of critical bands that were fixed or free for each of the aware encoder cases. Only the bit allocations in free critical bands are allowed to take on a different value in subsequent codec generations. The more fixed critical bands, the better the aware encoder will perform; if errors can be limited to fewer critical bands, less distortion of the original signal occurs.

Table 5.6 Fixed and free critical bands for each of the four aware test scenarios

	# fixed critical bands (0,2, or 3 bits allocated)	**# free critical bands (4 or more bits allocated)**
french @ 96kbps	22	3
french @ 128kbps	19	6
castanets @ 96kbps	20	5
castanets @ 128kbps	17	8

Each bit allocation decision that was different from a previous stage introduces some perceptual error. These differences in quantization choices are called “quantization deltas” or “q deltas”. A larger number of “q deltas” indicates more quantization noise injected into subsequent generations. A minimum number of “q deltas” is the goal to reduce perceptual quality loss. However, it is also beneficial to reduce the number of critical bands where the “q deltas” occur. Fewer critical band locations where “q deltas” are occurring is an indication of better performance. A smaller range of frequencies for introduced error results and therefore, the error should be less noticeable. Table 5.7 summarizes the accumulated “q deltas” and the number of critical bands in which these “q deltas” occurred for each case.

Table 5.7 Accumulated q-delta errors, critical bands, and associated SDG scores

	Fr @ 96k			**Cs @ 96k**			**Fr @128k**			**Cs @ 128k**		
	Δq	#cb	SDG	Δq	#cb	SDG	Δq	#cb	SDG	Δq	#cb	SDG
3awr	2	2	-1.64	5	5	-1.54	5	5	-1.21	8	7	-1.55
3nav	8	7	-2.18	15	13	-2.18	13	10	-1.32	17	12	-2.16
6awr	6	2	-1.69	8	5	-2.06	5	5	-1.41	11	7	-1.30
6nav	13	7	-1.75	25	14	-2.43	23	12	-1.71	36	16	-2.06
10awr	10	2	-1.38	11	5	-2.16	5	5	-1.35	15	7	-1.59
10nav	13	7	-2.08	28	14	-2.87	41	12	-1.69	61	16	-1.84

The table shows the numerical measures compared to the perceptual error as rated by the listeners for each audio clip. It demonstrates how the aware encoder was able to minimize both the number of quantization errors and the number of critical bands in which they occurred, when compared to its naïve sibling.

Chapter 6: Conclusions and Further Research

The expected results were that the aware encoder would outperform the naïve encoder with frame synchronization. Both the naïve encoder and aware encoder with frame synchronization were expected to outperform the naïve encoder without any frame synchronization. The experimental results indicate that the aware encoder performs better in almost all cases over the naïve encoder, and at a very minimal cost in bit-rate.

The listening test data showed that the perceptual audio quality could be retained at levels very close to the original, even after as many as ten generations of cascaded audio quality, by restricting the bit allocation with a 50-bit word of side information. In every individual case (comparing generations of similar number) the aware encoder outperformed its naïve brethren. In most of the trials, this difference was most noticeable at the tenth generation, however, even after three generations there was a noticeable difference in SDG grades. The bit allocation constraint is general enough to be able to be applied to any individual or custom bit allocation technique.

Due to the small size of the side information sent, there are many options for transmitting this analysis word to subsequent generations. The most promising method appears to be using the method of implementing a perceptually transparent steganographic data embedding technique as described by Kurth [6]. Kurth's data embedding scheme, as applied to this project, would have the added benefit of a much more robust embedding environment since the embedding constraints are relaxed compared to the large amount of data necessary to embed vis-à-vis the MOLE signal. The decoder, upon re-creation of the PCM samples, would embed each of these 50 bits of information into the output file. An aware encoder would have a detector that would

extract this data from the input file and use the enhanced information to decrease audible artifacts.

The basic premise outlined in this research can be used by a codec that maintains framing structure across multiple generations. For codecs that have mismatches upon successive generations, a framing synchronization step would have to be implemented to achieve alignment for subsequent encoded generations.

One suggestion for further research is to study the behavior and use of this technique in codecs using options such as joint stereo coding, spectral band replication, prediction, temporal noise shaping, or any other of a number of tools available in encoders. The impact of using these coding tools was not researched in this project and may present different issues for the fixed bit allocations.

Another area that might require further characterization is to study the effects of multiple generations across codecs that use differing filter banks. This encoder used only the MDCT, which is a very common filterbank, but it is not the only spectral analysis technique in use. It is possible that the MDCT performs differently than a PQMF or another filterbank type in entirely different ways.

Furthermore, this research did not address the issue of reducing perceptual loss across different codec types. Because different codecs use any number of techniques to code audio, a scheme that is capable of using a universal MOLE signal would be valuable. This research only addressed perceptual quality loss when using identical codecs. Also, there is a limitation of this method when edits to the audio are necessary such as equalization, gain changes, or other operations that change the signal. In these

cases, the analysis word or MOLE signal would not be useful any longer since the sample values that the analysis word refers to would no longer be valid.

This research studied the efficacy of sending a very lightweight analysis word that fixes certain bit allocations and allows efficiency in subsequent encoding generations as opposed to a large data structure such as the MOLE signal. This has advantages. First, if small enough, it is possible to send this information via an embedded channel or perhaps even in ancillary data fields (this would vary from encoder to encoder). Secondly, it can be used with any almost bit allocation routine and hence easily adapted to any codec since it is performed after all other bit allocation functions. Finally, it is perhaps an elegant solution that can provide improvement at very little cost computationally.

References

[1] Boley, J. http://www.perceptualentropy.com/ (for basic MATLAB codec).

[2] Beaton, R. *Objective Perceptual Measurement of Audio Quality*. From Collected Papers on Digital Audio Bit-Rate Reduction. pp 126-152.

[3] Brandenburg, K. *Introduction to Perceptual Coding*. From Collected Papers on Digital Audio Bit-Rate Reduction, pp. 23-30. 1996, Audio Engineering Society.

[4] Brandenburg, K., and G. Stoll. *ISO-MPEG-1 Audio: A Generic Standard for Coding of High-Quality Digital Audio*. From Collected Papers on Digital Audio Bit-Rate Reduction, pp. 31-42.

[5] Cavagnolo, B. and J. Bier. *Introduction to Digital Audio Compression*. http://www.bdti.com/

[6] Dimkovic, I. *Improved ISO AAC Coder*.

[7] Fletcher, J., *ISO/MPEG Layer-2 – Optimum re-Encoding of Decoded Audio using a MOLE Signal*. 104th AES Convention, Amsterdam, May 16-19, 1998. Preprint 4706.

[8] Gilchrist, N., *ATLANTIC Audio: Preserving Technical Quality During Low Bit Rate Coding and Decoding*. 104th AES Convention, Amsterdam, May 16-19, 1998. Preprint 4694.

[9] Herre, J. and M. Schug. *Analysis of Decompressed Audio – The "Inverse Decoder"*. 109th AES Convention, Los Angeles, September 22-25, 2000. Preprint 5256.

[10] Herre, J., S. Moehrs, and R. Geiger. *Analysing decompressed audio with the "Inverse Decoder" – towards an operative algorithm*. 112th AES Convention, Munich, May 10-13, 2002. Preprint 5576.

[11] ISO 11172-3 Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 MBit/s. Oct 1992.

[12] Komly, A. Assessing the Performance of Low-Bit-Rate Audio Codecs: An Account of the Work of ITU-R Task Group 10/2. From Collected Papers on Digital Audio Bit-Rate Reduction, pp 105-114.

[13] Kurth, F. *An Audio Codec for Multiple Generations Compression Without Loss of Perceptual Quality*. AES 17th International Conference on High Quality Audio Coding.

[14] Lincoln, B. *An Experimental High Fidelity Perceptual Audio Coder.* Center for Computer Rsearch in Music and Acoustics (CCRMA). Dept. of Music, Stanford University. Mar 1998.

[15] Mason, A., A. McParland, and N. Yeadon. *The ATLANTIC Audio Demonstration Equipment*. AES 106th Convention, Munich, Germany, May 8-11, 1999. Preprint 4935.

[16] Noll, P. *MPEG Digital Audio Coding Standards*. CRC Press LLC. 1999.

[17] Painter, T., and A. Spanias. *Perceptual Coding of Digital Audio*. Proceedings of the IEEE, Vol. 88, No 4, April 2000.

[18] "Perceptual Audio Coders: What to Listen for." Audio Engineering Society CD, 2002.

[19] Pohlmann, K. Principles of Digital Audio, 4th ed. McGraw-Hill, New York, 2000.

[20] Ryden, T. *Using Listening Tests to Assess Audio Codecs*. From Collected Papers on Digital Audio Bit-Rate Reduction. pp. 115-125. Audio Engineering Society, 1996.

[21] Sayoud, K. Introduction to Data Compression. 2nd ed. Morgan Kaufmann, San Francisco, 2000.

[22] "Sound, synthesis and audio reproduction. APPENDIX: Audio coding/compression." http://www.helsinki.fi/~ssyreeni/dsound/dsound-a-03

[23] Stoll, G. *ISO-MPEG-2 Audio: A Generic Standard for the Coding of Two-Channel and Multichannel Sound*. From Collected Papers on Digital Audio Bit-Rate Reduction, pp. 43-53. Audio Engineering Society, 1996.

[24] "Time Offsets for Various Codecs". http://ff123.net/abchr/offsets.html

[25] Weishart, M., R. Gobel, and R. Herre. *Two-Pass Encoding Of Audio Material Using MP3 Compression*. 113th AES Convention, Los Angeles, October 5-8, 2002.

[26] Yang, D., A. Hongmei, C. Kyriakakis, and C. Kuo. *Design of Error-Resilient Layered Audio Codec*. 113th AES Convention, Los Angeles, October 5-8, 2002.

[27] Wang, Y. and M. Velermo. *Modified Discrete Cosine Transform – Its Implications for Audio Coding and Error Concealment*. AES Journal Volume 51, No. 1/2, pp. 52-61.

Appendix A – MOLE Signal Format [7]

```
                                                                    # of bits
mole_frame () {
        header()
        mole_data()
        error_check()
        ancillary_data()
}

header() {
        synchronization_word                                        16
        length                                                      11
        type                                                        5
        introduced_time_offset_flag                                 1
        reserved                                                    3
        if (introduced_time_offset_flag = = 1)
                introduced_time_offset                              16
}

mole_data {
        if (type = = MPEG Layer 2) {
                ID                                                  1
                bitrate_index                                       4
                sampling_frequency                                  2
                mode                                                2
                mode_extension                                      2
                copyright                                           1
                original/copy                                       1
                emphasis                                            1
                for (sb=0; sb<30; sb++)
                        allocation[sb]                              4
                for (sb=0; sb<30; sb++)
                        for (scfblock=0;scfblock<3;scfblock++)
                                scalefactor[sb][scfblock]           6
        }
}

error_check() {
        crc_check                                                   16
}

ancillary_data() {
        for (b=0;b<no_of_ancillary_bits; b++)
                ancillary_bit                                       1
}
```

Appendix B – Inverse Decoder Results

Test item	Description	Worst NMR (mono) [dB]	ODG (mono)	Worst NMR (stereo) left [dB]	Worst NMR (stereo) right [dB]	ODG (stereo)
es01	vocal (Susan Vega)	-18.39	-0.05	-5.48	-8.14	-0.07
es02	German Speech	-9.92	-0.01	-7.48	-5.62	-0.17
es03	English Speech	-8.83	-0.12	-5.70	-9.00	-0.22
si01	harpsichord	-5.96	-0.45	-5.56	-4.65	-0.26
si02	castanets	-6.94	-0.11	-8.44	-7.05	-0.08
si03	pitch pipe	-7.03	-0.33	-4.00	-5.86	-0.49
sm01	bagpipes	-9.07	-0.01	-5.43	-4.96	-0.08
sm02	glockenspiel	-5.39	-0.36	-5.86	-2.99	-0.41
sm03	plucked strings	-7.70	-0.10	-7.56	-6.05	-0.06
sc01	trumpet solo and orchestra	-14.10	-0.05	-12.22	-10.36	-0.02
sc02	orchestral piece	-9.57	-0.03	-7.31	-6.82	-0.09
sc03	contemporary pop music	-10.74	-0.04	-8.89	-6.02	-0.02
average		-9.47	-0.14	-6.99	-6.46	-0.16

Table B-1. Test excerpts, achieved NMR and ODG for the inverse decoder [10]

Reconstruction Cycle	1	2	3	1	2	3
Test item	Worst NMR [dB]	Worst NMR [dB]	Worst NMR [dB]	ODG	ODG	ODG
es01	-18.39	-18.38	-17.79	-0.05	-0.05	-0.06
es02	-9.92	-8.83	-8.12	-0.01	-0.01	-0.02
es03	-8.83	-8.32	-7.87	-0.12	-0.18	-0.20
si01	-5.96	-5.55	-5.44	-0.45	-0.51	-0.52
si02	-6.94	-6.79	-6.70	-0.11	-0.14	-0.16
si03	-7.03	-5.99	-5.81	-0.33	-0.48	-0.52
sm01	-9.07	-8.94	-8.66	-0.01	-0.01	-0.01
sm02	-5.39	-5.15	-5.14	-0.36	-0.41	-0.45
sm03	-7.70	-7.40	-7.39	-0.10	-0.10	-0.17
sc01	-14.10	-11.16	-11.15	-0.05	-0.01	-0.01
sc02	-9.57	-9.55	-9.54	-0.03	-0.03	-0.04
sc03	-10.74	-9.82	-9.51	-0.04	-0.16	-0.17
average	-9.47	-8.82	-8.59	-0.14	-0.17	-0.19

Table B-2. Achieved NMR for multiple generations compressed/decompressed audio signals [10]

NMR values below 0 dB indicate headroom below the threshold of audibility whereas values larger than 0 dB indicate audible distortions. ODG is a computed value done by the PEAQ algorithm which attempts to mimic the typical objective listening test evaluations ranging from -4 (very annoying) to 0 (imperceptible difference) which is ideal for listening test but often cumbersome, difficult, and costly to obtain using real listeners.

Appendix C – MATLAB code

```
% code_stereo()
% Rob Burke 2005.
% adapted baseline MATLAB codec by Jon Boley (jdb@jboley.com)
%
% original_filename is the input file
% coded_filename is the encoded file
% decoded_filename is the decoded output file
% bitrate is the cbr for each mono channel; stereo bitrate = bitrate * 2
% N is length of input audio block and MDCT, and FFTs
% generation, aware, inputs by user

clear all;

scalebits = 4;
bitrate = 64000;  % bitrate per channel; stereo bitrate is twice this.
stBitK = bitrate/500; %stereobitrate for filenaming
N = 2048; % framelength

generation = input('Which generation of encodes is this run?');

disp('okay');
aware = input('Run in "aware" mode? 0=no 1=yes');

if (aware)
   disp('encoder running in aware mode');
else
   disp('encoder running in naive mode');
end

readfile = sprintf('analysis%i.rob',generation);
ritefile = sprintf('analysis%i.rob',generation+1);

if (generation > 1)
   fid2 = fopen(readfile,'r'); % use for reading
end

fid3 = fopen(ritefile,'w'); % use for writing

original_filename = sprintf('french-%i-%i-awr.wav', stBitK,generation - 1);
decoded_filename  = sprintf('french-%i-%i-awr.wav', stBitK,generation);
coded_filename = sprintf('french.rob', generation);

% read in the source file
[tone,Fs,NBITS] = wavread(original_filename);

% Input and debug section
% use only one of the following two options
% use offset = 0 if running in ‘naïve’ or ‘aware’ mode
% use random offset only when re-creating ‘offset’ case
offset = 0;
%offset = ceil(200*rand(1)); % offset random amt from 1-200 samps
outstring = sprintf('offsetting by %i samples this encode cycle', offset);
disp(outstring);

% offset
if offset ~= 0
   tone = tone(offset+1:end,:);
   tone(end+offset,2) = 0;   % add zeros at end to fill up array
end


%  Calculate the number of subbands
num_subbands = floor(fftbark(N/2,N/2,Fs))+1;
%  Figure out total bits per frame
%  raw bits - scalefactor gain - bit allocation
bits_per_frame = floor(((bitrate/Fs)*(N/2)) - (scalebits*num_subbands) - 100);
```

```
%  Enframe Audio (stereo)
FRAMES(:,:,1) = enframe(tone(:,1),N,N/2); % L channel
FRAMES(:,:,2) = enframe(tone(:,2),N,N/2); % R channel

% pre-computed contants/variables (to improve performance)
analysisWord = zeros(length(FRAMES(:,1)));
totalFrames = length(FRAMES(:,1));
floorFftBark = floor(fftbark(N/2,N/2,Fs))+1;
fftmax = 586.7143;  % max(abs(fft(1kHz tone)))... defined as 96dB
debugOn = 0;

% NEED TO CALCULATE SAMPLES PER SUBBAND
n = 0:N/2-1;
f_Hz = n*Fs/N;
f_kHz = f_Hz / 1000; % f in kHz
%  threshold in quiet calc (A)
A = 3.64*(f_kHz).^(-0.8) - 6.5*exp(-0.6*(f_kHz - 3.3).^2) + (10^(-3))*(f_kHz).^4;
%  bark frequency scale
z = 13*atan(0.76*f_kHz) + 3.5*atan((f_kHz/7.5).^2);
crit_band = floor(z)+1;
num_crit_bands = max(crit_band);
num_crit_band_samples = zeros(num_crit_bands,1);
for i=1:N/2  % used in bit allocation routine.  computed once out here
    num_crit_band_samples(crit_band(i)) = num_crit_band_samples(crit_band(i)) + 1;
end

%  Write File Header for encoded sound
fid = fopen(coded_filename,'w'); % Get File ID to coded filename
fwrite(fid, Fs, 'ubit16'); % Sampling Frequency
fwrite(fid, N, 'ubit12'); % Frame Length
fwrite(fid, bitrate, 'ubit18'); % Bit Rate
fwrite(fid, scalebits, 'ubit4'); % Number of Scale Bits per Sub-Band
fwrite(fid, totalFrames, 'ubit26'); % Number of total frames

outstring = sprintf('Encoding file %s, @%i kbps, total frames = %i', original_filename,
bitrate*2/1000, totalFrames);
disp(outstring);

%   Encoding Begins
for j=1:2 % for stereo loop
   % message update
   if j==1
      disp('L channel encode');
   else
      disp ('R channel encode');
   end  % end message update

   % frame loop - do once for each enframed block of input data
   for frame_count=1:totalFrames
      % display message every 10 frames
      if mod(frame_count,10) == 0
         outstring = sprintf('encoding frame %i', frame_count);
         disp(outstring);
      end
      fft_frame = fft(FRAMES(frame_count,:,j));

      % can zero out scalefactors (Gain) and bit_alloc if no energy in current frame
      if fft_frame == zeros(1,N)
         Gain = zeros(1,floorFftBark);
         bit_alloc = zeros(1,floorFftBark);
      % otherwise, compute everything
      else
         len = length(fft_frame);
         peak_width = zeros(1,len);
         peak_points = cell(len,len);
         peak_min_value = zeros(1,len);

         % Find Peaks in FFT
         % centers - array of peaks in audio signal
         centers = find(diff(sign(diff( abs(fft_frame).^2) )) == -2) + 1;
         spectral_density = zeros(1,length(centers));
```

```
         for k=1:length(centers)  % for each peak (k is peak index into FFT)           %
            peak_max(k) = centers(k) + 2; % highest index bin of kth peak
            peak_min(k) = centers(k) - 2; % lowest index bin of kth peak
            peak_width(k) = peak_max(k) - peak_min(k);

            for p=peak_min(k):peak_max(k) % p takes on each freq bin
               if (p > 0) & (p < N) % make sure it doesn't go out of bounds
                  % add the magnitude-squared of the FFT at that bin to overall sd
                  spectral_density(k) = spectral_density(k) + abs(fft_frame(p))^2;
               end
            end
         end

         % This gives the amplitude squared of the original signal
         modified_SD = spectral_density / ((N^2)/8);
         SPL = 96 + 10*log10(modified_SD);

         % TRANSFORM FFT'S TO SPL VALUES
         fft_spl = 96 + 20*log10(abs(fft_frame)/fftmax);

         % Masking Spectrum
         % find max of TiQ, Schroeder
         big_mask = max(A,Schroeder2(centers(1)*(Fs/2)/N,fft_spl(centers(1)),...
            14.5+bark(centers(1)*(Fs/2)/N),f_kHz,A));
         for peak_count=2:length(centers)
            big_mask =
max(big_mask,Schroeder2(centers(peak_count)*(Fs/2)/N,fft_spl(centers(peak_count)),...
               14.5+bark(centers(peak_count)*(Fs/2)/N),f_kHz,A));
         end

         %if (frame_count == 5) & (j==2)
         %   figure;semilogx([0:(Fs/2)/(N/2):Fs/2-1],big_mask,'m');
         %end

         % Signal Spectrum - Masking Spectrum (with min of 0dB)
         New_FFT = fft_spl(1:N/2)-big_mask;
         New_FFT_indices = find(New_FFT > 0);
         New_FFT2 = zeros(1,N/2);
         for i=1:length(New_FFT_indices)
            New_FFT2(New_FFT_indices(i)) = New_FFT(New_FFT_indices(i));
         end


         if (frame_count == 5) & (j==2)
            %semilogx([0:(Fs/2)/(N/2):Fs/2-1],fft_spl(1:N/2),'b');
            %hold on;
            %semilogx([0:(Fs/2)/(N/2):Fs/2-1],big_mask,'m');
            %hold off;
            %title(outstr);
            %figure;
            %semilogx([0:(Fs/2)/(N/2):Fs/2-1],New_FFT2);
            %title('SMR');
            debugOn = 1;
            %figure;
            %stem(allocate(New_FFT2,bits_per_frame,N,Fs));
            %title('Bits perceptually allocated');
         end


         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         %     READ ANALYSIS WORD   %
         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         if (aware) & (generation > 1)
            %disp('read analysis word');
            % read z_band info
            [z_word, word_count] = fread(fid2, 1, 'ubit25');


            % get the subbands w/ zeros
            z_bands = zeros(1,num_subbands); % create empty z_bands array
```

```
      for t = 1:num_subbands
         z_bands(t) = bitget(z_word,t);
      end

      % read two_band info
      [t_word, word_count] = fread(fid2, 1, 'ubit25');
      two_bands = zeros(1,num_subbands); % create empty two_bands array
      % get the subbands w/ 2's
      for t = 1:num_subbands
         two_bands(t) = bitget(t_word, t);
      end
      if debugOn
         disp('read the following word1');
         z_bands
         disp('read the following word2');
         two_bands
      end

   else % do for 1st generation
      z_bands = zeros(1,num_subbands);
      two_bands = zeros(1,num_subbands);
   end


   bit_alloc = allocate(New_FFT2,bits_per_frame,N,Fs,...
      num_crit_band_samples,debugOn,aware,z_bands,two_bands,generation);
   if (frame_count == 5) & (j==2)
      stem(bit_alloc,'mx');hold off;
      outstr = sprintf('bit allocation for %s frame %i generation %i', ...
      decoded_filename, frame_count, generation);
      title (outstr);
      debugOn = 0; % reset debug flag
      %outstr = sprintf('channel %i', j);
      %disp(outstr);
      % the folowing for showing avg energy/sb
      %frame_spl = zeros(1,25);
      %for sb = 1:length(frame_spl)
      %   indices = find((floor(fftbark([1:N/2],N/2,Fs))+1)==sb);
      %end
      %frame_spl(sb) = sum(fft_spl(indices(1:end))) / length(indices);
   end % end of debug if section for plotting, etc.

   [Gain,Data] = p_encode(mdct(FRAMES(frame_count,:,j)),Fs,N,bit_alloc);
end % end if-else statement


% Write Audio Data to File
qbits = sprintf('ubit%i', scalebits);  % read scalebits for scalefactors into qbits
count_gain = fwrite(fid, Gain, qbits); % scalefactors controlled by scalebits
count_allo = fwrite(fid, bit_alloc, 'ubit4');  % bit allocation always 4-bit max
count_ovr = 0; % initialize a counter to count bits used to write data
for i=1:num_subbands  % loop through subbands
   % find the current subbands to write with the current bit alloc
   indices = find((floor(fftbark([1:N/2],N/2,Fs))+1)==i);

   qbits = sprintf('ubit%i', bit_alloc(i));
   if (bit_alloc(i) > 16)
      qbits = sprintf('ubit16');
      bit_alloc(i) = 16;
   end
   if ((bit_alloc(i) ~= 0) & (bit_alloc(i) ~= 1))
      count_data = fwrite(fid, double(Data(indices(1):indices(end))), qbits);
      count_ovr = count_ovr + count_data*bit_alloc(i);
   end
end

% WRITE ANALYSIS WORD
% create the encoder analysis word here only if generation 1
% step 1. find the subbands having highest bit allocations (~11 out of the 25).
% step 2. set a bit in the analysis word for each subband having emphasis
if ((generation == 1) & aware) % do the first time through
```

```
         disp('writing analysis codeword');
         emphasis_count = 0;
         z_word = 0;
         t_word = 0;
         subbit_alloc = bit_alloc;

         % find and write all zero bit allocs to the analysis words
         % z_word            t_word               value to send
         %   0               0                    allow free bit alloc
         %   0               1                    2
         %   1               0                    0
         %   1               1                    3
         zeroed = find(subbit_alloc==0);
         for u = zeroed(1:end) % set a bit in analysis_word to signify a 0
            z_word = bitset(z_word, u);
            emphasis_count = emphasis_count + 1;
            %subbit_alloc(i) = 0; % remove possibility of returning this index
         end
         z_word = z_word(end);

         twoed = find(subbit_alloc==2);
         for u = twoed(1:end) % set a bit in analysis word to signify a 2
            t_word = bitset(t_word, u);
            emphasis_count = emphasis_count + 1;
            %subbit_alloc(i) = 0; % remove possibility of returning this index
         end
         t_word = t_word(end);

         threed = find(subbit_alloc==3);
         for u = threed(1:end) % set both bits in the analysis words to signify a 3
            t_word = bitset(t_word, u);
            z_word = bitset(z_word, u);
         end
         t_word = t_word(end);
         z_word = z_word(end);

         % get the subbands w/ emphasis
         z_bands = zeros(1,num_subbands);
         for t = 1:num_subbands
            z_bands(t) = bitget(z_word,t);
         end

         two_bands = zeros(1,num_subbands);
         for t = 1:num_subbands
            two_bands(t) = bitget(t_word,t);
         end

         if (frame_count==5) & (j==2)
            disp(outstr);
            bands = find(z_bands)
            bands2 = find(two_bands)
         end
      end

      if (aware)
         count_analysis0 = fwrite(fid3, z_word, 'ubit25');
         count_analysis2 = fwrite(fid3, t_word, 'ubit25');
      end

      % end write analysis word portion
   end % end of frame loop
   end % end of stereo loop

fclose(fid);

if (generation > 1)
   fclose(fid2);
end
fclose(fid3);
```

```
% decode_stereo()
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      READ FILE HEADER     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fid = fopen(coded_filename,'r');
Fs          = fread(fid,1,'ubit16'); % Sampling Frequency
framelength = fread(fid,1,'ubit12'); % Frame Length
bitrate     = fread(fid,1,'ubit18'); % Bit Rate
scalebits   = fread(fid,1,'ubit4' ); % Number of Scale Bits per Sub-Band
num_frames  = fread(fid,1,'ubit26'); % Number of frames

%counters
imdct_count = [0 0]; % L and R channel
%xtest = zeros(num_frames, framelength);
%xhold = zeros(1,framelength);

% pre-computed or static variables
floorFftBark = floor(fftbark(framelength/2,framelength/2,Fs))+1;

for j=1:2 % stereo loop
   outstring = sprintf('Decoding %s channel %i', decoded_filename, j);
   disp(outstring);
   for frame_count=1:num_frames % frame loop
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %     READ FILE CONTENTS    %
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      qbits = sprintf('ubit%i', scalebits);
           [gain, count_gain] = fread(fid,num_subbands,qbits);
      [bit_alloc, count_allo] = fread(fid,num_subbands,'ubit4');
      count_data = 0;

      for i=1:floorFftBark
         indices = find((floor(fftbark([1:framelength/2],framelength/2,Fs))+1)==i);
         if  (bit_alloc(i) ~= 0) & (bit_alloc(i) ~= 1)
            qbits = sprintf('ubit%i', bit_alloc(i));
            [InputValues(indices(1):indices(end)), count] = fread(fid, length(indices)
,qbits);
            count_data = count_data + count*bit_alloc(i);
         else
            InputValues(indices(1):indices(end)) = 0;
         end
      end




      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %     DEQUANTIZE VALUES     %
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      for i=1:length(InputValues)
         if InputValues(i) ~= 0
            if max(bit_alloc(floor(fftbark(i,framelength/2,Fs))+1),0) ~= 0
               InputValues(i) = midtread_dequantizer(InputValues(i),...
               max(bit_alloc(floor(fftbark(i,framelength/2,Fs))+1),0));
            end
         end
      end


      for i=1:25
         gain2(i) = 2^gain(i); % were written as log2: undo
      end



      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %         APPLY GAIN        %
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
      for i=1:floorFftBark
         indices = find((floor(fftbark([1:framelength/2],framelength/2,Fs))+1)==i);
         InputValues(indices(1):indices(end)) = ...
         InputValues(indices(1):indices(end)) * gain2(i);
      end
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %       INVERSE MDCT         %
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      x2((frame_count-1)*framelength+1:frame_count*framelength) = ...
        imdct(InputValues(1:framelength/2));
      imdct_count(j) = imdct_count(j) + 1;
   end  % END FRAME LOOP


   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %     RECOMBINE FRAMES       %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % for odd  # of frames - use ((frames+1)/2)*samples/frame = samples
   % for even # of frames - use (frames/2 + 1/2 frame)*samples/frame = samples
   if rem(num_frames,2) % odd # of frames
      disp('odd # of frames');
      outputBuffLen = ((num_frames+1)/2)*framelength;
      outstr = sprintf('total buffer length = %i samples', outputBuffLen);
      disp(outstr);
   else % even # of frames
      disp('even # of frames');
      outputBuffLen = ((num_frames/2)+0.5)*framelength;
      outstr = sprintf('total buffer length = %i samples', outputBuffLen);
      disp(outstr);
   end

   chanR = zeros(1,outputBuffLen);  %outputBuffLen determined above
   for i=0:0.5:floor(length(x2)/(2*framelength))-0.5
      start = i*framelength+1;
      next  = (i+1)*framelength;
      chanR(start:next) = chanR(start : next) + ...
         x2((2*i)*framelength+1:(2*i+1)*framelength);
   end

   if (j==1)
      chanL = chanR;  % just write all the reconstructed data to the left channel
      disp('done with L channel data!');
   end



end   % end stereo loop

status = fclose(fid);

% build stereo data
stereo_data = zeros(length(chanL),2); % create matrix
stereo_data(:,1) = reshape(chanL,length(chanL),1); % populate
stereo_data(:,2) = reshape(chanR,length(chanR),1); % populate


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        WRITE FILE          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
wavwrite(stereo_data,Fs,decoded_filename);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        ALLOCATE            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function
x=allocate(y,b,N,Fs,num_crit_band_samples,debugOn,aware,z_bands,two_bands,generation)
% x=allocate(y,b,N)
% Allocates b bits to the 25 subbands
% of y (a length N/2 SMR, in dB SPL)
% debugOn - just a flag to create figure or not
% aware 1-aware 0-naive
% z_bands contains the subband flags for 0 quantization

if (generation==1) | (aware==0)   % generation 1 - ignore z_bands,two_bands
   %disp('regular bit allocation');
   bits(floor(bark( (Fs/2)*[1:N/2]/(N/2) )) +1) = 0;
   for i=1:N/2
      bits(floor(bark( (Fs/2)*i/(N/2) )) +1) = ...
      max(bits(floor(bark( (Fs/2)*i/(N/2) )) +1) , ceil( y(i)/6 ));
   end
   indices = find(bits(1:end) < 2);
   bits(indices(1:end)) = 0;
   if debugOn
      figure;stem(bits);title('Bit Allocation Targets');hold on;
   end
   x=zeros(1,25);
   bitsleft=b;
   [blah,i]=max(bits);
   while bitsleft > num_crit_band_samples(i)
      [blah,i]=max(bits);
      x(i) = x(i) + 1;
      bits(i) = bits(i) - 1;
      bitsleft=bitsleft-num_crit_band_samples(i);
   end

   %  the main allocation is done.  fix any with 1 bit allocations.
   %  find any subbands with 1 bit coded (recover wasted bits)
   %if debugOn
   %disp('the following in the bit allocation were given 1 bit.  DOH!');
   indices = find(x(1:end) == 1);
   bitsfreed = 0;
   for k=indices(1:end)
      bitsfreed = bitsfreed + num_crit_band_samples(k);
      x(k)=0;
   end

   bitsleft = bitsleft + bitsfreed;

   indices = find(bits); % contains the indices of all the subbands not coded (lowest)
   q = min(indices);

   while bitsleft > num_crit_band_samples(q)
          q=min(indices);
      x(q) = x(q) + 1;
      bits(q) = bits(q) - 1;
      bitsleft = bitsleft - num_crit_band_samples(q);
      indices = find(bits);
   end

   %disp('------------------end bit alloc this frame-------------------');
else % use the subband flags (generation 2 or higher)
   %%%%%% ENHANCED BIT ALLOCATION use z_bands, two_bands

   if debugOn
      outstr = sprintf('generation %i using ENHANCED bit allocation.', generation);
      disp(outstr);
   end

   % create bits array which will contain the ideal masking of quantization noise
   bits(floor(bark( (Fs/2)*[1:N/2]/(N/2) )) +1) = 0;
```

```
% for each freq bin, find maximum dB of SMR
for i=1:N/2 % i is each freq bin, the value in bits, will set max in each bark band
   bits(floor(bark( (Fs/2)*i/(N/2) )) +1) = ...
   max(bits(floor(bark( (Fs/2)*i/(N/2) )) +1) , ceil( y(i)/6 )); %scale by 6 (6dB/bit)
end

indices = find(bits(1:end) < 2);
bits(indices(1:end)) = 0;  % sets the zero-bit allocation for any 'free' bark bands
if debugOn
   figure;stem(bits);title('Bit Allocation Targets');hold on;
end


% get the 3_values first; any sb appearing in both z_bands and two_bands is actually 3
thr_bands = zeros(1,length(z_bands));
fre_bands = zeros(1,length(z_bands));
for i=1:length(z_bands)
   if (z_bands(i) & two_bands(i)) % this band is a 3_band
      thr_bands(i) = 1;  % set a 3
      z_bands(i) = 0;      % clear the 0
      two_bands(i) = 0;    % clear the 2
   end
   if (~z_bands(i) & ~two_bands(i) & ~thr_bands(i))
      fre_bands(i) = 1; % set as a free band
   end
end

% now each array will have explicit on/off info for all set subbands
if debugOn
   disp('zeroed bands passed in');
   z_bands
   disp('twoed bands passed in');
   two_bands
   disp('threed bands passed in');
   thr_bands
   disp('free bands passed in');
   fre_bands
end

x=zeros(1,25);

bitsleft=b;


% pre-calculate bitsleft; there are a fixed amount already spoken for
% just do for 2 and 3; 0 is ignored!
for i=1:length(z_bands)
   if (two_bands(i))  % if its a 2
      bitsleft = bitsleft - (2*num_crit_band_samples(i));
      bits(i) = bits(i) - 2;
   end
   if (thr_bands(i))
      bitsleft = bitsleft - (3*num_crit_band_samples(i));
      bits(i) = bits(i) - 3;
   end
end


[blah,i]=max(bits);
while bitsleft > num_crit_band_samples(i)
   [blah,i]=max(bits);
   bits(i) = bits(i) - 1;
   x(i) = x(i) + 1;
   bitsleft=bitsleft-num_crit_band_samples(i);
end


%   the main allocation is done.  fix any with 1 bit allocations.
%   find all subbands with 1 bit coded
indices = find(x(1:end) == 1);
```

```
   bitsfreed = 0;
   for k=indices(1:end)
      bitsfreed = bitsfreed + num_crit_band_samples(k);
      x(k)=0;
   end

   bitsleft = bitsleft + bitsfreed;

   indices = find(bits); % contains the indices of all the subbands not coded (lowest)
   q = min(indices);

   while bitsleft > num_crit_band_samples(q)
      q=min(indices);
      x(q) = x(q) + 1;
      bits(q) = bits(q) - 1;
      bitsleft = bitsleft - num_crit_band_samples(q);
      indices = find(bits);
   end

   %% set all specified subbands (just overwrite - allocation above has compensated)
   for i = 1:length(z_bands)
      if z_bands(i) % zero band specified
         x(i) = 0;  % set bitalloc to 0
      end
      if two_bands(i)  % 2 band specified
         x(i) = 2;            % set bitalloc to 2
      end
      if thr_bands(i)  % 3 band specified
         x(i) = 3;            % set bitalloc to 3
      end
   end



   %outstring = sprintf('%i bits left', bitsleft);
   %disp(outstring);
   %disp('------------------end bit alloc this frame-------------------');
end % end if-else statement
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           BARK              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function b=bark(f)
% b=bark(f)
% Converts frequency to bark scale
% Frequency should be specified in Hertz

b = 13*atan(0.76*f/1000) + 3.5*atan((f/7500).^2);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          FFTBARK            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function b=fftbark(bin,N,Fs)
% b=fftbark(bin,N,Fs)
% Converts fft bin number to bark scale
% N is the fft length
% Fs is the sampling frequency
f = bin*(Fs*0.5)/N; % frequency representation of bin# passed in
b = 13*atan(0.00076*f) + 3.5*atan((f/7500).^2); % corresponding bark value
return;


function f=fftbinToFreq(bin,N,Fs);
%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   FFT BIN TO FREQ     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% bin - bin number of FFT of length N
% N - number of points in FFT
% Fs - sampling frequency of signal

f = [0:Fs/N*.5:Fs/2];
f = f(bin);
f


function y = quant(x,q)
%QUANT Discretize values as multiples of a quantity.
%
%  Syntax
%
%    quant(x,q)
%
%  Description
%
%    QUANT(X,Q) takes these inputs,
%      X - Matrix, vector or scalar.
%      Q - Minimum value.
%    and returns values in X rounded to nearest multiple of Q
%
%  Examples
%
%    x = [1.333 4.756 -3.897];
%    y = quant(x,0.1)

% Mark Beale, 12-15-93
% Copyright 1992-2002 The MathWorks, Inc.
% $Revision: 1.10 $  $Date: 2002/04/14 21:33:02 $

y = round(x/q)*q;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          ENFRAME             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function f=enframe(x,win,inc)
%ENFRAME split signal up into (overlapping) frames: one per row. F=(X,WIN,INC)
%
%       F = ENFRAME(X,LEN) splits the vector X up into

%       frames. Each frame is of length LEN and occupies
%       one row of the output matrix. The last few frames of X
%       will be ignored if its length is not divisible by LEN.
%       It is an error if X is shorter than LEN.
%
%       F = ENFRAME(X,LEN,INC) has frames beginning at increments of INC
%       The centre of frame I is X((I-1)*INC+(LEN+1)/2) for I=1,2,...
%       The number of frames is fix((length(X)-LEN+INC)/INC)
%
%       F = ENFRAME(X,WINDOW) or ENFRAME(X,WINDOW,INC) multiplies
%       each frame by WINDOW(:)

%       Copyright (C) Mike Brookes 1997
%
%      Last modified Tue May 12 13:42:01 1998
%
%   VOICEBOX home page: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   This program is free software; you can redistribute it and/or modify
%   it under the terms of the GNU General Public License as published by
%   the Free Software Foundation; either version 2 of the License, or
%   (at your option) any later version.
%
%   This program is distributed in the hope that it will be useful,
%   but WITHOUT ANY WARRANTY; without even the implied warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%   GNU General Public License for more details.
%
%   You can obtain a copy of the GNU General Public License fr
%   ftp://prep.ai.mit.edu/pub/gnu/COPYING-2.0 or by writing to
%   Free Software Foundation, Inc.,675 Mass Ave, Cambridge, MA 02139, USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nx=length(x);
nwin=length(win);
if (nwin == 1)
   len = win;
else
   len = nwin;
end
if (nargin < 3)
   inc = len;
end
nf = fix((nx-len+inc)/inc);
f=zeros(nf,len);
indf= inc*(0:(nf-1)).';
inds = (1:len);
f(:) = x(indf(:,ones(1,len))+inds(ones(nf,1),:));
if (nwin > 1)
    w = win(:)';
    f = f .* w(ones(nf,1),:);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           IMDCT              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y = imdct(X)

X=X(:);
N = 2*length(X);
ws = sin(([0:N-1]'+0.5)/N*pi);
n0 = (N/2+1)/2;
Y = zeros(N,1);

Y(1:N/2) = X;
Y(N/2+1:N) = -1*flipud(X);
Y = Y .* exp(j*2*pi*[0:N-1]'*n0/N);
y = ifft(Y);
y = 2*ws .* real(y .* exp(j*2*pi*([0:N-1]'+n0)/2/N));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            MDCT              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y = mdct(x)

x=x(:);
N=length(x);
n0 = (N/2+1)/2;
wa = sin(([0:N-1]'+0.5)/N*pi);
y = zeros(N/2,1);

x = x .* exp(-j*2*pi*[0:N-1]'/2/N) .* wa;

X = fft(x);

y = real(X(1:N/2) .* exp(-j*2*pi*n0*([0:N/2-1]'+0.5)/N));
y=y(:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   MIDTREAD_DEQUANTIZER    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ret_value] = midtread_dequantizer(x,R)

sign = (2 * (x < 2^(R-1))) - 1;
Q = 2 / (2^R - 1);

x_uint = uint32(x);
x = bitset(x_uint,R,0);
x = double(x);

ret_value = sign * Q .* x;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    MIDTREAD_QUANTIZER     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ret_value] = midtread_quantizer(x,R)

Q = 2 / (2^R - 1);
q = quant(x,Q);
s = q<0;
ret_value = uint16(abs(q)./Q + s*2^(R-1));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         P_ENCODE             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Quantized_Gain,quantized_words]=p_encode(x2,Fs,framelength,bit_alloc,scalebits)

for i=1:floor(fftbark(framelength/2,framelength/2,Fs))+1
    indices = find((floor(fftbark([1:framelength/2],framelength/2,Fs))+1)==i);
    Gain(i) = 2^(ceil(log2((max(abs(x2(indices(1):indices(end))+1e-10))))));
    if Gain(i) < 1
        Gain(i) = 1;
    end
    x2(indices(1):indices(end)) = x2(indices(1):indices(end)) / (Gain(i)+1e-10);
    Quantized_Gain(i) = log2(Gain(i));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           SCHROEDER          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function m=Schroeder2(freq,spl,downshift,f_kHz,A)
% Calculate the Schroeder masking spectrum for a given frequency and SPL

N = 2048;
%f_kHz = [1:48000/N:48000/2];
%f_kHz = f_kHz/1000;
%A = 3.64*(f_kHz).^(-0.8) - 6.5*exp(-0.6*(f_kHz - 3.3).^2) + (10^(-3))*(f_kHz).^4;
f_Hz = f_kHz*1000;

% Schroeder Spreading Function
dz = bark(freq)-bark(f_Hz);
mask = 15.81 + 7.5*(dz+0.474) - 17.5*sqrt(1 + (dz+0.474).^2);

New_mask = (mask + spl - downshift);

m = New_mask;
```

Appendix D Listening Test Data

Set 1 – french at 96kbps										
track	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
DB	4.2	4.4	4.4	2	3	5	2.2	2.3	1	2.4
KH	3	2.7	2.5	1.5	2.2	5	1.9	2	1.1	1
AM	2	2	3	1	2	5	2	3.2	2	2
KL	3.6	4	1.4	1	2.5	4.5	1	4.1	1.5	3
AH	3	4	3.9	1	2	5	3	3.9	2.5	3
BR	4	3	2	2.5	3	5	1	3	2	3
JK	3.8	4.2	4.2	3.2	4.6	4.8	3.2	3.8	3.4	4.6
KK	3	3.1	3.2	2.1	2.2	4	2	2	1.9	1.9
PB	3.1	3	3.1	3	3	4.8	3	3	2.8	2.8
HS	4.2	4.3	4	3.2	3.9	4.5	3	3.5	2.8	2.5
KS	2.2	3.8	4	1.8	2.2	5	1.8	2.5	2	2.4
SS	2.5	4.5	4	2	3.8	4.8	2.9	3.5	1	3.5
TL	3	3.5	2.5	1.5	1.5	4.5	3	3	1.5	1
MJL	2.5	3	3	2.5	4	5	3.5	4	2	3
SJ	2	2.5	2.5	1	2	5	1.5	2.5	1.2	2
ML	3.2	3.5	3.5	1.5	3.2	4.5	2.7	3.3	2	3.5
DL	2.4	2.5	2.3	1.5	1	5	3	3.1	2.5	2.8
file	6nav	10awr	3awr	10off	10nav	orig	3off	6awr	6off	3nav
mean	3.04	3.41	3.15	1.90	2.71	4.79	2.39	3.10	1.95	2.61
std dev	0.72	0.76	0.86	0.76	0.97	0.29	0.78	0.67	0.69	0.89
95% conf	0.34	0.36	0.41	0.36	0.46	0.14	0.37	0.32	0.33	0.42
90% conf	0.29	0.30	0.34	0.30	0.39	0.12	0.31	0.27	0.28	0.36
sdg	-1.75	-1.38	-1.64	-2.89	-2.08		-2.39	-1.69	-2.84	-2.18

Set 2 – castanets at 96kbps										
track	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
DB	4	3.8	2.5	4.2	3	4.7	5	2	2.8	4.2
KH	2.1	1.9	1.5	2.8	2	2.2	4.5	1	1.5	2.9
AM	1.3	1.2	1	2	1	3	5	1	1	1
KL	4	2.5	2	1.7	1	2	4.5	1	1.5	1.8
AH	3	2	1	1	1	1	5	1	1	1
BR	2	2	2	3	2	3	5	1	3	2
JK	4.7	3.5	3	4.6	4.2	4.6	4.8	3.4	3.6	4.2
KK	2.1	2.1	1.9	1.8	1.6	1.9	4	1.2	1.1	1.3
PB	3.1	3.4	2.8	3.1	3.1	4.5	4.6	2.5	2.9	3.1
HS	4.3	4.5	3.8	3.4	3	4.5	4.8	2.3	2	2.5
KS	1.4	1.2	1.2	2.5	2.4	4	5	1	1.2	4
SS	2	2.5	1	3	2.5	3.5	4.5	1	1.5	2
TL	3.5	3	2	3.7	3.7	4	4.5	1	2	4
MJL	4.5	4	2	1	2	3	4.5	1	1.5	2
SJ	2	1.5	1	2	1	1.5	5	1	1.5	2.5
ML	2.5	2	1.7	3	3	3.3	4.6	1.5	2.4	3
DL	2	2.1	2.2	2.5	2.5	3.5	5	1.1	1	2
file	3off	3nav	6off	6awr	6nav	3awr	orig	10off	10nav	10awr
mean	2.85	2.54	1.92	2.66	2.29	3.19	4.72	1.41	1.85	2.56
std dev	1.12	0.99	0.79	1.01	0.98	1.15	0.29	0.71	0.81	1.07
95% conf	0.53	0.47	0.38	0.48	0.46	0.55	0.14	0.34	0.39	0.51
90% conf	0.45	0.40	0.32	0.40	0.39	0.46	0.12	0.28	0.32	0.43
sdg	-1.87	-2.18	-2.81	-2.06	-2.43	-1.54		-3.31	-2.87	-2.16

Set 3 – french at 128kbps										
track	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*
DB	4.5	4.5	5	4.5	4.5	4.5	4.5	4.5	4.5	2.5
KH	1.9	1.3	5	2.8	2.7	2.1	2	1.8	1.9	1.1
AM	2	2	5	3	3.6	2	4	2	4	2
KL	2.5	3	4.5	3.5	3.8	3.2	2.3	2.9	3.6	1.3
AH	3	4.5	5	4	4	4	4	3.5	4.5	3.5
BR	3	4	5	4	4	3	3	3	4	3
JK	4	4.2	4.4	3.8	3.8	3.8	3.8	3.8	3.8	3.8
KK	2	2.2	4	2.5	2.4	2.1	2.1	2	2.1	1.9
PB	3.3	3.3	4.5	3.8	3.5	3.3	3.4	3.2	3.2	3.2
HS	2.5	2.7	4	3.3	3.1	3	2.9	2.9	2.7	2.4
KS	2.5	2.6	5	4	3.9	3.8	3.9	3.7	4.2	2.5
SS	2	3.5	4	3.5	4	3.5	4	3.5	3.8	3
TL	2	3	4	3	3.5	3	3.5	3.5	3.7	3
MJL	1	2.5	4.5	3	3	2.5	4	3.5	3	3
SJ	1	1.5	5	2	2.5	1.5	2.5	2	2.5	1.5
ML	1.5	2.4	4.5	3	3.7	3.3	2.7	2.5	2.5	1.7
DL	2	2.1	5	2.2	1.9	1	1.8	1.7	1.5	1.1
file	10off	6nav	Orig	3nav	3awr	10nav	6awr	3off	10awr	6off
mean	2.39	2.90	4.61	3.29	3.41	2.92	3.20	2.94	3.26	2.38
std dev	0.95	0.99	0.42	0.69	0.70	0.94	0.85	0.82	0.93	0.85
95% conf	0.45	0.47	0.20	0.33	0.33	0.44	0.40	0.39	0.44	0.40
90% conf	0.38	0.39	0.17	0.28	0.28	0.37	0.34	0.33	0.37	0.34
sdg	-2.22	-1.71		-1.32	-1.21	-1.69	-1.41	-1.67	-1.35	-2.23

Set 4 – castanets at 128kbps										
track	*31*	*32*	*33*	*34*	*35*	*36*	*37*	*38*	*39*	*40*
DB	3.5	4	3.2	3.5	2.3	2.2	4.2	3.5	2.5	4.4
KH	4.3	5	3	2.7	1.2	1.5	2.1	1.6	2.3	1.9
AM	2.7	5	2	3.5	1	1	3.7	3.2	2	4
KL	3	3.7	3.5	3	1	3.7	3.2	2.7	2.6	3
AH	2	5	2	1.5	1	1.5	1.5	2	1.5	3.5
BR	3	5	2	3	2	3	2	2	2	3
JK	3.8	4	3.6	3.8	3	3	3.4	3.6	3.2	3.4
KK	2.8	4.1	2.6	2.7	1.8	1.7	3.5	2	1.9	3
PB	4.1	4.1	4	4.1	3.8	3.5	3.1	3.1	3.4	3.5
HS	4.5	4.7	4.3	4.5	3	2.7	2.5	2.3	2.5	2.6
KS	3	5	1.7	2.1	1.5	1	4	2.5	2.2	3.4
SS	2	4.5	1.5	2.5	1	2	3	2.5	3	3.5
TL	2.7	3.5	2	3	1.5	1.5	3	2.5	2	3
MJL	4	5	3.5	2	1	2	3	2	3.5	4
SJ	2	5	1.5	2.5	1	1	2.5	2	1.5	2.5
ML	2.7	4.8	3.3	3.7	2	3	3.7	3.5	3.3	3.6
DL	2.7	5	2.5	2.9	1	1.1	2	1.3	1.2	3
file	3off	orig	10nav	3awr	10off	6off	10awr	6nav	3nav	6awr
mean	3.11	4.55	2.72	3.00	1.71	2.08	2.96	2.49	2.39	3.25
std dev	0.80	0.53	0.90	0.78	0.87	0.91	0.77	0.69	0.70	0.61
95% conf	0.38	0.25	0.43	0.37	0.41	0.43	0.36	0.33	0.33	0.29
90% conf	0.32	0.21	0.36	0.31	0.35	0.36	0.31	0.28	0.28	0.24
sdg	-1.45		-1.84	-1.55	-2.84	-2.47	-1.59	-2.06	-2.16	-1.30