國 立 交 通 大 學

電機與控制工程學系

碩 士 論 文

MPEG-2/4 低複雜度先進音訊編碼演算法最佳化及在 StrongARM 平台上之實現

MPEG-2/4 LOW COMPLEXITY AAC ENCODER OPTIMIZATION AND IMPLEMENTATION ON A StrongARM PLATFORM

研 究 生：林映伶

指導教授：吳炳飛 教授

中華民國 九十四 年 七 月

MPEG-2/4 低複雜度先進音訊編碼演算法最佳化及在 StrongARM 平台上之實現

研　究　生：林映伶　　　　Student : Yin-Ling Lin

指導教授：吳炳飛 教授　　Advisor : Prof. Bing-Fei Wu

國立交通大學

電機與控制工程學系

碩士論文

A Thesis
Submitted to Department of Electrical and Control Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
In Partial Fulfillment of the Requirements
For the Degree of Master
In
Electrical and Control Engineering
July 2005
Hsinchu, Taiwan, Republic of China

中華民國 九十四 年 七 月

MPEG-2/4 低複雜度先進音訊編碼演算法最佳化及在 StrongARM 平台上之實現

研究生：林映伶　　　　　　　　指導教授：吳炳飛 教授

國立交通大學電機與控制工程學系碩士班

摘要

這篇論文提出一套 AAC 編碼的最佳化演算法以及在 AAC 編碼系統中加入資料嵌入演算法的應用。最後將這兩套系統實現在一顆 206 MHz 的 32 位元定點處理器 StrongARM SA-1110 上。實驗結果顯示，我們所提出的架構在實驗平台上可執行至少一倍速的壓縮。在 AAC 編碼最佳化中，我們移除計算量龐大的長短窗轉換，簡化 TNS 及 M/S 立體聲編碼的控制流程，數學函式的簡化運算及較快速的量化模組，在 MDCT 的實現方式上，也採用了以快速演算法。為了彌補定點化過程中所產生的誤差，我們加入了頻寬控制及動態精確度的 MDCT 運算等。最後，為了進一步增加 AAC 檔案的功能性，並在 AAC 編碼系統中加入資料嵌入的應用。

MPEG-2/4 LOW COMPLEXITY AAC ENCODER OPTIMIZATION AND IMPLEMENTATION ON A StrongARM PLATFORM

Student : Yin-Ling Lin Advisor : Prof. Bing-Fei Wu

Department of Electrical and Control Engineering

National Chiao Tung University

ABSTRACT

In this thesis, we present an optimized AAC encoding scheme and also proposed a data embedded method integrated into AAC encoding system. Both of them are finally realized on a 32-bit fixed-point processor, StrongARM SA-1110. Experimental result shows that at least 1 encoding speed is achieved. In the AAC encoding algorithm, we propose several approaches including the removal of block switching, fast MDCT, simplified TNS, simplified M/S stereo coding, mathematical function optimization and fast quantization. To compensate the error caused by fixed-point conversion, a bandwidth control and a dynamic data precision MDCT are applied. Finally, a data embedded method is implemented to further increase its utility.

ACKNOWLEDGEMENTS

首先，要感謝我的指導教授吳炳飛老師，這兩年來在研究方式及研究態度上的指導，讓我受益良多。尤其提供了一個資源充足的研究環境，找到我想做的東西，還有參賽寫書的經驗，都是我在進研究所之前所沒想過的。也許進交大念研究所並不在我原先一直以為的人生規劃表上，但真的很感恩目前為止經歷過的一切，很感謝吳老師給我的機會。

此外，最要感謝的是旭哥，不但常在學術研究上給我很多的建議，上班後仍然很有義氣的帶我們 meeting 到畢業。還有在進入音訊編碼領域幫忙很多的鐵男學長，教導實現資料嵌入的榮煌學長，在系統上幫忙很多的俊傑，剛進研究所常載我上下學的螞蟻學姐，及常請大家吃水果的飛鼠學姐。

當然不能不提的是一起八卦、shopping 的好姐妹，元馨學妹，一起吃飯、玩樂、幫忙搬家的培恭、晏阡及學弟小熊、宗堯，煮咖啡老歌同好重甫學長，還有幫忙買電腦的子萱學弟，一起做實驗的皓昱學弟。如果沒有你們，我的研究所生活一定會單調許多吧！

最後，最要感謝的是一直支持著我的父母、姐姐，提供我很好的生活條件，讓我可以無後顧之憂的專心學業。還有把我像親生女兒一樣照顧的新竹親戚，大姑姑、和英阿姨。還有許多感謝無法一一列述，謹以這篇論文獻給所有在研究生涯幫助過我的人，謝謝你們！

林映伶

民國九十四年七月 於新竹

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. Introduction

1.1 Background

With the rapid development of computer science, our life style has been changed a lot in recent years. Data are digitalized and distributed through Internet, wireless, and communication. Due to the limited bandwidth or storage space, data compression has become an important issue. Focusing on audio field, lots of audio codecs have been proposed in past years. For example, MPEG-1 layer III, generally known as MP3, has gained its popularity. Though there still has the vagueness of legalization, one can not deny that digital audio will gradually replace the traditional music market seems to be an irresistible trend.

1.2 Motivation

We’ve seen the wildly popularity of MP3 format. The request for higher coding efficiency and multichannel support drives the development of AAC format. As compared to MP3, AAC provides higher-quality results with smaller file sizes, higher resolution and multichannel support. AAC proves itself worthy of replacing MP3 as the new Internet audio standard.

Because of its superior performance and quality, AAC also constitutes the kernel of MPEG-4 audio and has been adopted in several application areas as Internet streaming, ISDN music transmission, high definition television (HDTV), satellite and terrestrial digital audio broadcasting and for audio transmission in third generation mobile networks (as 3GPP and 3GPP2 for UMTS/CDMA2000). Especially, Apple Computer, arms with its cool devices and online music store, spares no efforts to promote AAC format. AAC is gaining its importance in the market.

1.3 Innovation

Most AAC encoders are restricted to PC-based applications, since it consumes too much computational resources to implement on portable devices which often powered by batteries.

In this thesis, we present an optimized AAC encoding algorithm which can be implemented on power-limited portable systems. An additional functionality of data embedded technique is also presented. Finally, this proposed AAC encoding algorithm with data embedding feature is realized with a 206MHz 32-bit RISC CPU, Intel® StrongARM SA-1110. At least 1X encoding speed is achieved.

1.4 Content Organization

There are 7 chapters in this thesis. Chapter 2 and Chapter 3 briefly explain the AAC encoding algorithm. Chapter 2 focuses on psychoacoustic model and chapter 3 discusses the remaining modules. Chapter 4 describes our proposed optimization for MPEG-2 low complexity (LC) profile AAC. Chapter 5 illustrates the implementation of this modified MPEG-2 LC AAC encoder with Intel® StrongARM SA-1110 RISC CPU. A data embedded method specific to AAC file is also realized, and this is described in Chapter 6. The final experimental results can be found in Chapter 7. Chapter 8 concludes the whole thesis and discusses some future possibilities.

CHAPTER 2. Psychoacoustic Model

Most current audio coders achieve compression by exploiting the fact that "irrelevant" information is not detectable in general case. Irrelevant information is identified by incorporating several psychoacoustic principles.

2.1 The Absolute Threshold of Hearing

The absolute threshold of hearing, also called the quiet threshold, can be described as the amount of energy needed in a pure tone such that it can be detected by a listener in a noiseless environment. It is well approximated [18] by the nonlinear function:

$$T_q(f) = 3.64\left(\frac{f}{1000}\right)^{-0.8} - 6.5e^{-0.6\left(\frac{f}{1000}-3.3\right)^2} + 10^{-3}\left(\frac{f}{1000}\right)^4 \qquad (dB\quad SPL) \qquad (2.1)$$

It is the representative of a young listener with acute hearing. When $T_q(f)$ is applied to signal compression, it can be interpreted as a maximum allowable energy level for coding distortions introduced in frequency domain. Fig. 2.1 shows the curve of the absolute threshold of hearing.

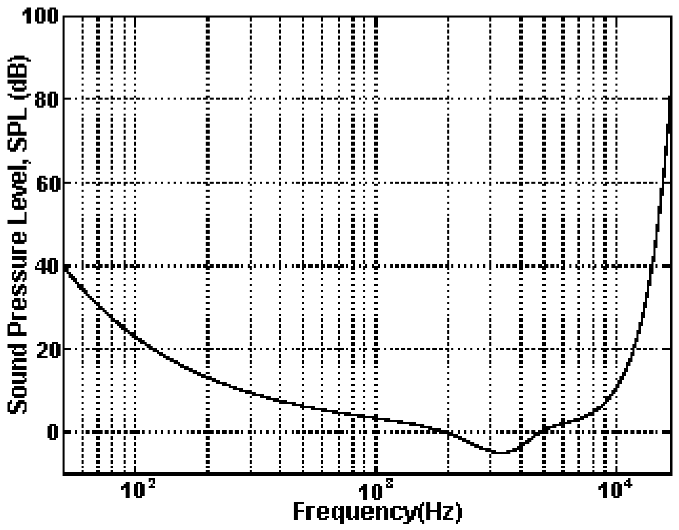


Fig. 2. 1 The absolute threshold of hearing [17]

The sound pressure level (SPL) is a measurement of sound intensity which is calculated as

$$\text{Number of decibels} = 10\log_{10}\left(\frac{I_1}{I_0}\right) \tag{2.2}$$

where,

I_0 is the reference intensity,

The most commonly used reference intensity is 10^{-12} (W/m^2) [8].

I_1 is the intensity to be measured.

2.2 Critical Band

It turns out that a frequency-to-place transformation takes place in the inner ear, along the basilar membrane. Distinct regions in the cochlea, each with a set of neural receptors, are responsible for a limited range of frequencies. This limited frequency resolution can be expressed in terms of "*critical band*". From the experimental sense, critical bandwidth can be loosely defined as the bandwidth at which subjective responses change abruptly. Fig. 2.1 shows how critical bands correspond to frequency domain. Table 2.2 illustrates the nonuniform Hertz spacing of the critical band.

Band NO.	Fc(Hz)	Bandwidth(Hz)	Band NO.	Fc(Hz)	Bandwidth(Hz)
1	50	0-100	14	2150	2000-2320
2	150	100-200	15	2500	2320-2700
3	250	200-300	16	2900	2700-3150
4	350	300-400	17	3400	3150-3700
5	450	400-510	18	4000	3700-4400
6	570	510-630	19	4800	4400-5300
7	700	630-770	20	5800	5300-6400
8	840	770-920	21	7000	6400-7700
9	1000	920-1080	22	8500	7700-9500
10	1170	1080-1270	23	10500	9500-12000
11	1370	1270-1480	24	13500	12000-15500
12	1600	1480-1720			
13	1850	1720-2000			

Table 2. 1 Critical bands. F_c – center frequency of the critical band [16]

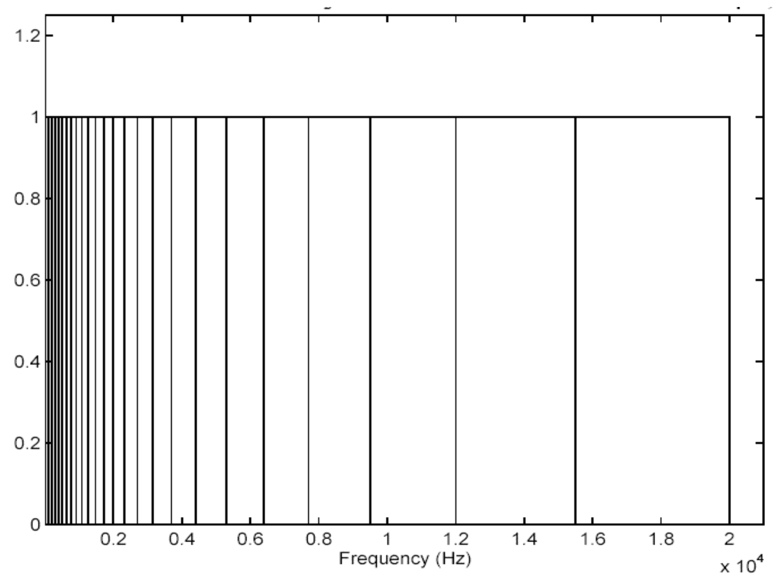


Fig. 2. 2 Idealized critical band filterbank [17]

2.3 Masking Effects

Masking refers to a process where one sound turns out to be inaudible because of the presence of another sound. There are two types of masking: simultaneous masking and temporal masking.

2.3.1 Simultaneous Masking

Simultaneous masking occurs in frequency domain, thus it is also called frequency masking. A simplified explanation of the mechanism is as follows: The presence of a strong signal (masker) creates an excitation of sufficient strength on the basilar membrane at the critical band location to effectively block the transmission of a weaker signal (maskee) [19]. This phenomenon has been observed both within a single critical band and between critical bands. The latter one is also known as the *spread of masking*. Fig. 2.3 gives an example of simultaneous masking with a masker at 150Hz.

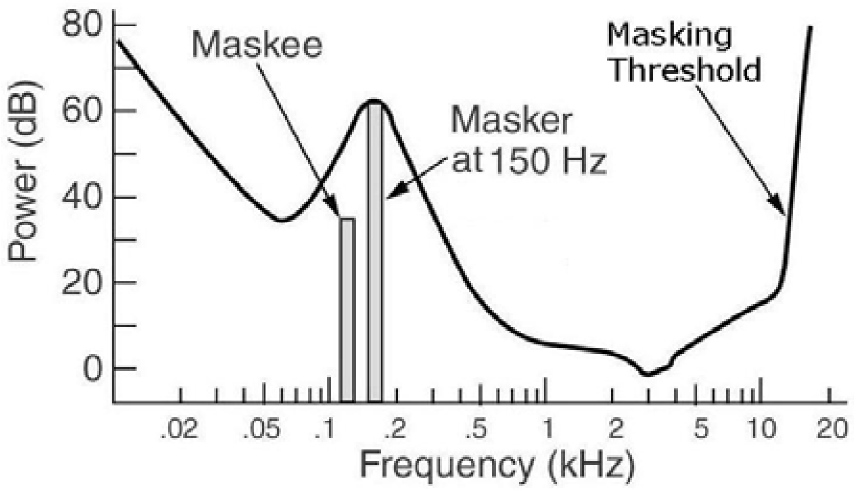


Fig. 2. 3 An example of simultaneous masking [19]

2.3.2 Temporal Masking

Masking effect is also happened in time domain which is called the temporal masking or nonsimultaneous masking. It is the term describing those situations where sounds are hidden due to maskers which have just disappeared (this is also called *post-masking*), or maskers which are about to appear (this is also called *pre-masking*). In the context of audio signal analysis, abrupt signal transients often creates *pre-masking* and *post-masking* regions in time during which a listener will not perceive signals beneath the elevated audibility thresholds produced by a masker.

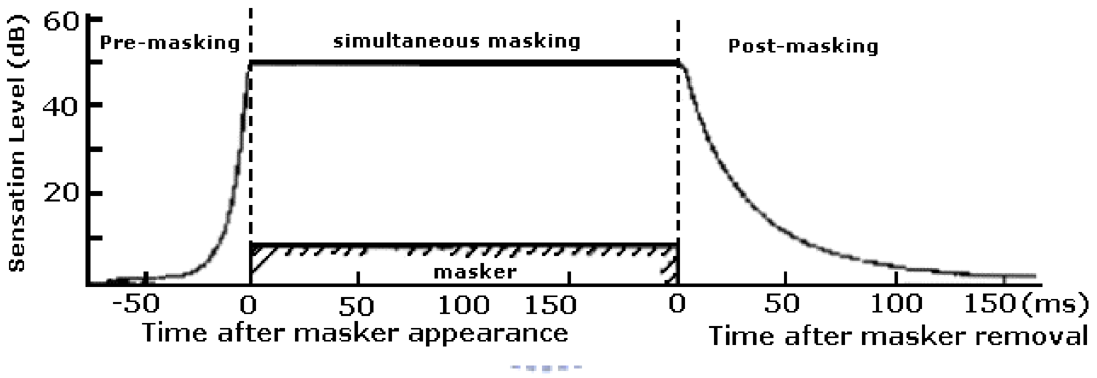


Fig. 2. 4 Schematic representation of temporal masking effect [16]

2.4 Psychoacoustic Model

The MPEG audio algorithm compresses the audio data in large part by removing the acoustically irrelevant parts of the audio signal. Psychoacoustic model (PM) exploits the masking effect of the human auditory system to calculate maximum allowable amount of quantization noise. This maximum level is referred to masking threshold. PM also uses this information along with input signal to decide bit allocation and block type switching.

The psychoacoustic model used for AAC system is similar to the one used in MPEG-1 audio [1]. The simplified description of its process is as follows:

1. Performing a 2048-point or 256-point FFT.
2. Using FFT-transformed spectrum to calculate the unpredictability measure.
3. Calculating the threshold (part I) by input signal energy and considering the quiet threshold.
4. Computing perceptual entropy (PE) to determine which block size (long or short) to use.
5. Calculating the minimum of masking threshold (part II) of each scalefactor band (see Section 3.6.2).
6. Calculating the signal-to-mask ratio (SMR) for each scalefactor band, and sending them to the quantizer.

The outputs from the psychoacoustic model are:

1. A set of signal-to-mask ratios and thresholds.
2. The delayed time domain data (PCM samples), which re-used by the MDCT.
3. The block type for the MDCT.
4. An estimation of how many bits should be used for encoding in addition to the average bits.

A simplified block diagram of psychoacoustic model is plotted in Fig. 2.5.

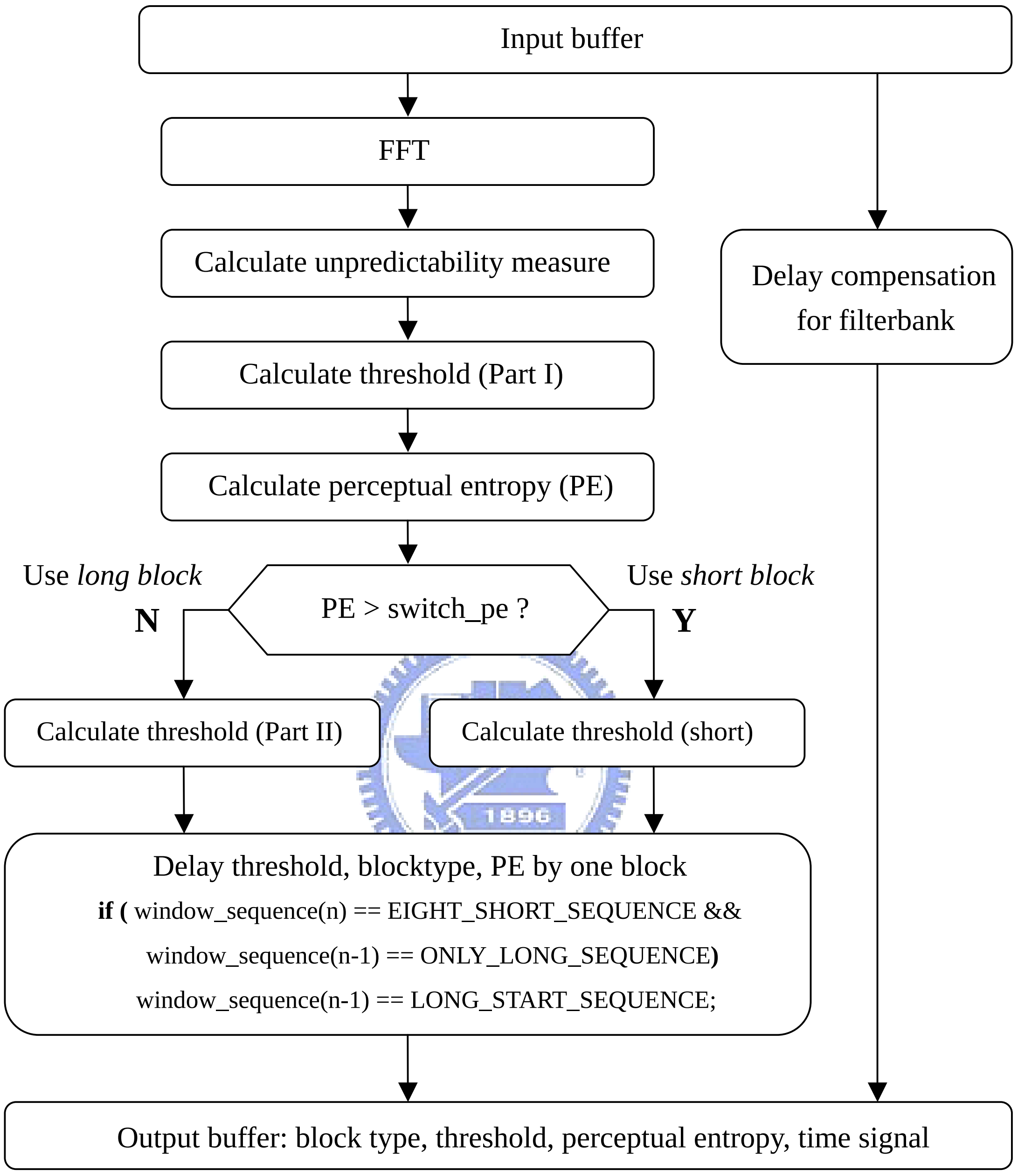

Input buffer
FFT
Calculate unpredictability measure
Calculate threshold (Part I)
Calculate perceptual entropy (PE)
Delay compensation for filterbank
Use long block
N
PE > switch_pe ?
Use short block
Y
Calculate threshold (Part II)
Calculate threshold (short)
Delay threshold, blocktype, PE by one block
if (window_sequence(n) == EIGHT_SHORT_SEQUENCE &&
window_sequence(n-1) == ONLY_LONG_SEQUENCE)
window_sequence(n-1) == LONG_START_SEQUENCE;
Output buffer: block type, threshold, perceptual entropy, time signal

Fig. 2. 5 Block diagram of psychoacoustic model [2]

For more details about how psychoacoustic model implements, please see [2].

CHAPTER 3. MPEG-2 AAC Algorithm

3.1 Overview

Started in 1994, the ISO/IEC MPEG-2 advanced audio coding (AAC) system was designed to provide best audio quality without any restrictions due to compatibility requirements. It was finalized as an international standard in 1997 April (ISO/IEC 13818-7). The MPEG-2 AAC scheme also constitutes the kernel of the MPEG-4 audio standard. All *profiles* (will be introduced later) defined in MPEG-2 AAC also appear in MPEG-4 standard.

AAC can include 48 full-bandwidth audio channels in one stream plus 15 low frequency enhancement (LFE, limited to 120 Hz) channels. The sampling rates supported by the AAC system vary from 8 to 96 kHz, as shown in Table 3.1.

Sampling Frequency (Hz)	**Maximum Bitrate Per Channel (kbit/s)**
96000	576
88200	329.2
64000	384
48000	288
44100	264.6
32000	192
24000	144
22050	132
16000	96
12000	72
11025	66.25
8000	48

Table 3. 1 MPEG-2 AAC sampling frequencies and maximum data rates

The basic structure of the MPEG-2 AAC system is shown in Fig. 3.1 and Fig. 3.2. We can briefly describe the AAC encoder process as follows. First, a filter bank is used to decompose the input signal into spectral components. Based on the input signal, an estimate of current signal-to-mask ratio is computed by psychoacoustic model, which will be utilized in quantization stage in order to minimize the audible distortion.

After the analysis filter bank, the TNS technique permits the encoder to exercise control over the temporal fine structure of quantization noise. For multichannel signals, intensity stereo coding and M/S (M as in middle, S as in side) stereo coding are used to reduce irrelevancies and redundancies. The former allows for a reduction in the spatial information, the latter transmits the normalized sum and difference signals instead of the left and right signals.

The time-domain prediction tool further increases the redundancy reduction of stationary signals. Next, in the quantizer, the spectral components are quantized and coded with the aim of keeping the quantization noise below the masked threshold.

Finally, all quantized and coded spectral coefficients and control parameters are assembled to form the target AAC bit stream.

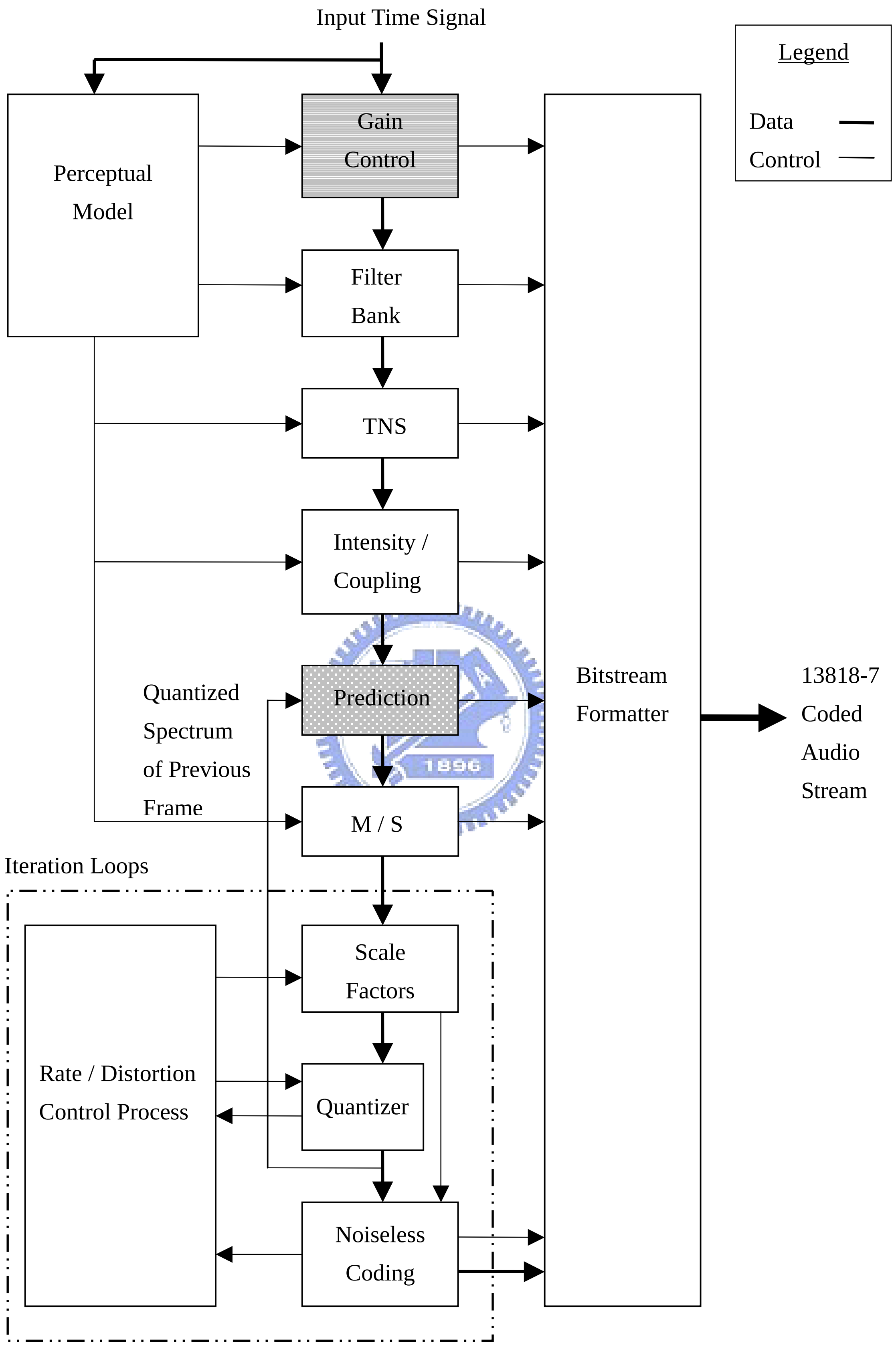


Fig. 3. 1 MPEG-2 AAC encoder block diagram

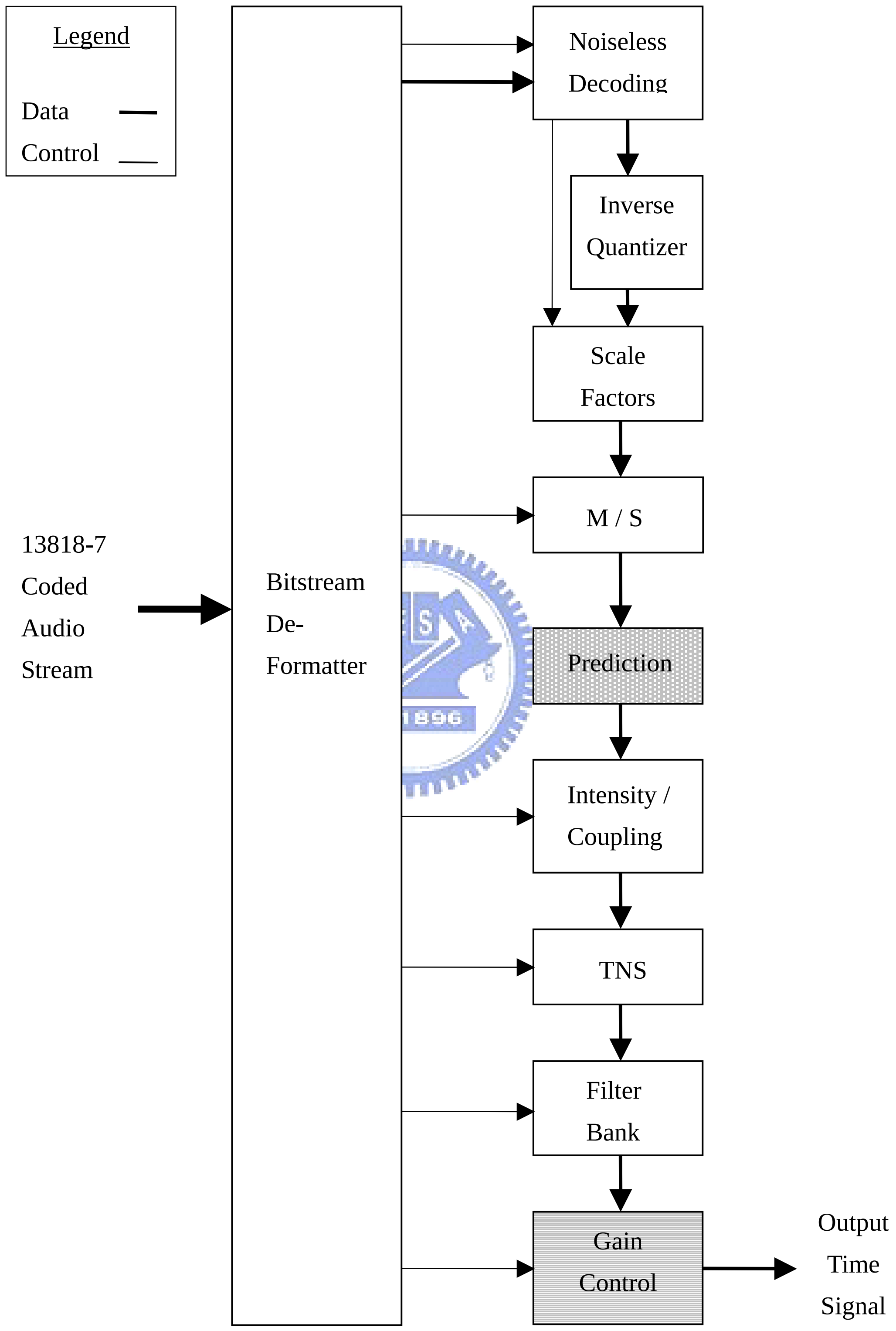


Fig. 3. 2 MPEG-2 AAC decoder block diagram

In order to allow a tradeoff among the quality, the memory and processing power requirements, the AAC system offers three profiles: main profile, low-complexity (LC) profile, and scalable sampling rate (SSR) profile.

- *Main profile*: In this configuration the AAC system provides the best audio quality at the given data rate. All AAC tools are applied except for gain control tool. Thus, it requires most computing power and memory usage.

- *Low-complexity (LC) profile*: The prediction tool and gain control tool are not employed and TNS order is limited in this configuration. Comparing to main profile, this reduces processing power and memory requirements.

- *Scalable sampling rate (SSR) profile*: Gain control tool is used only in this configuration. However, the prediction module is excluded and TNS order and bandwidth are limited. SSR profile provides the lowest complexity and a frequency scalable capability.

Table 3.2 describes the tools used by three profiles.

Tool Name	**Main Profile**	**LC Profile**	**SSR Profile**
Gain Control	╳	╳	○
Filter Bank	○	○	○
TNS	○	○*	○*
Intensity/Coupling	○	○	╳
Prediction	○	╳	╳
M/S	○	○	○
Quantizer	○	○	○
Noiseless Coding	○	○	○

* presented but limited

Table 3. 2 Tools used by three profiles

3.2 Filter Bank and Block Switching

Filter bank is a fundamental component of MPEG-2 AAC system that transforms the time-domain input signals into a time-frequency representation. This conversion is done by a forward modified discrete cosine transform (MDCT) in the encoder.

3.2.1 MDCT

The modified discrete cosine transform (MDCT) is a Fourier-related transform based on the type-IV discrete cosine transform (DCT-IV), with the additional property of being lapped. This overlapping, in addition to the energy-compaction quality of the DCT, makes the MDCT especially attractive for signal compression applications, since it helps to avoid artifacts stemming from the block boundaries.

In AAC system, the filterbank takes the appropriate block of input samples, modulates them by an appropriate window function, and performs the MDCT. Each block of input samples is overlapped by 50% with the immediately preceding block and the following block.

The expression for the MDCT is

$$X_i(k) = 2\sum_{n=0}^{N-1} x_{in}(n)\cos[\frac{2\pi}{N}(n+n_0)(k+\frac{1}{2})], \quad \text{for} \quad 0 \le k < \frac{N}{2}, \qquad (3.1)$$

where

x_{in} = windowed input sequence,

n = sample index,

k = spectral coefficient index,

i = block index,

N = window length of the one transform window based on the window_sequence value,

n_0 = (N/2 + 1) / 2,

3.2.2 Window Shape

The frequency selectivity of an MDCT filter bank is dependent on the window function. A window function commonly used in audio coding is the sine window. This window produces a filter bank with good separation of nearby spectral components. Another window function provided in AAC system is the Kaiser-Bessel derived (KBD) window which allows optimization of the transition bandwidth and the ultimate rejection of the filter bank.

Sine window coefficients are given as follows:

$$W_{SIN\_LEFT,N}(n) = \sin\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)\right) \quad for \quad 0 \le n < \frac{N}{2} \tag{3.2}$$

$$W_{SIN\_RIGHT,N}(n) = \sin\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)\right) \quad for \quad \frac{N}{2} \le n < N \tag{3.3}$$

Kaiser-Bessel derived window coefficients are given as follows:

$$W_{KBD\_LEFT,N}(n) = \sqrt{\frac{\sum_{p=0}^{n}[W'(n,\alpha)]}{\sum_{p=0}^{N/2}[W'(p,\alpha)]}} \quad for \quad 0 \le n < \frac{N}{2} \tag{3.4}$$

$$W_{KBD\_RIGHT,N}(n) = \sqrt{\frac{\sum_{p=0}^{N-n}[W'(n,\alpha)]}{\sum_{p=0}^{N/2}[W'(p,\alpha)]}} \quad for \quad \frac{N}{2} \le n < N \tag{3.5}$$

where
W', Kaiser-Bessel kernel window function is defined as follows:

$$W'(n,\alpha) = \frac{I_0\left[\pi\alpha\sqrt{1.0 - \left(\frac{n - N/4}{N/4}\right)^2}\right]}{I_0[\pi\alpha]} \quad for \quad 0 \le n \le \frac{N}{2} \tag{3.6}$$

$$I_0[x] = \sum_{k=0}^{\infty}\left[\frac{\left(\frac{x}{2}\right)^k}{k!}\right]^2 \tag{3.7}$$

α = kernel window alpha factor, $\alpha = \begin{cases} 4, & for \quad N = 2048 \\ 6, & for \quad N = 256 \end{cases}$

The AAC system allows seamless switching between KBD and sine windows. Perfect reconstruction is preserved in the filter bank during window shape changes. Fig. 3.3 shows the window shape switching process. The sequence of windows labeled A-B-C employs the KBD window, whereas the sequence D-E-F shows the transition to and from a single frame employing the sine window.

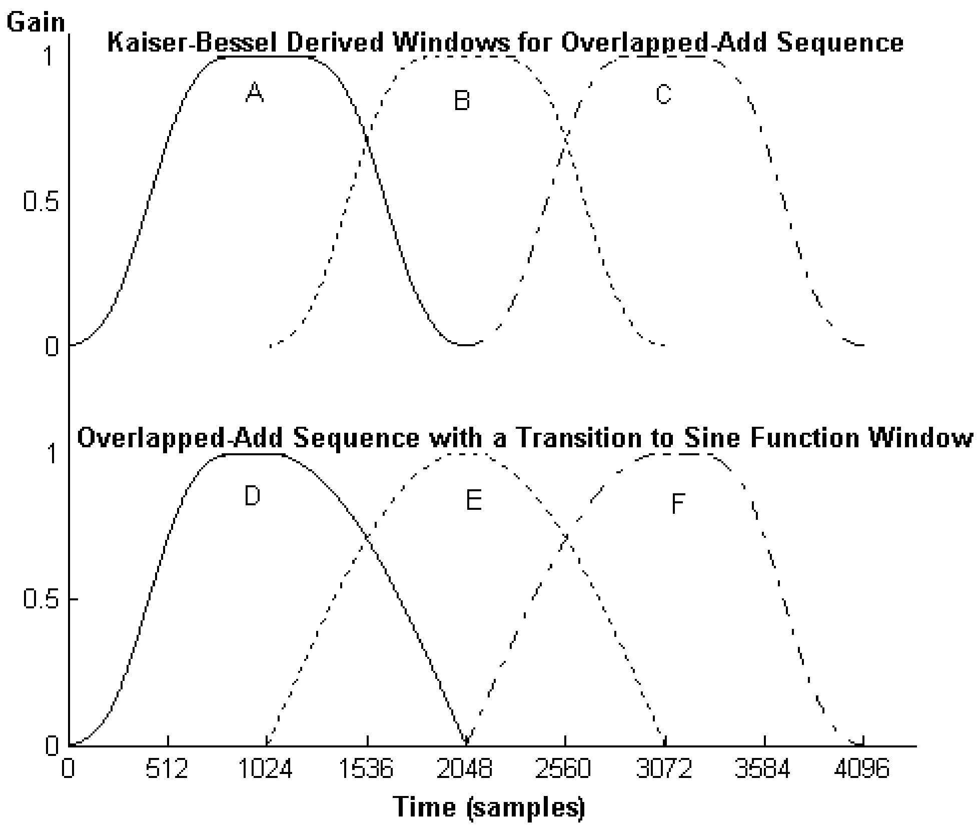


Fig. 3. 3 Example of window shape switching process [1]

3.2.3 Block Switching

To adapt the time-frequency resolution of the filter bank to the characteristics of the input signal, AAC system provides two kinds of transformation lengths: the long transformation with 2048 samples is termed a “long” sequence, while the short transformation occur in groups called “short” sequence. The short sequence is composed of eight short block transforms, and each with 256 samples.

This block switching, however, potentially creates a problem of block synchrony between the different channels being coded. To maintain block alignment and to preserve the time-domain aliasing cancellation properties of MDCT and IMDCT, a “start” and “stop” bridge window is used during transitions. Fig. 3.4 shows the window overlap-add process appropriate for both steady-state and transient conditions[1].

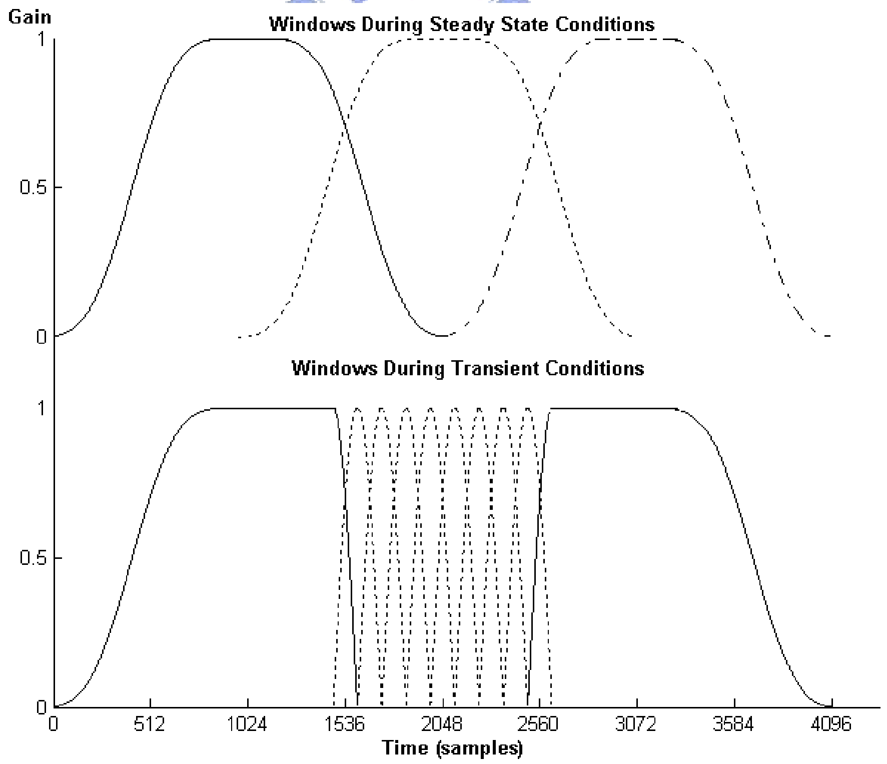


Fig. 3. 4 Comparison of window overlap-add processes for steady-state and transient conditions

According to the **window_sequence** and **window_shape** element, different transformation windows are used. All possible combinations are described as follows:

Let N = window length, we have:

$$window\_sequence\begin{cases} ONLY\_LONG\_SEQUENCE, & N = 2048 \\ LONG\_START\_SEQUENCE, & N = 2048 \\ EIGHT\_SHORT\_SEQUENCE, & N = 2048 \\ LONG\_STOP\_SEQUENCE, & N = 2048 \end{cases}$$

$$window\_shape\begin{cases} KBD & window \\ sine & window \end{cases}$$

a) ONLY_LONG_SEQUENCE

- **window_shape** = = KBD window

$$W(n) = \begin{cases} W_{LEFT,2048}(n), & for \quad 0 \le n < 1024 \\ W_{KBD\_RIGHT,2048}(n), & for \quad 1024 \le n < 2048 \end{cases}$$

- **window_shape** = = sine window

$$W(n) = \begin{cases} W_{LEFT,2048}(n), & for \quad 0 \le n < 1024 \\ W_{SIN\_RIGHT,2048}(n), & for \quad 1024 \le n < 2048 \end{cases}$$

b) LONG_START_SEQUENCE

- **window_shape** = = KBD window

$$W(n) = \begin{cases} W_{LEFT,2048}(n), & for \quad 0 \le n < 1024 \\ 1.0, & for \quad 1024 \le n < 1472 \\ W_{KBD\_RIGHT,256}(n+128-1472), & for \quad 1472 \le n < 1600 \\ 0.0, & for \quad 1600 \le n < 2048 \end{cases}$$

- **window_shape** = = sine window

$$W(n) = \begin{cases} W_{LEFT,2048}(n), & for \quad 0 \le n < 1024 \\ 1.0, & for \quad 1024 \le n < 1472 \\ W_{SIN\_RIGHT,256}(n+128-1472), & for \quad 1472 \le n < 1600 \\ 0.0, & for \quad 1600 \le n < 2048 \end{cases}$$

c) EIGHT_SHORT_SEQUENCE

The total length of the window_sequence together with leading and following zeros is 2048-bit. Each of the eight short blocks are windowed separately first. The short block number is indexed with the variable j = 0,1,…,7.

- **window_shape** = = KBD window

$$W_0(n) = \begin{cases} W_{LEFT,256}(n), & for \quad 0 \le n < 128 \\ W_{KBD\_RIGHT,256}(n), & for \quad 128 \le n < 256 \end{cases}$$

$$W_{1-7}(n) = \begin{cases} W_{KBD\_LEFT,256}(n), & for \quad 0 \le n < 128 \\ W_{KBD\_RIGHT,256}(n), & for \quad 128 \le n < 256 \end{cases}$$

- **window_shape** = = sine window

$$W_0(n) = \begin{cases} W_{LEFT,256}(n), & for \quad 0 \le n < 128 \\ W_{SIN\_RIGHT,256}(n), & for \quad 128 \le n < 256 \end{cases}$$

$$W_{1-7}(n) = \begin{cases} W_{SIN\_LEFT,256}(n), & for \quad 0 \le n < 128 \\ W_{SIN\_RIGHT,256}(n), & for \quad 128 \le n < 256 \end{cases}$$

d) LONG_STOP_SEQUENCE

- **window_shape** = = KBD window

$$W(n) = \begin{cases} 0.0, & for \quad 0 \le n < 448 \\ W_{LEFT,256}(n-448), & for \quad 448 \le n < 576 \\ 1.0, & for \quad 576 \le n < 1024 \\ W_{KBD\_RIGHT,2048}(n), & for \quad 1024 \le n < 2048 \end{cases}$$

- **window_shape** = = sine window

$$W(n) = \begin{cases} 0.0, & for \quad 0 \le n < 448 \\ W_{LEFT,256}(n-448), & for \quad 448 \le n < 576 \\ 1.0, & for \quad 576 \le n < 1024 \\ W_{KBD\_RIGHT,2048}(n), & for \quad 1024 \le n < 2048 \end{cases}$$

3.3 Temporal Noise Shaping

The handling of transient and pitched input signal has always been a challenge of today's audio coding, which is due to the so called "pre-echo" phenomenon.

3.3.1 Pre-echo Phenomenon

In psychoacoustic model, we exploit the perceptual effect of *simultaneous masking* (see Chapter 2). However, from Fig. 2.4, we observed that pre-masking, in the order of 2-5 ms, is much shorter than post-masking. At the same time, to achieve perceptually transparent coding quality, quantization noise must not exceed the time-dependent masking threshold.

This requirement is not easy to meet for perceptual coders. Because quantizing and coding in frequency domain implies that the quantization error introduced in this domain will be spread out in time domain after reconstruction. Assuming sampling rate is 44.1 kHz, AAC system performs 2048-point MDCT which means the quantization noise can be spread out over a period of more than 46 ms. In particular, if quantization noise is spread out before the onsets of the signal and in extreme cases may even exceed the original signal in level during certain time. Fig. 3.5 gives a pre-echo phenomenon example.

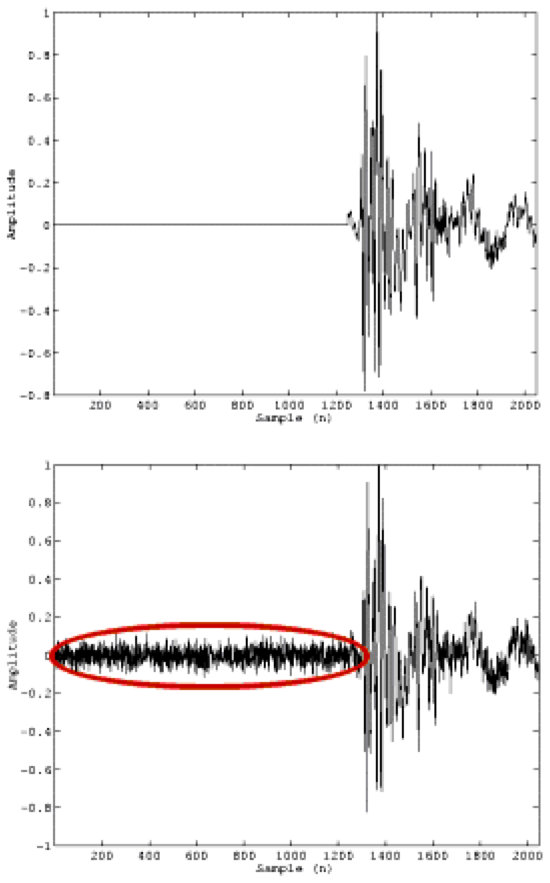


Fig. 3. 5 Pre-echo phenomenon example[39]

Some traditional techniques have been proposed to avoid pre-echo phenomenon, including bit reservoir, gain control and adaptive window switching. Here, AAC provides a new powerful tool called *temporal noise shaping* (TNS) to further exercise control over the temporal fine structure of the quantization noise even within a filter bank window.

3.3.2 TNS Processing

The basic concept of TNS can be outlined as follows:

- Time-frequency duality considerations:

It is well known that signals with an “unflat” spectrum can be coded efficiently either by directly coding spectral values or by applying predictive coding methods to the time domain signal [3]. According to duality between frequency and time domain, we can say that signals with an “unflat” time structure, that is, transient signals can be coded efficiently either by directly coding time-domain signals or by applying predictive coding methods to the spectral values. Table 3.3 summarizes this concept.

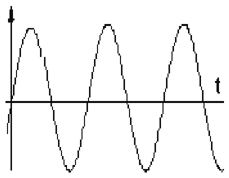
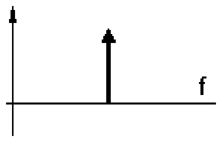
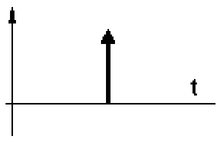
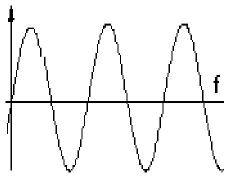
Input Signal		**Optimum Coding**	
Time Domain	Freq. Domain	Direct Coding	Predictive Coding
		Coding of spectral data	Prediction in time domain
		Coding of time domain data	*Prediction in frequency domain*

Table 3. 3 Optimum Coding Methods for Extreme Input Signal Characteristics [4]

- *Noise shaping by predictive coding*:

Although both open-loop and close-loop predictive coding techniques can be employed to provide coding gain, the distribution of quantization error in the final decoded signal are different. If a close-loop prediction scheme is used the error introduced in the final decoded signal has a “flat” power spectral density (PSD). However, if an open-loop prediction scheme is used, the PSD of its quantization error is known to adapt to the PSD of the input signal. This effectively puts the quantization noise under the actual signal and therefore avoids problems of temporal masking in either transient or pitched signals. Fig. 3.6 shows this concept of DPCM coding. For more details, please refer to [3].

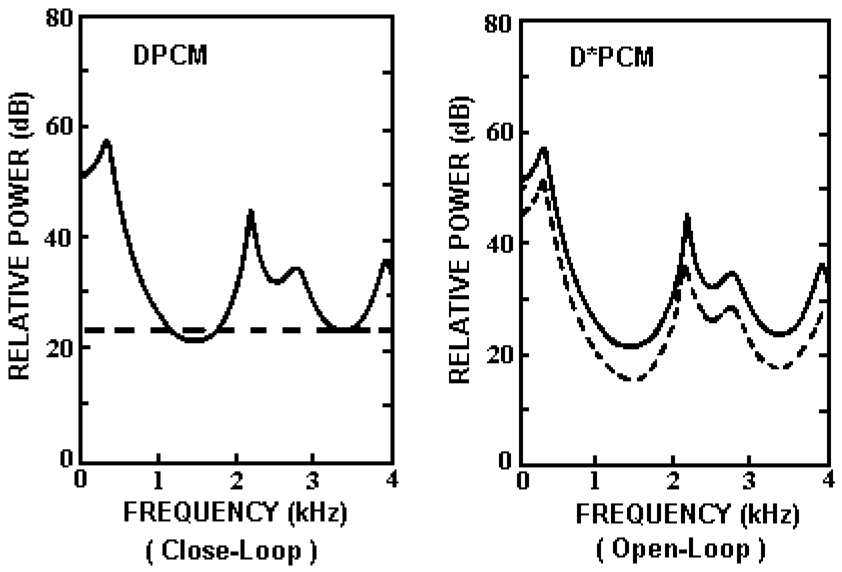


Fig. 3. 6 Comparisons of input (solid) and coding noise (dashed) spectrum [3]

This type of linear prediction coding of spectral data is referred to as the TNS method.

3.4 M/S Stereo Coding

AAC system includes two techniques for stereo coding of signals – mid/side (M/S) stereo coding (also known as sum-difference coding) and intensity stereo coding. Both stereo coding strategies can be combined by applying them to different frequency regions.

Middle/Side (M/S) stereo coding primarily has two effects: one is to control the imaging of coding noise, as compared to the imaging of the original signal. In particular, this technique is capable of addressing the issue of binaural masking level difference (BMLD). The other is simply to reduce interchannel redundancies.

3.4.1 Binaural Masking Level Difference

The masking threshold of a signal can sometimes be markedly lower down when listening by two ears than when listening by only one. Considering the situation shown in Fig. 3.7(a). White noise from the same noise generator is fed into both ears via headphones. Pure tones, also from the same signal generator, are fed separately into each ear and mixed with the noise. Thus the total signals in two ears are identical. Assuming that the level of the tone is adjusted until it is masked by the noise, i.e. it is at its masking threshold, and let this level be L_0 dB. Now that we invert the tone signal at only one ear, i.e. the phase of the tone signal is shifted by 180°, as shown in Fig. 3.7(b). The result is that the tone becomes audible again. The tone can be adjusted to a new level, L_π, and it is again its masking threshold. The difference between the two levels, $L_0 - L_\pi$ (dB), is known as a *binaural masking level difference* (BMLD).

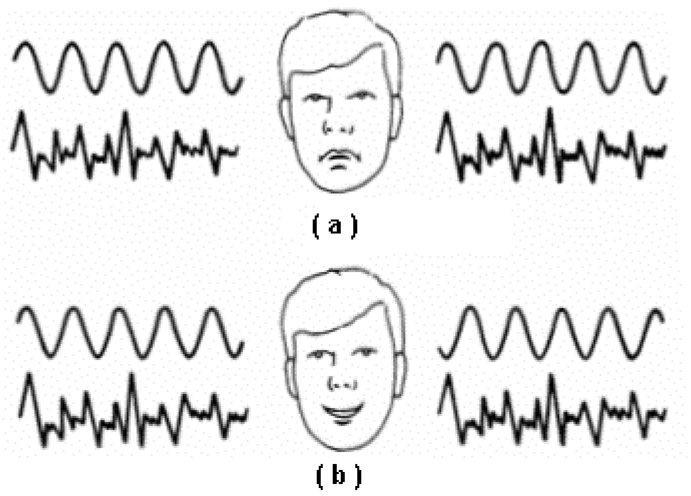


Fig. 3. 7 Illustration of the situation in which BMLD occurs [8]

BMLD value may be as large as 15 dB at low frequencies (around 500 Hz), decreasing to 2 dB for frequencies above 1500 Hz.

This phenomenon is not limited to pure tones. Similar effects have been observed for complex tones, transient and pitched signals. Our ability to detect and identify the signals depends on the phase of the signal and noise presented (or lack of correlation in the case of noise).

3.4.2 M/S Stereo Threshold

Due to the BMLD phenomenon stated previously, to prevent stereo unmasking, M and S, left and right thresholds are again calculated:

$$t = Mthr / Sthr$$

$$if\,(t > 1) \quad t = t^{-1}$$

$$Rfthr = \max\left(Rthr \times t, \quad \min\left(Rthr, b\max \times Rengy \right)\right)$$

$$Lfthr = \max\left(Lthr \times t, \quad \min(Lthr, b\max \times Lengy)\right)$$

$$t = \min(Lthr, Rthr)$$

$$Mthr = \min\left(t, \max\left(Mthr, \quad \min(Sthr, b\max \times Sengy)\right)\right)$$

$$Sthr = \min\left(t, \max\left(Sthr, \quad \min(Mthr, b\max \times Mengy)\right)\right)$$

where,
Mthr, Sthr, Rthr, Lthr are thresholds of M, S, right and left channel.
Mengy, Sengy, Rengy, Lengy are spread energy of M, S, R, L channel.
Mfthr, Sfthr, Rfthr, Lfthr are final thresholds
bmax represents BMLD protection ratio, as can be calculated from

$$b\max(b) = 10^{-3\left[0.5+0.5\cos\left(\pi\frac{\min(bval(b),15.5)}{15.5}\right)\right]} \qquad (3.8)$$

where, *bval(b)*: median bark value of b^{th} partition.

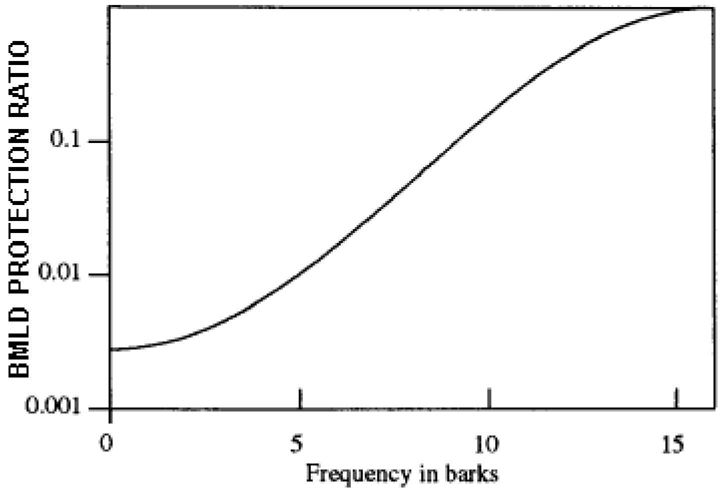


Fig. 3. 8 BMLD protection ratio (*bmax*) [6]

3.4.3 L/R and M/S Switching

If the difference between THR_l and THR_r is less than 2 dB, and the bits required is fewer than L/R mode, the coder will switch to M/S mode, i.e. the left signal for that given band of frequencies is replaced by $M = \frac{L+R}{2}$ and the right signal is replaced by $S = \frac{L-R}{2}$.

3.5 Intensity Stereo Coding

Intensity stereo coding exploits the fact that the human perception of high frequencies components relies on the analysis of the energy–time envelopes [7][8] to increase the reduction of irrelevancy at high frequencies. This is done based on the channel-pair concept as used for M/S stereo, the following explanation will use L/R pair for convenience. Instead of transmitting both left and right channel signals, a single representing signal plus directional information will be transmitted only. Thus, the reconstructed signals for the left and right channel consist of differently scaled versions of the same transmitted signals which have different amplitudes but have the same phase information. The energy-time envelope is preserved by means of the scaling operation; however, due to the loss of phase information, the waveform of the original signal is generally not preserved. The directional information, *is_position*, is computed as:

$$is\_position[sfb] = NINT\left(2 \times \log_2\left(\frac{E_l[sfb]}{E_r[sfb]}\right)\right) \tag{3.9}$$

Next, the intensity signal spectral coefficients $spec_i[i]$ are calculated for each scalefactor bands (see section 3.6.2) by adding spectral samples from the left and right channel (*spec$_l$[i]* and *spec$_r$[i]*) and rescaling the resulting values like

$$spec_i[i] = (spec_l[i] + spec_r[i]) \times \sqrt{\frac{E_l[sfb]}{E_s[sfb]}} \tag{3.10}$$

where
sfb is the index of scalefactor band,
NINT means "nearest integer",
E_l, E_r, E_s represent the energy of left, right and sum channel. The sum channel is calculated by summing the squared spectral coefficients.

The signal flow of an intensity stereo coding / decoding scheme is shown in the Fig. 3.9.

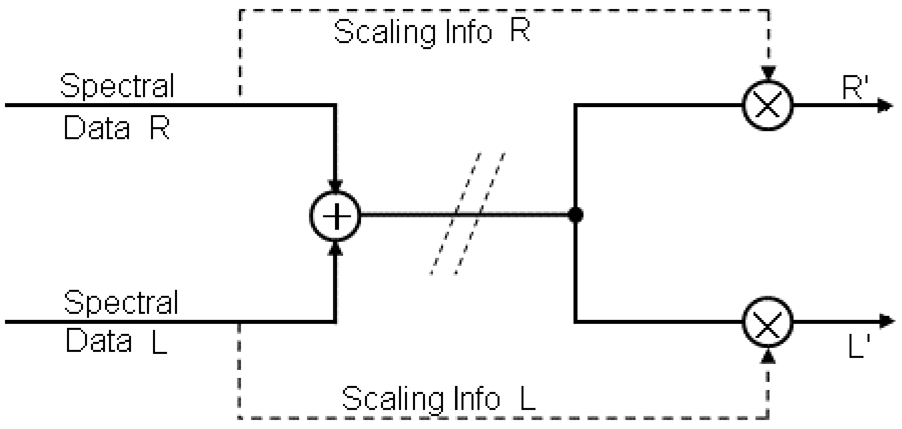


Fig. 3. 9 signal flow of an intensity stereo coding / decoding scheme [14]

3.6 Prediction

Prediction is used for an improved redundancy reduction and is especially effective if the signal is more or less stationary. Since a window sequence of type EIGHT_SHORT_SEQUENCE indicates signal changes, i.e. non-stationary signal characteristics, prediction is used only for long window. There is one corresponding predictor for each spectral component, resulting in a bank of predictors.

3.6.1 Predictor Structure

Backward-adaptive predictors are adopted in AAC system. The predictor coefficients are calculated from preceding quantized spectral components in the encoder as well as in the decoder. Thus, no additional side information is needed for the transmission of predictor coefficients. A second-order backward-adaptive lattice structure predictor is used for each spectral component, so that each predictor is working on the spectral component values of the two preceding frames.

Due to the realization in a lattice structure, the predictor contains two basic elements that are cascaded. The overall estimate results in

$$x_{est}(n) = x_{est,1}(n) + x_{est,2}(n) \tag{3.11}$$

In each element, the part $x_{est,m}(n), m = 1,2$, of the estimate is calculated according to:

$$x_{est,m}(n) = b \times k_m(n) \times a \times r_{m-1}(n-1) \tag{3.12}$$

where *backward prediction error at stage m*, $r_m(n)$, is calculated as:

$$r_m(n) = r_{m-1}(n-1) - b \times k_m(n) \times e_{m-1}(n) \tag{3.13}$$

and *forward prediction error at stage m*, $e_m(n)$:

$$e_m(n) = e_{m-1}(n) - x_{est,m}(n) \tag{3.14}$$

The *attenuation factors*, *a* and *b* are chosen as *a* = *b* = 0.953125.

And a least-mean-square (LMS) approach is used here, the *prediction coefficients*, k_m, are calculated as follows:

$$k_m(n+1) = \frac{\sum_{i=2}^{n} \alpha^{n-i} e_{m-1}(i) r_{m-1}(i-1)}{\frac{1}{2}\sum_{i=2}^{n} \alpha^{n-i}\left[e_{m-1}^{2}(i) + r_{m-1}^{2}(i-1)\right]} = \frac{COR_m(n)}{VAR_m(n)} \tag{3.15}$$

with

$$COR_m(n) = \alpha \times COR_m(n-1) + r_{m-1}(n-1) \times e_{m-1}(n) \tag{3.16}$$

$$VAR_m(n) = \alpha \times VAR_m(n-1) + \frac{1}{2}\left[r_{m-1}^{2}(n-1) + e_{m-1}^{2}(n)\right] \tag{3.17}$$

where α is an *adaptation time constant* which determines the influence of the current sample on the estimate of the expected values. The value of α is chosen to be 0.90625.

More explanations can be found in [9][10]. Fig. 3.10 shows the second-order backward-prediction lattice structure.

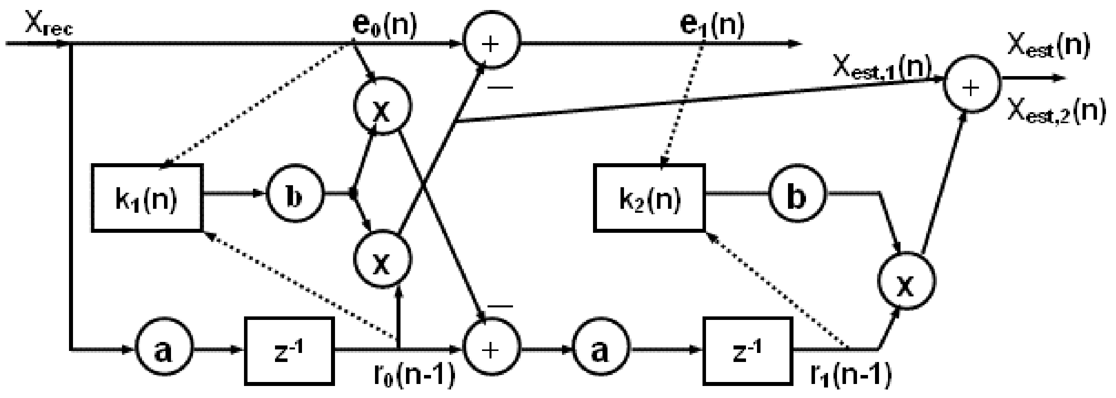


Fig. 3. 10 The second-order backward-prediction lattice structure [1]

3.6.2 Predictor Control

A predictor control is required to guarantee that there is a prediction gain. The decision, whether the predictor is on or off, is made in the unit of one scalefactor band (see Section 3.6.2). Two considerations are taken into account - First, if prediction gives a prediction gain in that scalefactor band, and all predictors belonged switch on or off accordingly. Second, whether the overall coding gain of current frame compensates at least the additional bits needed for the prediction side information. Prediction is activated only if the above two conditions are met. In order to increase the stability of the predictors, a cyclic reset mechanism is applied, in which all predictors are initialized in a certain time interval. The whole set of predictors are subdivided, in an interleaving way, into 30 so-called reset groups.

Reset Group Number	**Predictors of Reset Group**
1	P_0, P_{30}, P_{60}, P_{90}, ….
2	P_1, P_{31}, P_{61}, P_{91}, ….
3	P_2, P_{32}, P_{62}, P_{92}, ….
….	….
30	P_{29}, P_{59}, P_{89}, P_{119}, ⋯

Table 3. 4 Reset groups of predictors [2]

Fig. 3.11 shows the block diagram of the prediction unit for one single predictor of the predictor bank. P – predictor ; Q – quantizer ; REC – reconstruction of last quantized value. For more detailed description of the principles can be found in [11].

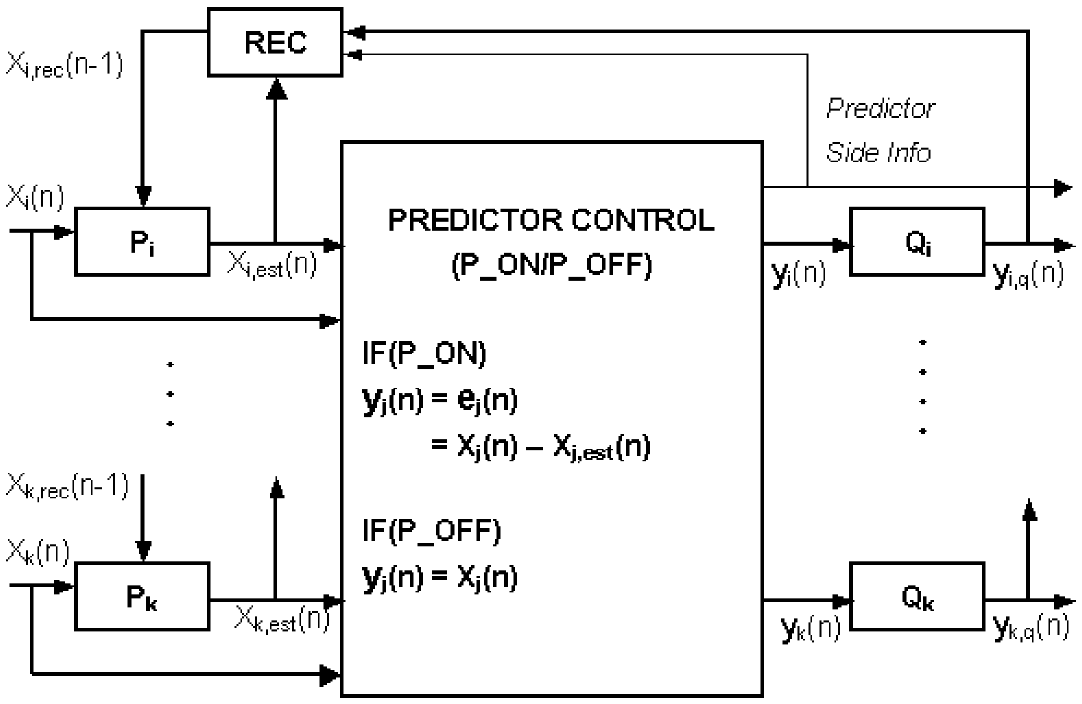


Fig. 3. 11 Block diagram of prediction unit for one scale factor band [2]

3.7 Quantization

The primary goal of quantization is to quantize spectral data in such a way that the quantization noise fulfills the demands of the psychoacoustic model, and at the same time, the bits required must also be below a certain limit, normally the average number of bits available for a block of audio data.

3.7.1 Nonuniform Quantization Function

The nonuniform quantizer used in AAC is shown as follow:

$$x\_quant = \text{int}\left\{\left[\frac{|xr(i)|}{\sqrt[4]{2^{scalefactor-common\_scalefactor}}}\right]^{0.75} - MAGIC\_NUMBER\right\} \qquad (3.18)$$

where MAGIC_NUMBER is defined to 0.4054, and scalefactor will be described in

Section 3.6.2.

The main advantage of nonuniform quantizer is the implicit noise shaping depending on the coefficient amplitudes. Its increasing signal-to-error ratio (SNR) with rising signal energy is much lower than that in a linear quantizer.

3.7.2 Scalefactor Band

Noise shaping in quantization process is done by using scalefactors. For this purpose, spectrum is divided into several (depending on sampling rates) scalefactor bands which is very similar to the critical bands of human auditory system. Each scalefactor band has a scalefactor that represents a certain gain value. All spectral coefficients belong to that scalefactor band will be rescaled by their scalefactor. The noise shaping is therefore achieved because amplified coefficients have larger amplitudes, and thus obtained a higher SNR after quantization.

3.7.3 Iteration Process

Optimum quantization is done by an iteration process consisting of two nested loops, an inner loop which is aimed at controlling coding bits required and an outer loop which is used to shape the quantization noise. The overall iteration process can be shown as Fig. 3.12.

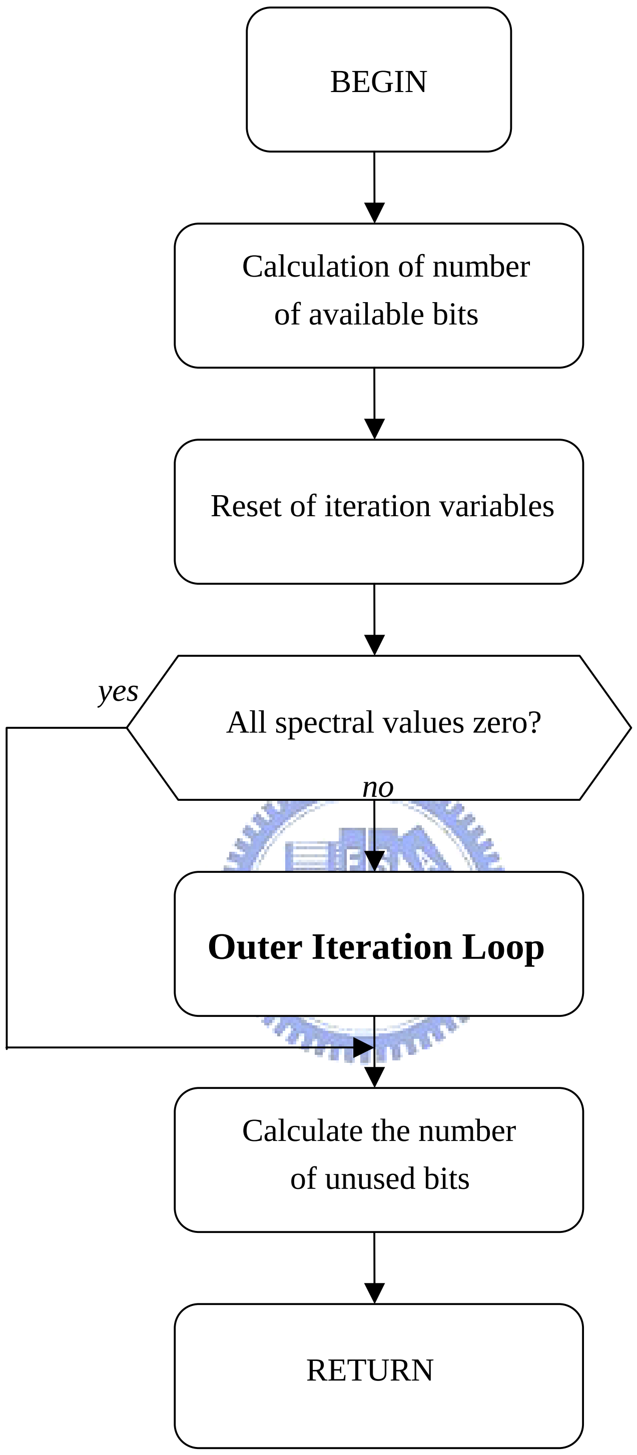

BEGIN
Calculation of number of available bits
Reset of iteration variables
yes
All spectral values zero?
no
Outer Iteration Loop
Calculate the number of unused bits
RETURN

Fig. 3. 12 A simplified block diagram of iteration process

3.7.3.1 Inner Iteration Loop

The task of inner iteration loop is controlling the coding bit required by adjusting quantizer step size until all spectral data can be encoded with the number of available bits. If the number of bits needed for encoding is higher than available bits, the quantizer step size is increased, and the process will repeat until reaching its goal. Thus, inner iteration loop is also called rate control loop. A simplified processing description is as follows:

1. At the beginning, the spectral data are quantized by nonuniform quantization function (3.18).
2. Number of bits required to encode the quantized data is counted.
3. If the number of bits required is higher than the number of available bits, the quantizer step size is increased, and go back to (1), repeating the whole process.
4. Else, the inner iteration loop is ended.

A simplified block diagram of inner iteration loop is shown in Fig. 3.13.

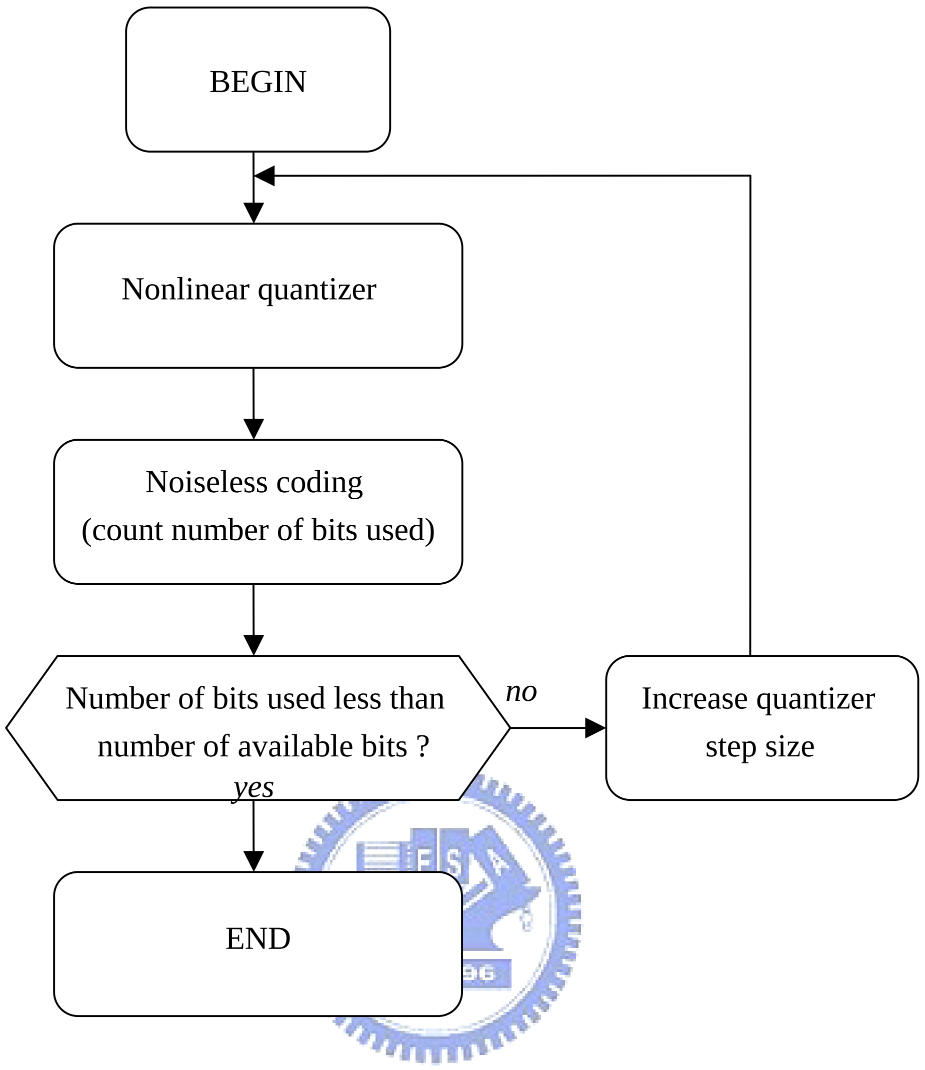

BEGIN
Nonlinear quantizer
Noiseless coding
(count number of bits used)
Number of bits used less than
number of available bits ?
no
Increase quantizer
step size
yes
END

Fig. 3. 13 A simplified block diagram of inner iteration loop

3.7.3.2 Outer Iteration Loop

The task of outer iteration loop is amplifying the scalefactor bands in such a way that the demands of the psychoacoustic model are achieved. Thus, outer iteration loop is also called distortion control loop. A simplified processing description is as follows:

1. At the beginning, no scalefactor is amplified.
2. Inner iteration loop is called.
3. The distortion causes by quantization of every scalefactor band is calculated.

4. The actual distortion is compared with the permitted distortion calculated by psychoacoustic model.

5. If it is the best result so far, store this result, and this process stops. Note that this iteration process is not always converges.

6. Amplifying the scalefactor band which has a higher distortion than the allowed.

7. If all scalefactor bands have been amplified, this process stops.

8. If the distortions of all scalefactor bands are smaller than permitted, this process stops.

9. Otherwise, the whole process will repeat.

A simplified block diagram of outer iteration loop is shown in Fig. 3.14.

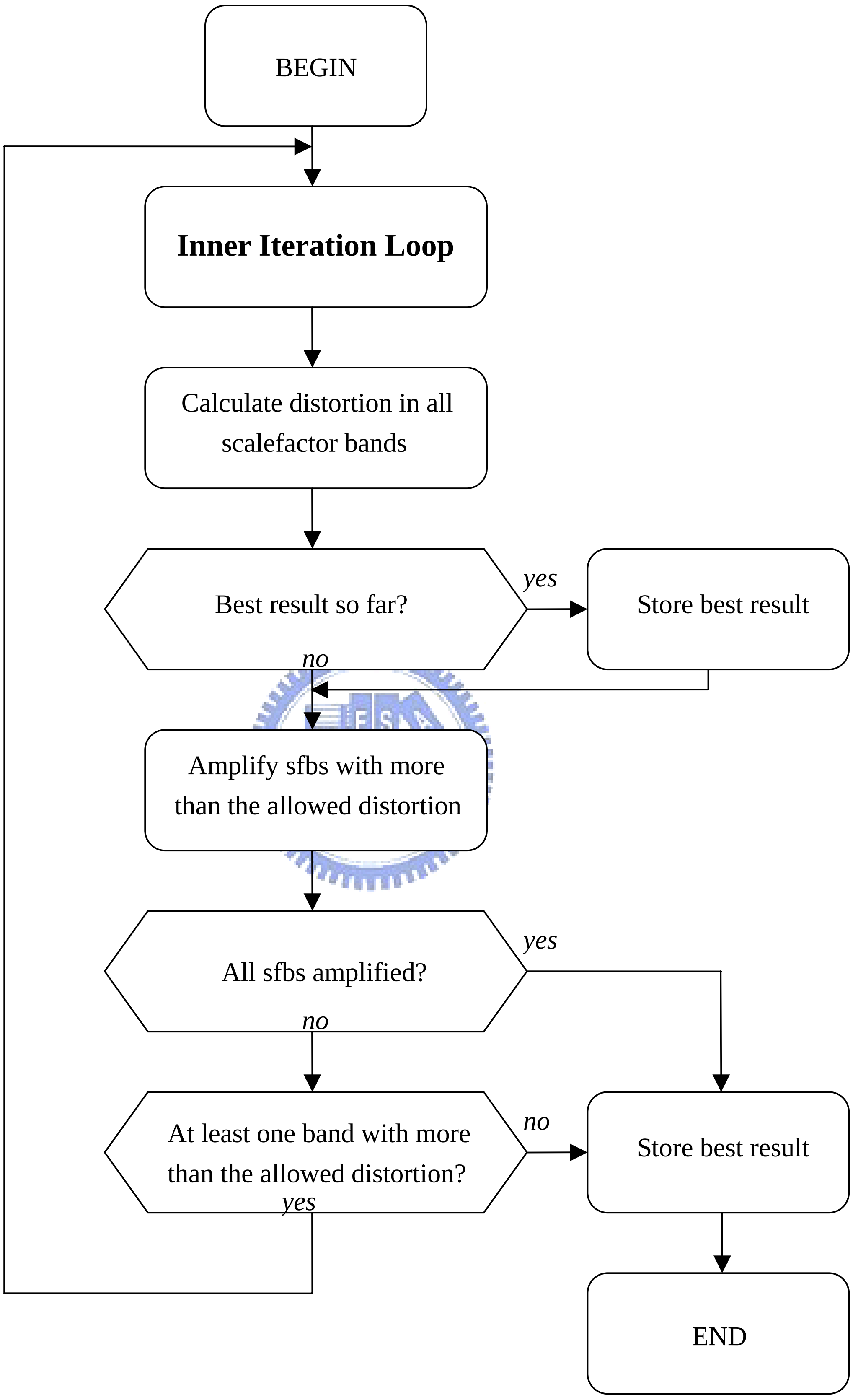

BEGIN
Inner Iteration Loop
Calculate distortion in all scalefactor bands
Best result so far?
yes
Store best result
no
Amplify sfbs with more than the allowed distortion
All sfbs amplified?
yes
no
At least one band with more than the allowed distortion?
no
Store best result
yes
END

Fig. 3. 14 A simplified block diagram of outer iteration loop

3.8 Noiseless Coding

Noiseless coding is used to further reduce the redundancy of scalefactors and the quantized spectrum. This is done by lossless packing of quantized spectral data exploiting statistical dependencies and other properties.

3.8.1 Grouping and Interleaving

As for the window sequence of type EIGHT_SHORT_SEQUENCE, there could be a possibility that some of eight short blocks are very different from the other. For example, the first three sets are nearly silent in time domain, the next two sets are actually where the onset event happens, and the final three are the decay of the event. In such cases, sets of 128 coefficients that have similar statistics are grouped together and interleaved to form a single spectrum. Fig. 3.15 shows the grouping example stated above.

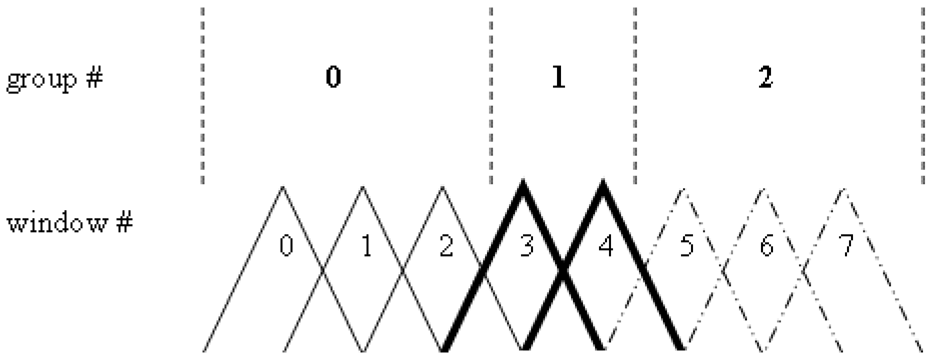


Fig. 3. 15 Example for short window grouping

To be specific, assume that before interleaving the spectral coefficients are indexed as

C[g][w][b][k]

where
g is the index of groups,
w is the index of windows within a group,
b is the index of scalefactor bands within a window,
k is the index of coefficients within a scalefactor band.

After interleaving the coefficients are indexed as

C[g][b][w][k]

This has the advantage of combining the high-frequency zero-valued coefficients (due to band-limiting) within each group. Fig. 3.16 shows the spectral order within one group before interleaving, and Fig. 3.17 shows the spectral order after interleaving.

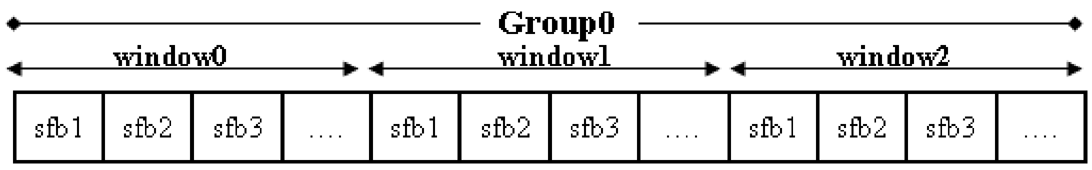


Fig. 3. 16 Spectral order within one group before interleaving

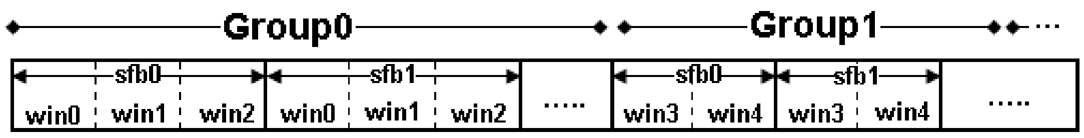


Fig. 3. 17 Spectral order after interleaving

3.8.2 Spectral Clipping

The first step of noiseless coding is a method of dynamic range compression that may be applied to the spectrum. Up to four coefficients can be coded separately as magnitudes in excess of one, with a value of ± 1 left in the quantized coefficient array to carry the sign. The "clipped" coefficients are coded as integer magnitudes and an offset from the base of the coefficient array to mark their location. This method is applied only if it results in a net savings of bits.

3.8.3 Huffman Coding

A variable-length Huffman coding is employed to compensate the nonuniform probability distribution for the levels in quantizer and to represent *n*-tuples of quantized coefficients. In the AAC system, Huffman codewords are drawn from one of 11 codebooks. The maximum absolute value of the quantized coefficients that can be represented by each Huffman codebook and the number of coefficients in each *n*-tuple for each codebook is shown in Table 3.4. There are two codebooks for each maximum absolute value, with each representing a distinct probability distribution. The best fit is always chosen.

Codebook Index	***n*-Tuple Size**	**Maximum Absolute Value**	**Signed**
0		0	
1	4	1	Yes
2	4	1	Yes
3	4	2	No
4	4	2	No
5	2	4	Yes
6	2	4	Yes
7	2	7	No
8	2	7	No
9	2	12	No

10	2	12	No
11	2	16(ESC)	No

Table 3. 5 Huffman codebooks [4]

Note that codebook 0 indicates all coefficients within that scalefactor band are zero, and codebook 11 can especially represent those who have an absolute value greater than or equal to 16, and a special *escape coding* mechanism is used to represent them. For each coefficient magnitude greater or equal to 16, an escape sequence is appended, as follows:

escape sequence = <escape_prefix><escape_separator><escape_word>

where
<*escape_prefix*> is a sequence of N binary “1’s”,
<*escape_separator*> is a binary “0”,
<*escape_word*> is an N+4 bit unsigned integer, MSB first and N is a count that is just large enough so that the magnitude of the quantized coefficient is equal to

$2^{(N+4)} + < escape\_word >$.

3.8.4 Sectioning

The noiseless coding segments the scalefactor bands into *sections*. Each section uses only one Huffman codebook, thus the number of bits needed to represent the full block is minimized.

Section is dynamic and typically varies block from block. This is done using a greedy merge algorithm by starting with the maximum possible number of sections (only one scalefactor band per section). Sections are merged if the resulting merged section needs lesser number of bits. If the sections to be merged use different Huffman codebooks, the codebook with higher index is always chosen.

3.9 Gain Control

The gain control module is employed only in the SSR profile. It consists of a polyphase quadrature filter (PQF), gain detectors, and gain modifiers. By neglecting the signals from the upper bands of PQF, this output bandwidths can be 18, 12, and 6 kHz when one, two, or three PQF outputs are ignored, respectively.

The advantage of this scalability is that the complexity can be reduced as the output bandwidth is reduced.

3.9.1 Polyphase Quadrature Filter

The PQF splits each audio channel's input signal into four frequency bands of equal width. The coefficients of each band's PQF are given by

$$h_i = \frac{1}{4}\cos\left[\frac{(2i+1)(2n+5)\pi}{16}\right]Q(n), \qquad 0 \le n \le 95, \qquad 0 \le i \le 3 \qquad (3.19)$$

Where,

$$Q(n) = Q(95-n), \qquad 48 \le n \le 95$$

The Q(n) is the filter coefficients that are standardized in [2].

3.9.2 Gain Detector

The gain detector produces gain control data including number of bands receiving gain modification, and the number of modified segments and indices indicating the location and level of gain modification for each segment. Note that the gain detector has a one-frame delay.

3.9.3 Gain Modifier

The gain modifier applies gain control to the signal in each PQF band by windowing the signals of the gain control function. Fig. 3.18 shows the block diagram of gain control module.

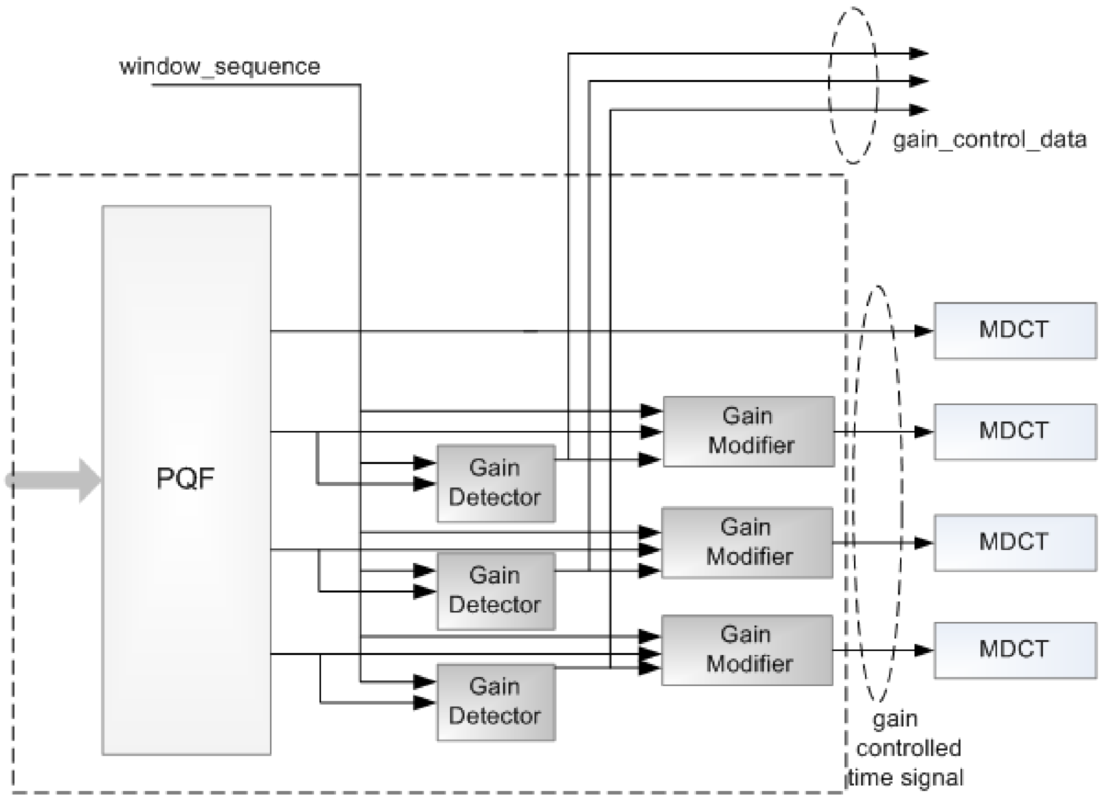


Fig. 3. 18 Block Diagram of AAC encoder gain control module [2]

3.10 Bitstream Format

There are two kinds of transport syntax that have been standardized in AAC:

- *Audio Data Interchange Format (ADIF)*: The audio bitstream contains one single header with all information necessary to control the decoder. The main application of ADIF is exchange of audio files.

ADIF	block	block	block	block	block

- *Audio Data Transport Stream (ADTS)*: The audio bitstream consists of a sequence of frames with headers similar to MPEG-1 audio frame headers. The encoded audio data of one frame is always contained between two sync words.

ADTS	block	**ADTS**	block	**ADTS**	block

There are mainly five elements in the bitstream: audio data element, data stream element (DSE), program configuration element (PCE), fill element (FIL), and terminator (TERM). Audio data element also consists of four possible elements: single channel element (SCE), channel pair element (CPE), coupling channel element (CCE), and low frequency enhancement channel element (LFE).

SCE contains coded data for a single audio channel. CPE contains data for a pair of channels, and the two channels may share common side information. CCE represents the information for intensity stereo coding. LFE gives the low frequency (under 120 Hz) audio data. PCE contains program configuration data, such as profile, sampling rate, channel information, etc. FIL is used when transporting over a constant rate channel to adjust instantaneous bitrate. DSE contains any additional data that is not part of the audio information itself. TERM indicates the end of a raw data block. Example of possible bitstreams are:

- The syntax of a single channel element (SCE)

ICS Info.	Section	Scalefactor	Pulse Data	TNS	Gain Data	Spectral Data

- Mono signal

SCE	TERM	SCE	TERM	SCE	TERM

- Stereo signal

CPE	TERM	CPE	TERM	CPE	TERM

- 5.1 channel signal

SCE	CPE	CPE	LFE	TERM	SCE	CPE	CPE	LFE	TERM

- If the bitstreams are to transmit over a constant rate channel

CPE	FIL	TERM	CPE	FIL	TERM

- If the bitstreams are to carry ancillary data and run over a constant rate channel

CPE	DSE	FIL	TERM	CPE	DSE	FIL	TERM

CHAPTER 4. MPEG-2/4 LC AAC Encoder Optimization

Comparing with main profile, low complexity (LC) profile significantly reduces memory usage and computational effort, but still maintains good quality [1]. Therefore, we choose low complexity profile to implement and this is also the most commonly used profile.

4.1 Complexity Analysis

Table 4.1 shows the computational demand of a standard AAC LC implementation from MPEG reference coder:

Module	**Percentage**
Psychoacoustic Model	22%
Filter Bank	5%
Quantization	64%
Others	9%

Table 4. 1 Distribution of resources in AAC-LC encoder [21]

The most demanding module is quantization due to the presence of nested loops. Psychoacoustic model also takes up to 22% computation effort. Besides these two critical modules, our optimization covers other modules as well.

4.2 Removal of Block Switching

Modern audio compression algorithm often adopts dynamic block switching to avoid pre-echoes (see Section 3.3.1). In general, psychoacoustic model decides whether block type changes or not depending on perceptual entropy. However, the calculation of perceptual entropy requires lots of computation effort.

A related research [22] shows that encoding without block switching didn't cause significant negative effect, TNS module in AAC system also aims at controlling pre-echo phenomenon which can compensate the lack of block switching. Under these considerations, we remove the mechanism of block switching. For this, not only the calculation of perceptual entropy, but also the complexity of block synchrony and short block related grouping and interleaving algorithm are eliminated.

4.3 Fast MDCT

AAC uses MDCT with 50% overlap in its filterbank module. However, there are lots of multiply-accumulation operation within this module, thus adopting a fast algorithm is necessary. According to [20], MDCT can be rewritten as the real part of odd-time odd-frequency discrete Fourier transform ($\mathrm{O^2DFT}$), and finally need only *N/4*-point FFT calculation:

Coefficient number k of $\mathrm{O^2DFT}$ of length N is defined as:

$$O^2DFT_{[N]_k}\{\underline{u}\} = U_k = \sum_{r=0}^{N-1} u_r e^{-j\frac{\pi}{2N}(2k+1)(2r+1)} \tag{4.1}$$

MDCT can be rewritten as the real part of $\mathrm{O^2DFT}$:

$$X(k) = \mathrm{Re}\left\{O^2DFT(k)\right\} = \mathrm{Re}\left\{\sum_{r=0}^{N-1} x(r - N/4)e^{-j\frac{\pi}{2N}(2k+1)(2r+1)}\right\} \tag{4.2}$$

[26][20] further presented a fast algorithm for calculating:

$$W = O^2DFT\{odd(x(r - N/4))\} = X(k) \tag{4.3}$$

as

$$\begin{aligned} W_{2k} &= \mathrm{Re}\{P_k\} \\ W_{N/2+2k} &= \mathrm{Im}\{P_k\} \\ W_{N-1-2k} &= -\mathrm{Re}\{P_k\} \\ W_{N/2-1-2k} &= -\mathrm{Im}\{P_k\} \end{aligned} \tag{4.4}$$

Where

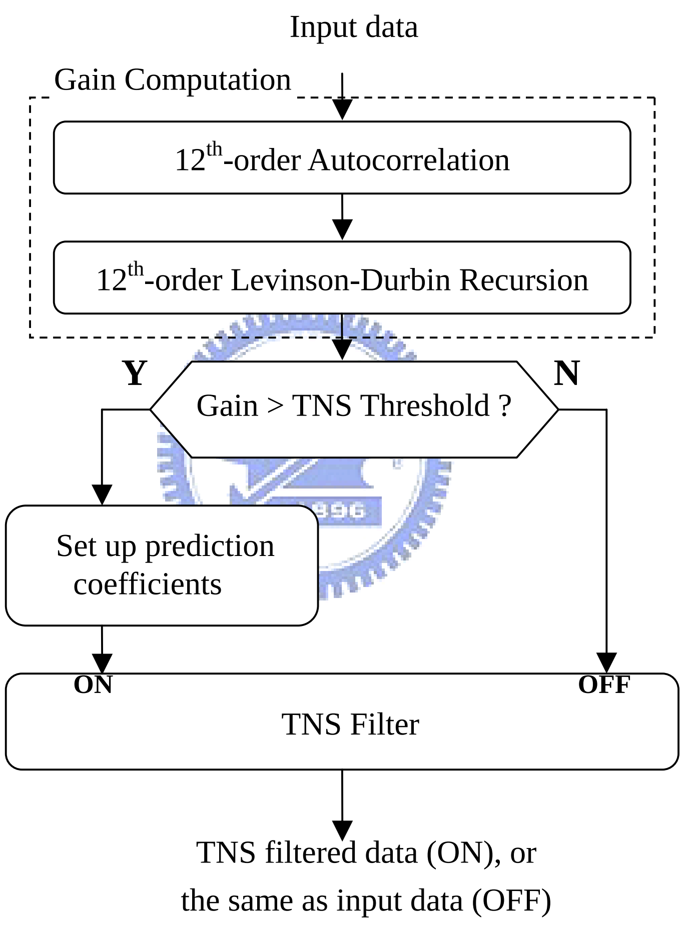
$$\begin{aligned} P_k &= U_{2k} - jU_{N/2+2k} \\ &= 2\sum_{r=0}^{N/4-1}(u_{2r} - ju_{N/2+2r})e^{-j\frac{2\pi}{N}\left(2k+\frac{1}{2}\right)\left(2r+\frac{1}{2}\right)} \\ &= 2e^{-j\frac{2\pi}{N}\left(k+\frac{1}{8}\right)}\underbrace{\sum_{r=0}^{N/4-1}\left[(u_r' e^{-j\frac{2\pi}{N}\left(r+\frac{1}{8}\right)})e^{-j\frac{2\pi}{N/4}rk}\right]}_{N/4 \quad point \quad FFT} \end{aligned} \tag{4.5}$$

$$* u_r' = u_r - ju_{N/2+2r} \tag{4.6}$$

Through this process introduced in [26][20], only *even* indices are calculated because of the basic symmetries of the O^{2}DFT. Another saving of 50% computation is calculating two values simultaneously, as shown in (4.5). Finally, only one *N/4* point FFT is needed with some overhead of pre and post rotation of the sample point.

4.4 Simplified TNS

In general, TNS method is the linear prediction performed in frequency domain. We use the popular Levinson-Durbin recursive procedure to achieve linear predictive coding (LPC). The TNS prediction order of LC profile is designed to be 12 as described in [2]. Fig. 4.1 shows the basic TNS implementation flow:



Input data
Gain Computation
12th-order Autocorrelation
12th-order Levinson-Durbin Recursion
Y
N
Gain > TNS Threshold ?
Set up prediction coefficients
ON
OFF
TNS Filter
TNS filtered data (ON), or
the same as input data (OFF)

Fig. 4. 1 Original TNS implementation flow

We are curious about how often TNS is active, therefore, the percentage of whether TNS is finally applied is statistically measured. Table 4.2 shows the result.

Test audio sample	Active percentage
Violoncello[23]	5.5%
Quartet[23]	19.9%
Soprano[23]	6.7%
Radio[24]	4.3%

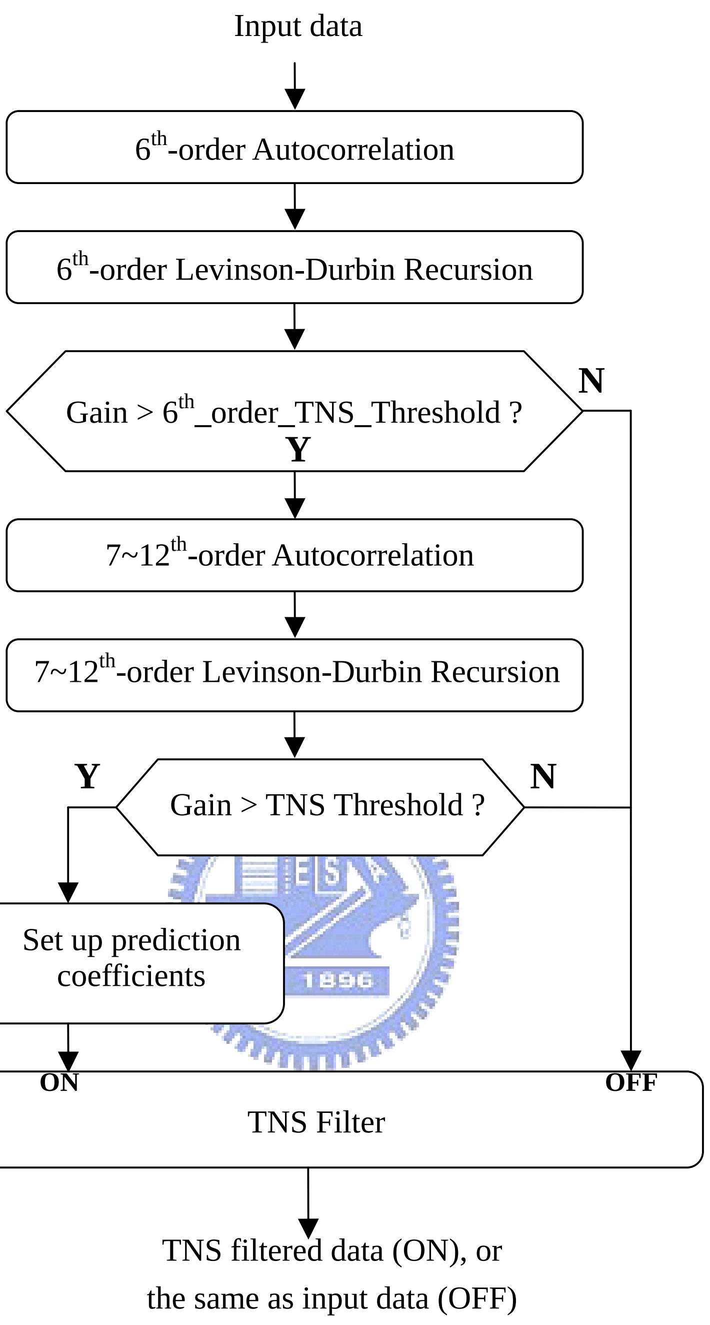
Table 4. 2 the percentage of TNS being active

We found that the active percentage is pretty low, however, the prediction gain has to be computed every time no matter TNS is finally on or off. Generally, the procedure to obtain prediction gain (including autocorrelation and Levinson-Durbin recursion blocks) requires as much computing effort as TNS filter, see Table 4.3. If further taking the active frequency into account, prediction gain computation actually contributes the most complexity in TNS module. Thus, if we can decide whether TNS is on or off earlier, that is, before 12th-order LPC has been completed, the complexity will be reduced.

Block	Percentage
Gain Computation	49.5%
TNS Filter	49.5%
Others	1%

Table 4. 3 Distribution of resources in TNS module (TNS is finally "ON" case)

To eliminate this over computation, we propose using 6th order prediction gain to determine if LPC procedure will go on. Fig. 4.2 shows the modified TNS implementation flow.



Input data
6th-order Autocorrelation
6th-order Levinson-Durbin Recursion
Gain > 6th_order_TNS_Threshold ?
N
Y
7~12th-order Autocorrelation
7~12th-order Levinson-Durbin Recursion
Y
Gain > TNS Threshold ?
N
Set up prediction coefficients
ON
OFF
TNS Filter
TNS filtered data (ON), or
the same as input data (OFF)

Fig. 4. 2 Modified TNS implementation flow

The matching percentage of this early-deciding mechanism is about 90% in average. Table 4.4 shows some results.

Test audio sample	**Matched**(1)	**Missed**(2)	**Over**(3)
Violoncello	90.0%	1.5%	8.5%
Quartet	87.9%	3.0%	9.1%
Soprano	96.3%	0.05%	3.6%
Radio	95.9%	0.4%	3.7%

Table 4. 4 Comparisons between original and modified TNS

(1) Matched – Both 6th-and 12th-order prediction gain made the same decision.
(2) Missed – The TNS filter should be turned on in original case, however, it is set to be off by mistake in 6th-order LPC stage.
(3) Over – Though TNS filter should finally be turned off, 6th-order LPC wrongly passes the calculation to 12th order. Note this part *doesn't* really cause the wrong implementation, since the TNS filter will still be turned off in 12th-order LPC stage.

Only the "*missed*" portion indicates how much that proposed modified TNS method leads to a different result as comparing to original TNS mechanism. Thus, from Table 4.4, we can say this modified TNS method has very much the same effect as the original TNS method, but the computation effort is a lot reduced in the same time.

The complexity of LPC by Levinson-Durbin approach is generally known as O(N^2), where "N" denotes the prediction order. Therefore, 6th-order LPC has only one-fourth of complexity as comparing to 12th-order LPC. The computation reduction in this modified TNS method is achieved only in TNS-inactive frame. From Table 4.2, roughly 91% frames are originally TNS-inactive. And from Table 4.4, the "*over*" portion should be excluded, almost 84% of total frames are benefited from this modified TNS method. In this way, the computation effort within TNS module can be reduced to about $84\% \times \frac{1}{4} + 2\%_{missed} \times \frac{1}{4} + 14\% \times 1 = 35.5\%$ of the original TNS method.

4.5 Simplified M/S Stereo Coding

Although the heart of mid/side stereo coding is simply calculating the sum and difference between two channels, the process to determine whether mid/side stereo coding should be applied or not requires much more effort. Bit-consuming of both normal modes (L/R) and M/S mode are calculated and compared. The mode with fewer bits requirement will be chosen. And the decision is also made scalefactor band by scalefactor band.

Usually, except the multilingual or 1/+1 case, the data similarity between channel pair is quite high. We wonder if the decision can be made more easily. Therefore, we try to investigate how often the encoder decides switching to M/S mode.

Test audio sample	**M/S mode percentage**
Elliott[24]	88.16%
Quartet	88.84%
Devic[24]	95.03%
Sandee[24]	88.34%
Soprano	89.46%

Table 4. 5 The percentage that encoder switching to M/S mode

We found the encoder decides to switch to M/S mode very often, at least 80% which is shown in Table 4.5, and nearly 90% scalefactor bands of a single frame will be switched to M/S mode. So we think the decision can be made more easily, and shouldn't cause too much overhead. Such as using the energy of the entire frame as what MPEG-1 layer III does. Also, by observing that the energy ratio of channel pair of the scalefactor band that switches to M/S mode is often around 1~2, as you can see in Fig. 4.3, whereas the energy ratio of the scalefactor band that remains in L/R mode is usually larger, as shown in Fig. 4.4.

Also, since a frame with lower energy is less benefited from M/S stereo coding, we should save computation for these frames, which usually occurs in the beginning and the end of the song, or the near silence in the middle of the song.

Thus, the proposed implementation is setting a first *threshold_1*, when the average energy of channel pair is below this threshold, M/S stereo mode will be enabled to the entire frame temporarily. And then, further takes the energy ratio into consideration, a second *threshold_2* is set, when the energy ratio of the entire frame of channel pair exceeds this threshold, this frame will remain in L/R mode still, even its energy is quite large that has been set M/S mode enabled in the first stage. The proposed decision flowchart can be found in Fig. 4.5.

Because the energy of entire frame has been pre-calculated for the use in quantizaton module, there are only one energy ratio calculation and few comparisons need to be computed to make the decision.

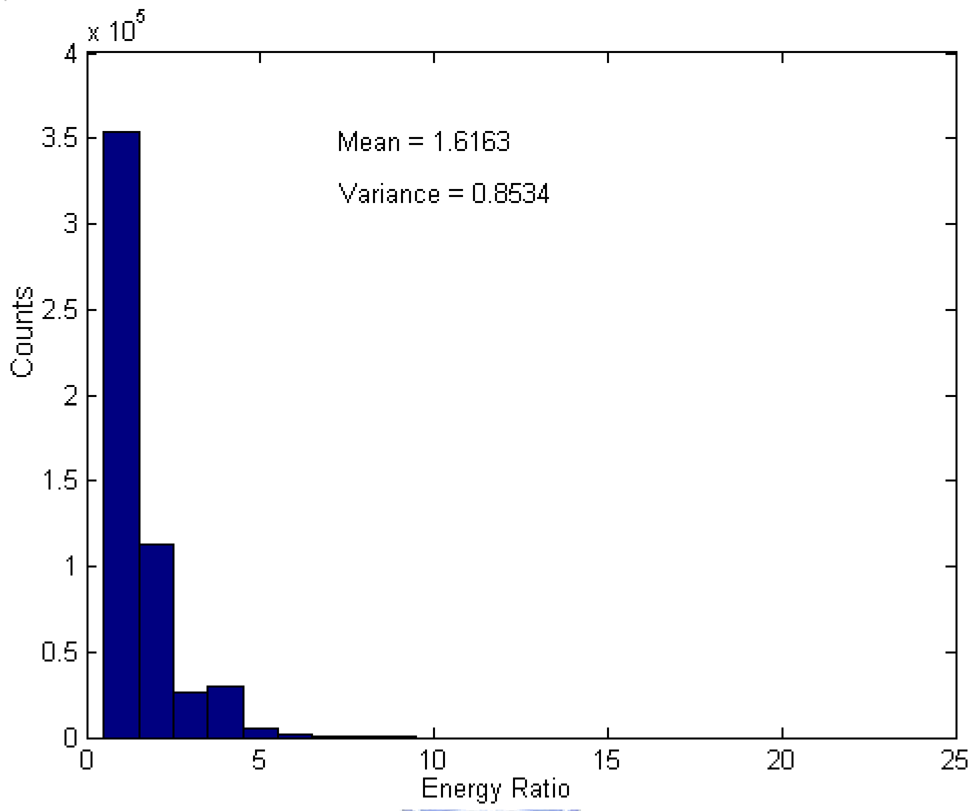


Fig. 4. 3 Energy ratio of channel pair of the scalefactor band switches to M/S mode

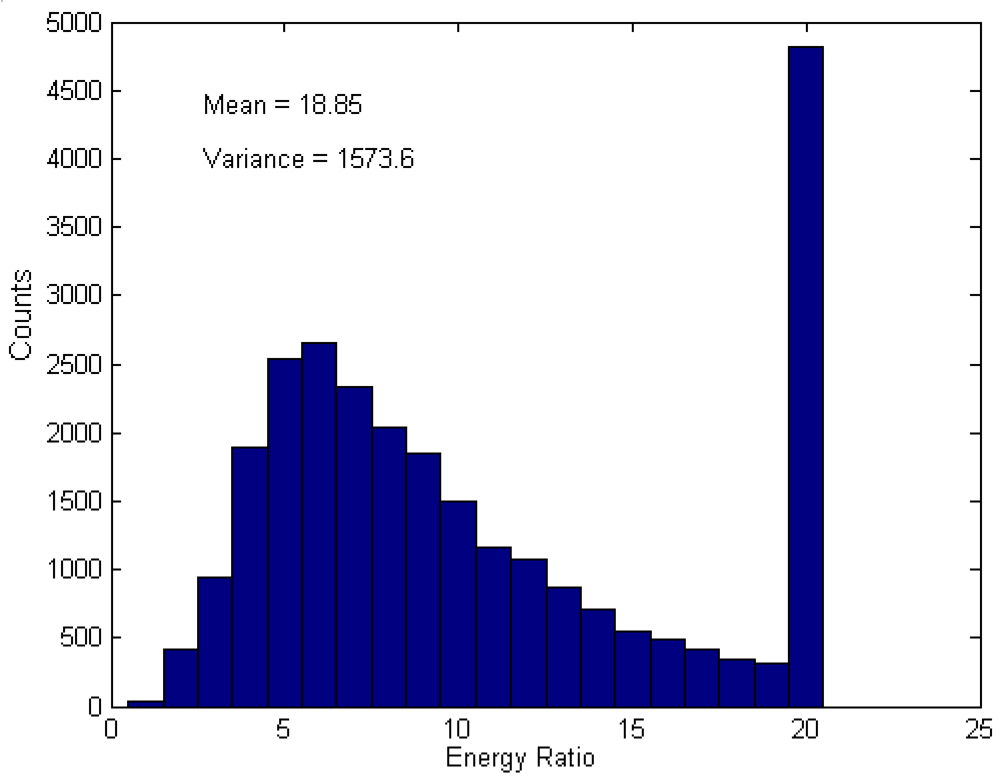
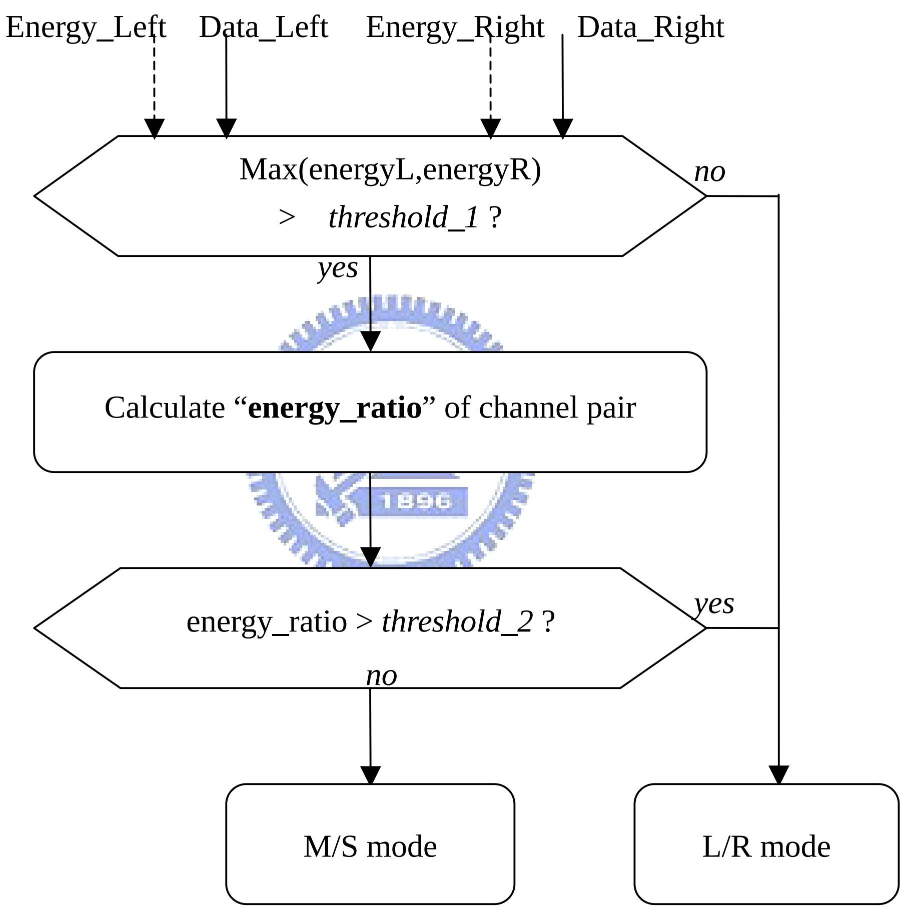


Fig. 4. 4 Energy ratio of channel pair of the scalefactor band remains in L/R mode

The energy ratio is defined as:

$$energy\quad ratio = \begin{cases} \dfrac{Energy\_Left}{Energy\_Right}, & Energy\_Left \geq Energy\_Right \\ \dfrac{Energy\_Right}{Energy\_Left}, & Energy\_Right > Energy\_Left \end{cases} \tag{4.7}$$

Fig. 4.5 blow shows the flowchart of the proposed M/S stereo switching decision scheme.



Energy_Left
Data_Left
Energy_Right
Data_Right
Max(energyL,energyR) > threshold_1 ?
no
yes
Calculate "energy_ratio" of channel pair
energy_ratio > threshold_2 ?
yes
no
M/S mode
L/R mode

Fig. 4. 5 The flowchart of the proposed M/S stereo decision scheme

4.6 Quantization Optimization

From Table 4.1, we find that quantization consumes most computation power, and this is primarily due to the nested inner and outer loops. We based on FAAC's[15] implementation of quantization and further do some modifications.

Our encoder performs an average bit rate (ABR) encoding, in this way, the inner loop can be simplified to merely Huffman coding without bit rate control. The nested loops now have only outer loop left. The relationship between ABR encoding and single loop quantization can be shown as follows:

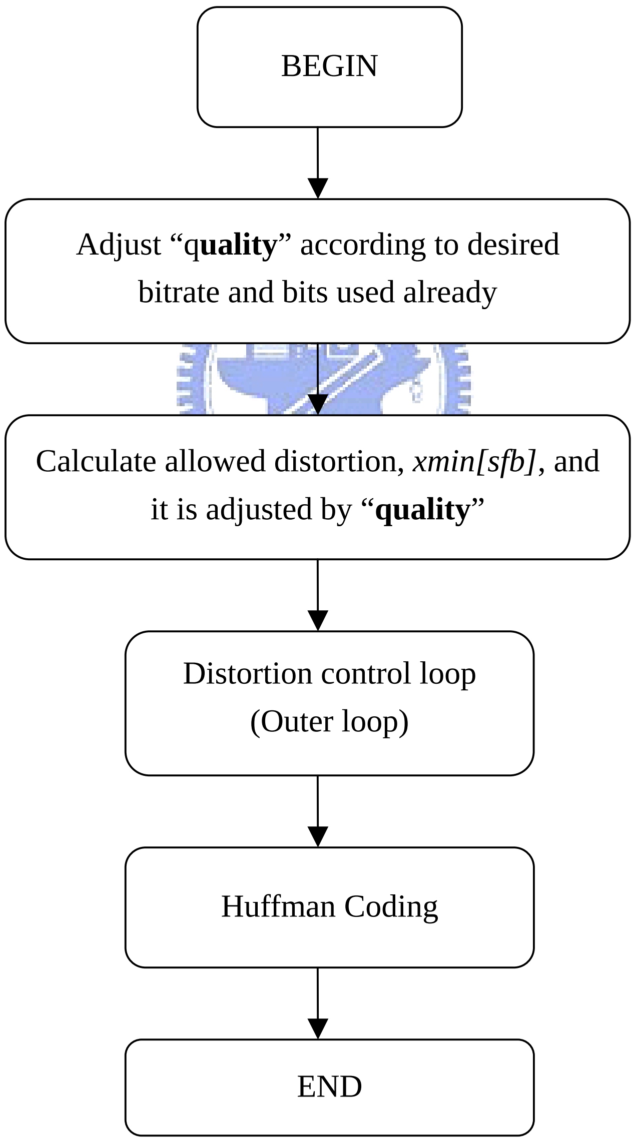

BEGIN
Adjust "**quality**" according to desired bitrate and bits used already
Calculate allowed distortion, *xmin[sfb]*, and it is adjusted by "**quality**"
Distortion control loop (Outer loop)
Huffman Coding
END

Fig. 4. 6 The relationship between ABR encoding and single loop quantization

4.6.1 Scalefactor Prediction

Now our encoder performs single loop quantization, thus the complexity primarily lies in the distortion control loop. The distortion control is achieved by adjusting scalefactors of scalefactor bands so that a higher SNR will be obtained. Due to the usually quasi-stationary property of audio and speech data, it is intuitively to think of using previous frame's scalefactor as the initialization of current frame's scalefactor. However, to prevent special cases, some thresholds should be set to make sure when special case occurs, the original initialization will be used instead. If an unreasonable predicting scalefactor is adopted, more loops may be required to achieve distortion control. Table 4.6 shows some results of the average loop count of quantizer before and after scalefactor prediction.

Test audio sample	**Original**	**Modified**
Violoncello	9.22	6.55
Quartet	8.61	6.35
Soprano	8.45	6.80
Radio	7.96	6.57
Bass[23]	8.27	6.45

Table 4. 6 Average iteration loop counts of quantizer

From experimental results, 20% of iteration loop counts can be reduced in average.

4.6.2 Simplified QuantizeBand()

By statistical measure, we found QuantizeBand()[15] is the most computation demanding function in quantization module. Over 50% of quantization computing effort spent in this function. Thus, we should try to do some optimization here.

What QuantizeBand() actually dose is exploiting Takehiro IEEE 754 Hack, a fast method turning float digits into integer digits, to perform quantization. However, the vital drawback of this method is that a large amount of memory is needed for audio fine-tune. Here, a large table of 8192 entries is required. By observation, we find the relationship between the input and output of QuantizeBand() function is quite simple and could be much easier to realize. Only a five-entry lookup table and one linear approximation segment are needed. In average, most inputs of QuantizeBand() are distributed within the range $x < 4.51$, see Table 4.7.

Iutput Range	**Percentage**
$x < 4.51$	99.10%
$x \geqq 4.51$	0.90%

Table 4. 7 The distribution of input range of QuantizeBand()

Thus, applying a small lookup table can largely save computation power. Not only the table size is significantly reduced, the approximating error is also quite small, as you can see in Fig. 4.7.

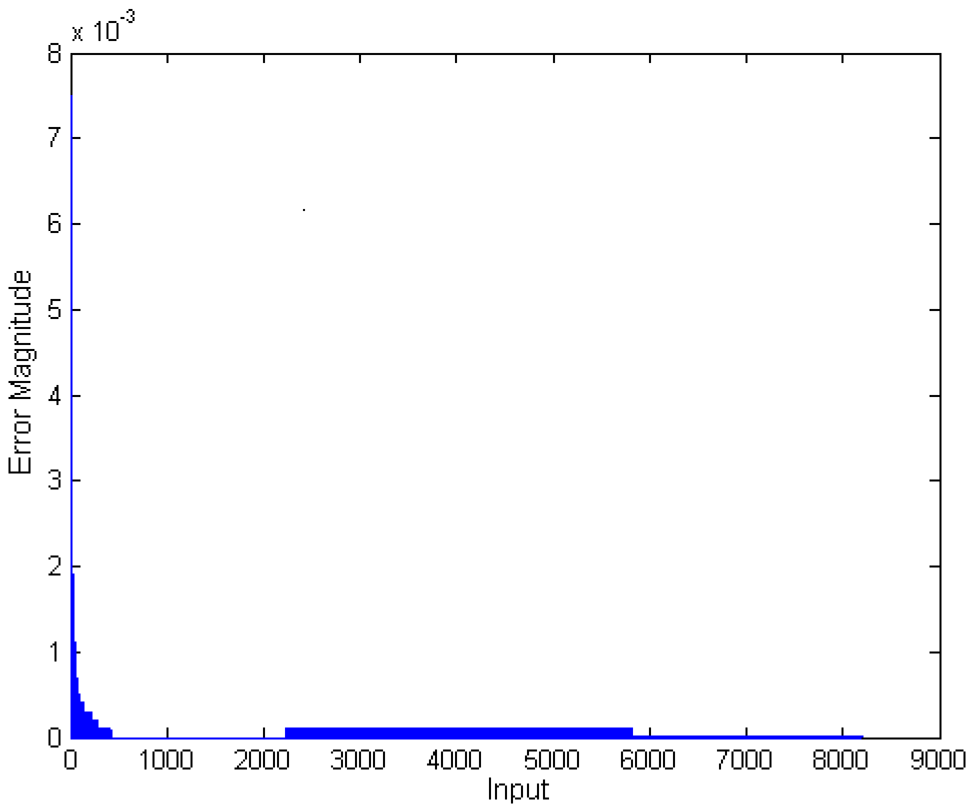


Fig. 4. 7 The error magnitude of the approximating QuantizeBand() function

4.7 Math Functions Approximation

There are lots of complicated mathematical calculations in AAC encoding system which are originally solved by *math.h* of C Library. However, it is unfeasible when implementing on some power-limited devices. Thus, a simplified approach to these math functions is necessary. Some of them may use simply linear approximation, and some of them can play little tricks to approximate

4.7.1 TNS

When TNS is applied, the reflection coefficients are therefore needed to be transmitted. However, if we transmit reflection coefficients directly, it would cost too many bits which is undesirable. Thus, these reflection coefficients will be quantized, and only *index* will be transmitted. As we know, the reflection coefficients of LPC always lie within the interval +1 ~ -1. The quantization is done by a sin^{-1} function.

$$index = (\text{int})\sin^{-1}(k_n) \times iqfac \tag{4.8}$$

where, $$iqfac = \frac{[1 << (coeff\,\mathrm{Re}s - 1)]}{\pi/2} \tag{4.9}$$

From (4.8) and (4.9), *index* can be rewritten as:

$$index = (\text{int})\left[\frac{\sin^{-1}(k_n)}{\pi/2} \times 2^{(coeff\,\mathrm{Re}s-1)}\right] \tag{4.10}$$

$$invert\_quantized\_k_n = \sin\left(\frac{index}{2^{(coeff\,\mathrm{Re}s-1)}} \times \frac{\pi}{2}\right) \tag{4.11}$$

where,
coeffRes = reflection coefficient resolution,
k_n = reflection coefficient.

For simplicity and generality, some rounding factor in (4.7) and (4.8) are eliminated.

When the coefficient resolution is defined as a constant, this quantization can be done by using a $2^{coeffRes}$-entries lookup table.

4.7.2 Quantization

Most of those unusual math functions appear in quantization module. And they especially need a simplified implementation.

- **$2^{-0.25x}$**

In the function BalanceEnergy()[15], $2^{-0.25x}$ is used. Since x is an integer, it makes this function much easier to approximate.

First, x can be decomposed into ($4a + b$), thus, $2^{-0.25x}$ can be rewritten as

$$2^{-0.25x} = 2^{a} \times 2^{\frac{1}{4}b} \tag{4.12}$$

where a and b are both integers, especially, b is always less than 4.

Therefore, 2^{a} can be implemented by a simple shift. And since $\frac{1}{4}b$ can only be four numbers: 0.0, 0.25, 0.5, and 0.75. If three constants, $2^{0.25}$, $2^{0.5}$, and $2^{0.75}$, are predefined, and $2^{-0.25x}$ can thus be easily realized by a shift and a multiplication. In this way, only small computing effort and small extra memory requirement are needed.

Now you may wonder how we obtain a and $\frac{1}{4}b$? First, a = (x >> 2) which is quite simple. To obtain $\frac{1}{4}b$, let $x = (x_4\, x_3\, x_2\, x_1\, x_0)_2$. Apparently, b is the first LSB and the second LSB of x:

$x = (x_4\, x_3\, x_2\, x_1\, x_0)_2$	$\frac{1}{4}b$
$(x_4\, x_3\, x_2\, 0\, 0)_2$	$(0.00)_{10}$
$(x_4\, x_3\, x_2\, 0\, 1)_2$	$(0.25)_{10}$
$(x_4\, x_3\, x_2\, 10)_2$	$(0.50)_{10}$
$(x_4\, x_3\, x_2\, 11)_2$	$(0.75)_{10}$

Table 4. 8 The relationship between x and $\frac{1}{4}b$

Therefore, $2^{\frac{1}{4}b}$ can be easily detected by (x & `0x03`).

▪ *ln*(*x*)

Natural logarithm is widely used in quantization module. This simplified approach is basically based on the linear approximation. Thanks to the simple division and multiplication identities of logarithm, the range needed to be *actually* approximated can be narrowed. The "*actually*" mentioned here, will be explained later.

To do the linear approximation, we must know what range the input, we say x here, of logarithm is distributed. It is statistically measured that the range is roughly from 10^{-6} to 500. We first perform the linear approximation to the range of $0.5 < x < 5.0$ which is divided into 9 segments. And this range is what we mean "*actually-approximated*" range. The remaining range exploits the division and multiplication properties of logarithm:

$$\log(a \times b) = \log a + \log b \tag{4.13}$$

$$\log(a/b) = \log a - \log b \tag{4.14}$$

They are not approximated directly, but consume an extra multiplication and an addition, and then take the already-build linear approximation of the range $0.5 < x < 5.0$ as a lookup table.

For example, $x = 0.0452$,

$$\ln(x) = L(x \times 500) + (-\ln 500) \tag{4.15}$$

Another example, x = 76.42,

$$\ln(x) = L(x \times 0.01) + \ln 100 \tag{4.16}$$

Here, $L(.\ )$ denotes the already-build linear approximation of the range $0.5 < x < 5.0$.

Though some extra works are needed, the memory is saved and approximation error is reduced. This approach has the same effect of dividing the ranges $0.1 < x < 0.5$, $10^{-2} < x < 10^{-1}$,…., $10^{-6} < x < 10^{-5}$, $5.0 < x < 50.0$, and $50.0 < x < 500.0$ into 9 segments separately, and approximating accordingly. This same-effect dividing is quite reasonable. As shown in Fig. 4.8, we all know the curve of logarithm gets smoother when x increases, whereas the curve of logarithm becomes sharper as x decreases. This approach is just well fitted to this characteristic of logarithm curve, which uses more segments to approximate smaller x, and uses fewer segments to approximate larger x.

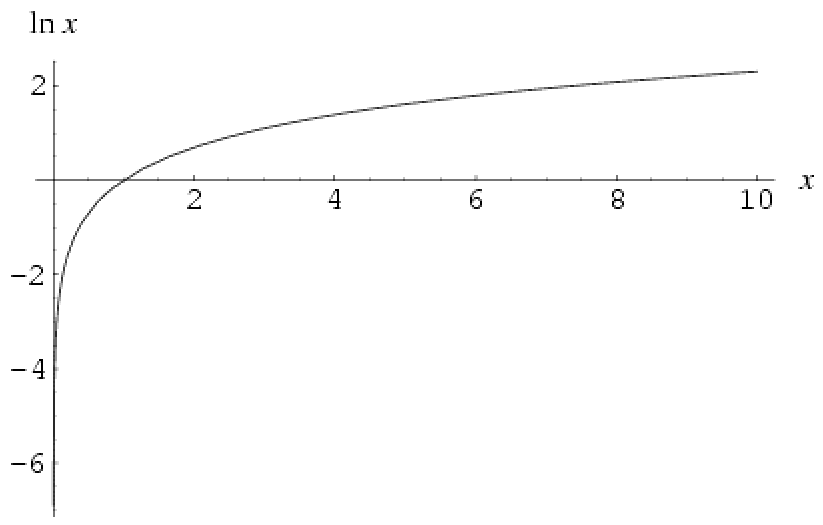


Fig. 4. 8 The curve of nature logarithm

- $tmp^{\frac{0.1*(total\_scalefactorbands-currnet\_scalefactorband)}{total\_scalefactorbands}+0.3}$

This calculation appears in the function CalcAllowedDist()[15] of quantization module. This calculation is quite annoying since both of *tmp* and its exponent are variables. The proposed approach exploits the simple properties of logarithm again. For simplicity, *total_scalefactorbands* and *current_scalefactorband* will be abbreviated as *total_sfbs* and *current_sfb* separately.

First, the exponent can be rewritten as

$$\frac{0.1\times(total\_sfbs-currnet\_sfb)}{total\_sfbs}+0.3=0.4-(\frac{current\_sfb}{total\_sfbs}) \tag{4.17}$$

Thus, to obtain $\left(\frac{current\_sfb}{total\_sfbs}\right)$ only one division is needed in first scalefactor band, the remaining scalefactor bands can use addition instead.

Now let $C=tmp^{0.4-\frac{currnet\_sfb}{total\_sfbs}}$, and taking logarithm of *C* with base 2:

$$B=\log_2 C=(0.4-\frac{current\_sfb}{total\_sfbs})\times\log_2(tmp) \tag{4.18}$$

The reason why we choose logarithm with base 2 is that this can simplify the implementation, and you will see later.

Thus,

$$C=2^B=2^{(0.4-\frac{current\_sfb}{total\_sfbs})\times\log_2 tmp}=tmp^{0.4-(\frac{current\_sfb}{total\_sfb})} \tag{4.19}$$

To solve this equation, there are two functions have to be implemented: 2^x and

$\log_2(x)$. Fortunately, these two functions are familiar, we've implemented similar functions before. First, $\log_2(x)$ can be realized by *ln*(x) linear approximation table as discussed previously, using the property of logarithm which can be used to change from one logarithm base to another:

$$\log_b(x) = \frac{\log_a(x)}{\log_a(b)} \tag{4.20}$$

Therefore, $\log_2(x)$ can be obtained by using *ln*(x):

$$\log_2(x) = \frac{\ln(x)}{\ln(2)} = \frac{1}{\ln(2)} \times \ln(x) \tag{4.21}$$

Since *ln*(x) approximation has been build already, there is no extra memory cost here.

Though we've implemented $2^{-0.25x}$ before, the situation here, 2^x, is a little bit different. Because x here is a floating-point number, it's impossible to predefine constants as what we do when implementing $2^{-0.25x}$. Howerver, we still decompose x into integer (I) part and mantissa (F) part, then applying linear approximations to implement 2^F. Fortunately, the range of F is quite small, the memory cost here won't be severe.

$$2^x = 2^I \times 2^F = (1 << I) \times 2^F \tag{4.22}$$

where $0.0 < F < 1.0$.

To estimate the performance of *ln*(x) approximation, we first assumed *uniform* distribution. The average SNR of the range $0.5 < x < 5.0$ is about 45.17 dB. The average SNR of the entire range, i.e. $10^{-6} < x < 500$, rises to 59.76 dB. And then, we further tested with varies audio samples, the resulting average SNR is 63.56 dB.

CHAPTER 5. Implementation on a StrongARM Processor

This work is primarily done in the software part. First, most speedup and memory reduction work is done in floating-point C code, and then convert it to fixed-point C code, since our target processor, Intel® StrongARM SA – 1110, is a 32-bit fixed-point processor. Some fixed-point specific modifications are also done in this stage. Further, coding style is tuned according to StrongARM implementation behavior. Finally, execution file is generated by ARM C compiler and then porting to our implementing platform, Advantech PCM-7130 SBC (single board computer).

PCM-7130 based on a 32-bit microprocessor, Intel® StrongARM SA-1110, supports various kinds of peripherals such as USB, CF, Ethernet, and so on. For more detailed description, you can find in Appendix A and [35][36].

5.1 Implementation Flow

The implementation flow can be shown as Fig. 5.1:

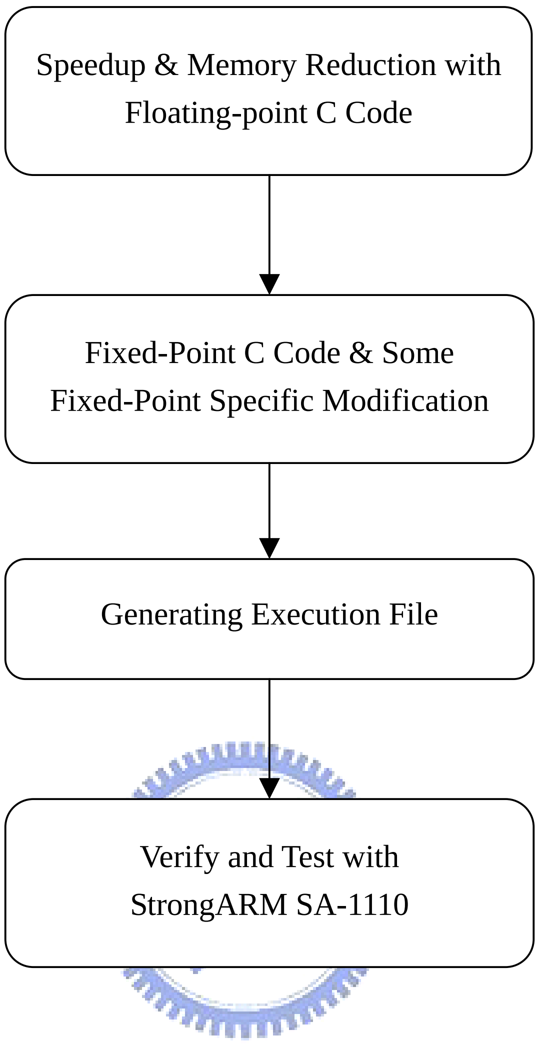

Speedup & Memory Reduction with Floating-point C Code
Fixed-Point C Code & Some Fixed-Point Specific Modification
Generating Execution File
Verify and Test with StrongARM SA-1110

Fig. 5. 1 Implementation Flow

5.2 Fixed-point C Code Implementation

Basically, StrongARM SA-1110 performs 32-bit arithmetic. To maintain more accuracy of processing data, some functions may employ 64-bit calculation, including FFT implementation of MDCT, average energy calculation of quantization module, and autocorrelation calculation of TNS module.

Especially in MDCT module, the dynamic range of processing data convert from $\pm 2^{15}$ to $\pm 2^{25}$, first explicit error may be caused here. And this early-caused error would diverge in the later approximating implementation. However, using 64-bit calculation requires large computation effort. It's impossible to perform 64-bit arithmetic through out the entire encoder. Thus, we choose to give the early module a finer error control.

Since almost 80% of transformed data are still within the range of $\pm 2^{15}$, if performing 64-bit calculation to the whole FFT process, it would cause too much overhead. Therefore, a dynamic precision calculation is applied finally. The implementation flow is shown as Fig. 5.2:

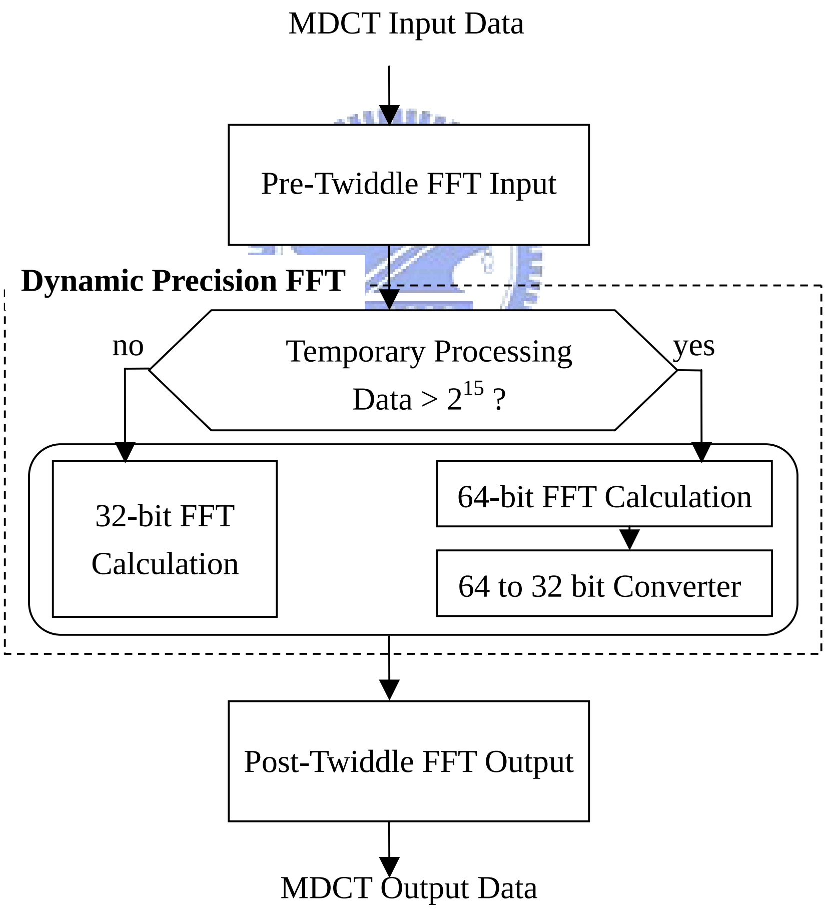


Fig. 5. 2 Dynamic precision FFT calculation in MDCT module

In order to compensate the distortion mainly caused by the conversion from floating-point source code to approximating fixed-point source code, a bandwidth control of input signal is employed [22]. Subjective tests also revealed people prefer the sound with a limited bandwidth to the sound with full bandwidth but with unmasked distortion [31].

It is in fact a low pass filter which will be applied after MDCT module. Therefore, the implementation is quite easy:

$$L(x_f(i)) = \begin{cases} x_f(i) & ,if \quad i \le n\mathrm{int}\left(\dfrac{f_c}{f_s} \times 1024\right) \\ 0 & ,if \quad i > n\mathrm{int}\left(\dfrac{f_c}{f_s} \times 1024\right) \end{cases} \tag{5.1}$$

where,

$L(.\ )$ indicates the Bandwidth control function,

$x_f(i)$ represents the ith frequency line of fth frame,

f_c represents the cut-off frequency,

f_s represents the sampling rate.

And since this modified encoding algorithm adapts long block only, the multiplicator is set 1024 as a constant.

With this bandwidth control tool, thus more bits can be allocated in the low frequency bands, and then the quality can be improved [31]. Also, since the number of nonzero data decreases, the encoding speed is therefore increased. Some cutoff frequency settings of corresponding bitrate are listed in Table 5.1. It is referenced and modified from FAAC's suggestion for VBR settings and approximating bitrate.

Bitrate (kbps/channel)	Cutoff frequency (Hz)
192 kbps	20800
160 kbps	19600
128 kbps	17200
112 kbps	16000
96 kbps	14800
80 kbps	12400
64 kbps	10000

Table 5. 1 Some desired bitrate and corresponding cut-off frequency of bandwidth control module

5.3 Modify Coding Style

Finally, the C code is further modified according to the target processor, StrongARM. Such as optimizing loop termination, simplified Boolean expressions of range checking, loop unrolling, and being caution of choosing local variable types, etc. For more details, you can find in [28][29].

CHAPTER 6. Implementation of Data Embedded Method

There are many watermark techniques in terms of their application areas and purposes. The technology of data embedding is, in fact, one kind of watermarking. With data embedding, this highly compressed audio data is more attractive to consumers since it provides extra services while listening to music.

6.1 The Properties of Data Embedded Method

This data embedded method is mainly derived from [32], with modifications accommodate it to AAC file structure. This method is specifically available for audio data due to its exploiting the psychoacoustic characteristic of human hearing. The classification of this data embedded method is summarized in Table 6.1.

Classification	Contents
Perceptivity	Invisible
Watermark Type	Public Watermarking
Inserting File Type	Any Type
Robustness	Fragile
File Size After Inserting	Unchanged
Processing Method:Frequency Domain	Spread Spectrum of High Frequency

Table 6. 1 Classification of the watermarking technique in this thesis

- **Perceptivity of watermark: Invisible**

The embedded data must be invisible, since the inserted media type is audio. And the quality of audio file after data embedding should be unaffected, or at least imperceptible.

- **Watermark type: Public watermark**

This data embedded method belongs to public watermark which doesn't aim at protecting file but providing additional services such as some related information.

- **Inserting file type: Any type**

No matter what file type inserted, the data embedded algorithm remains the same. Since the data-embedding extractor simply reads the binary data stream. There's also a header in the temporary file of data embedding process, *package file,* which contains the synchronization bits, the file size and the file type of each embedded files, and of course the file data stream. The data embedded decoder can therefore reconstruct embedded files according to the information in the header of package file.

- **Robustness of watermark: Fragile**

As stated previously, this data embedded method doesn't mean to provide protection rather than giving additional services. The embedding data is easily destroyed when these data embedded audio files undergo re-compression or some post-processing, such as filtering, reverberation and equalization, etc.

- **File size after inserting: Unchanged**

It is the most attractive feature of the proposed data embedded method. That is, under the same encoding settings, the resulting AAC file sizes with and without data embedding are the same! This data embedding doesn't cause the resulting file size to increase. Further, the data embedded AAC file is also compatible to those decoders without proposed data embedded algorithm. Thus, these data embedded AAC files can be play safely with general AAC decoders.

6.2 Implementation of Data Embedded Encoder

Fig. 6.1 shows the basic structure of data embedded AAC encoder. The data embedded algorithm is bundle within the AAC encoding process rather than a post- or pre- processing of AAC system.

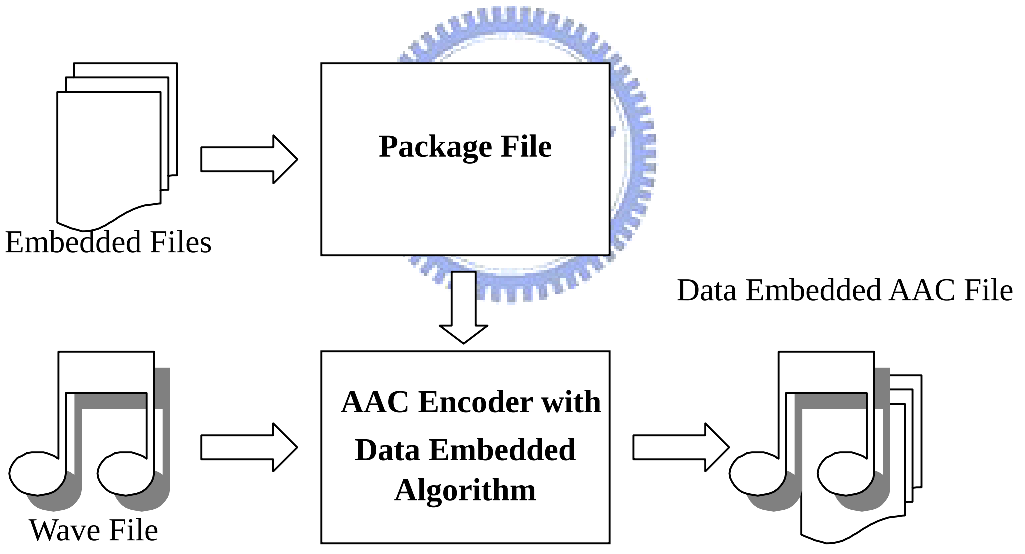


Fig. 6. 1 The structure of data embedded AAC encoder

First, the files you want to insert will be pack into a so called package file. Meanwhile, the file size and file type of each inserting files are also store in the header of package file. Additional synchronization bits are used to recognize the start of a new file. The file structure of package file is shown in Fig. 6.2.

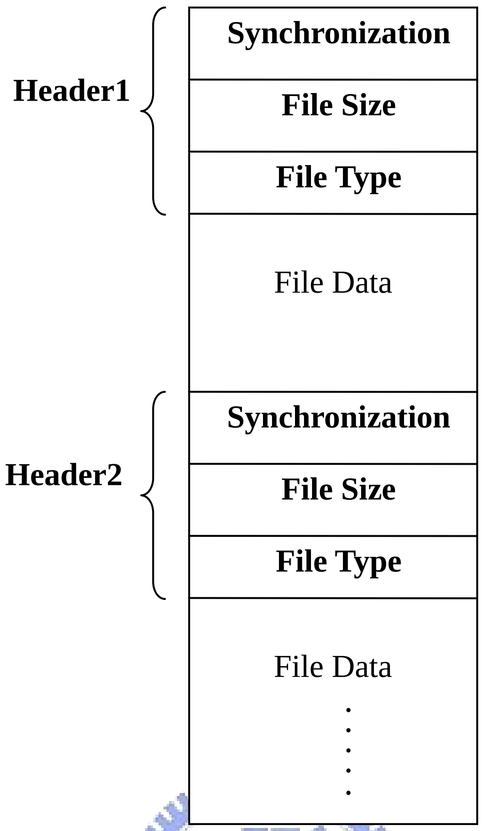


Fig. 6. 2 The structure of package file [32]

After the generation of package file, the AAC encoding routine then begins. The package file is extracted bit by bit and embedded into quantized data before entering the noiseless coding module. The later implementation is exactly the same as the original AAC encoder. The package file structure is exactly the same design as [32], if you are interested, please see [32] or Appendix B.

6.2.1 Embedding Data into High Frequency Range

As we've discussed in Chapter 2 and Chapter 3, at high frequencies, human hearing are relatively insensitive to phase identity as comparing to energy. The intensity stereo coding technique already exploits this characteristic of human hearing to further reduce irrelevancy. Here, we again take advantage of this characteristic, i.e. replace the sign bits of high frequency data with our embedded bits. Because we've known that the sign bits of high frequency data are unimportant, and the errors caused by this replacement could be insignificant to human perception.

At even higher frequency range, the embedding bits can replace the data directly. That is, not only sign bits but also data quantities will be changed. The advantage of this method is that even a zero-valued data can be embedded data. There is no sign bits saved in bitstream for zero-valued data, so these data always have to be avoid in the previously-stated method(i.e. replacing the sign bit). However, this method should be applied very carefully, or the music quality would be damaged seriously. Thus, to preserve the audio quality, this method is not utilized finally.

Fig. 6.3 shows the basic implementation flow of data embedded method.

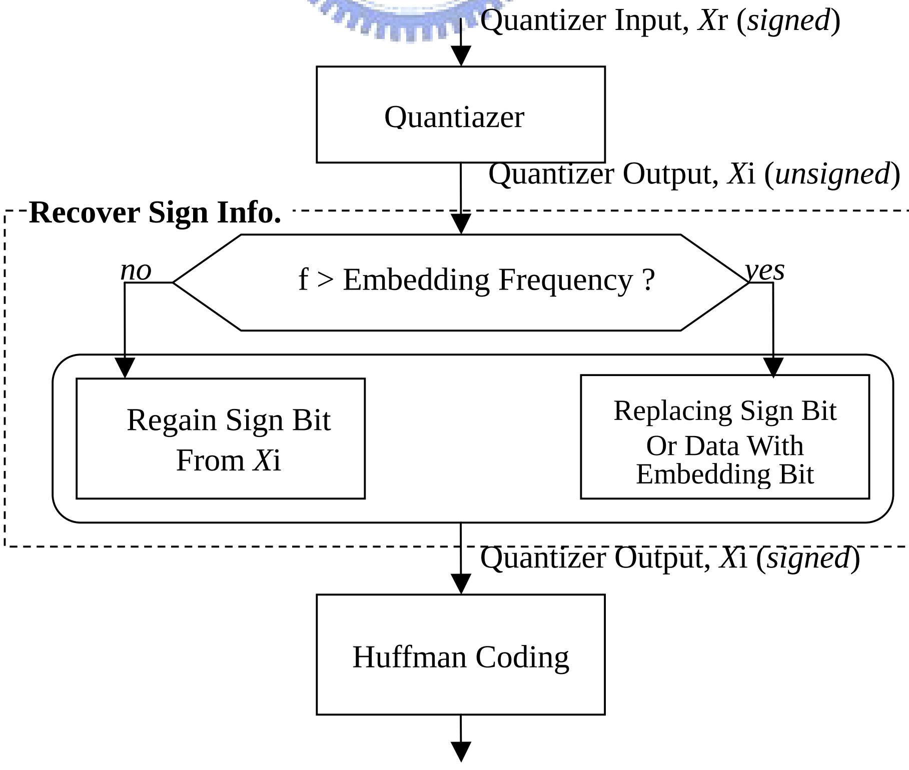
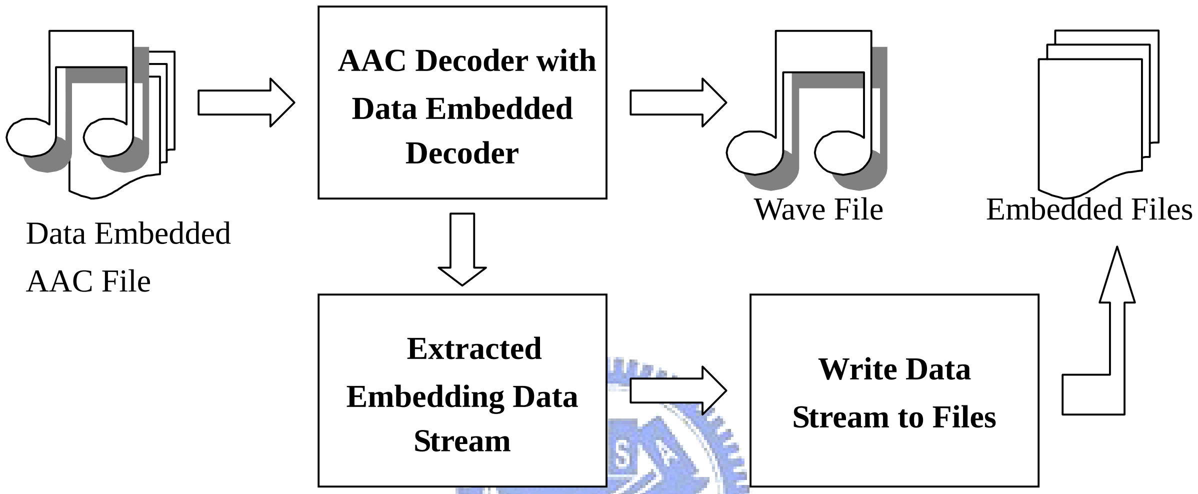

Quantizer Input, *X*r (*signed*)
Quantiazer
Quantizer Output, *X*i (*unsigned*)
Recover Sign Info.
f > Embedding Frequency ?
no
yes
Regain Sign Bit From *X*i
Replacing Sign Bit Or Data With Embedding Bit
Quantizer Output, *X*i (*signed*)
Huffman Coding

Fig. 6. 3 The implementation flow of data embedded method.

6.3 Implementation of Data Embedded Decoder

Fig. 6.4 shows the basic structure of data embedded AAC decoder. The data embedded decoding algorithm is also bundle within the AAC decoding process



Data Embedded AAC File
AAC Decoder with Data Embedded Decoder
Wave File
Embedded Files
Extracted Embedding Data Stream
Write Data Stream to Files

Fig. 6. 4 The structure of data embedded AAC decoder

The embedded data will be extracted bit by bit during AAC decoding process. And the embedded data stream are extracted to be analyzed by data stream analyzer and then reconstructed according to their headers. Thus, there are two kinds of final outputs, including general decoded wave file and embedded files. Fig. 6.5 shows the more detailed implementation flow of data embedded decoding procedure.

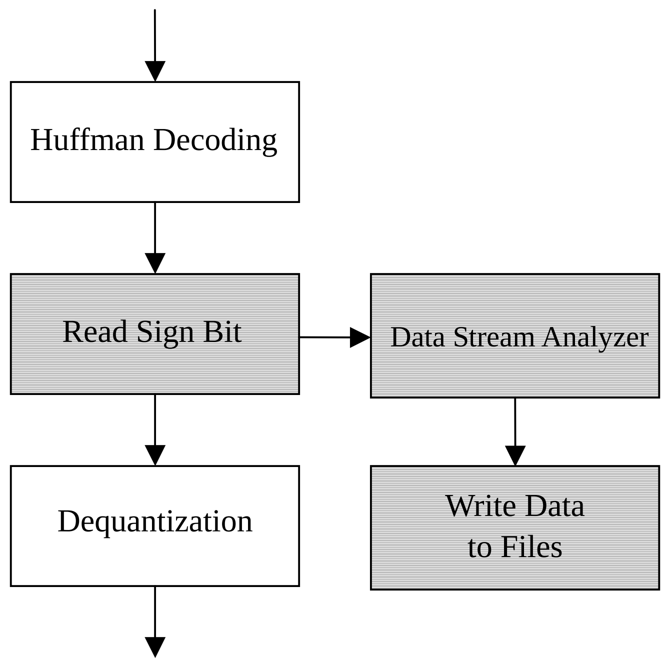


Fig. 6. 5 The implementation flow of data embedded decoding process

If a lyrics text file is embedded, a synchronous lyrics display is also implemented, so you can show the current lyrics while playing music. The data stream analyzer and lyrics analyzer, which mainly deals with lyrics display synchronization, are inherited from [32], to known more specifics, please see [32] and Appendix B.

CHAPTER 7. Experimental Results

This chapter will be divided into two parts, 7.1 shows the performance of the proposed MPEG-2/4 LC AAC encoding scheme, and 7.2 shows the results after the addition of data embedded method.

7.1 MPEG-2/4 LC AAC Encoder

The proposed AAC encoder is implemented on a 32-bit fixed-point processor, StrongARM SA-1110. The performance of the proposed AAC encoder, including the final resource distribution, encoding speed, memory usage and audio quality test results will be presented in the follows.

7.1.1 Resource Distribution

After modifications to the original algorithm, the computational requirements in each module have been changed. The final resource distribution is listed in Table 7.1. Note that since the psychoacoustic model has been nearly removed: block switching is discarded and allowed distortion evaluation is moved into the quantization module (by FAAC's implementation). Therefore, it is not presented in the Table 7.1.

Module	Percentage
Filter Bank	26.5%
TNS	5.8%
M/S Stereo	0.9%
Quantization	39.8%
Noiseless Coding	5.9%
Bitstream Formatting	13.7%
Others	7.4%

Table 7. 1 Distribution of resources in proposed AAC-LC encoder

Comparing with Table 4.1, the complexity of quantization module is reduced, and the computational requirements in other modules are relatively increased because of the lack of psychoacoustic model. Thus, from Table 7.1, we can quickly conclude that quantization module has been simplified significantly, since it even has a 30~40% reduction in resource requirement while the others are increased relatively. However, you may wonder why it seems that the computational effort consumed by filterbank module is increased. It is mainly because filterbank adopts lots of 64-bit calculation, or we could say the complexity reduced in filterbank module is not as much as the other modules, including TNS, M/S stereo, bitstream formatter and so on.

7.1.2 Resource Requirement Improvement

Comparing with the original source code, 86.36% of the RAM requirement is reduced and also 3.36% of the ROM requirement is reduced. The reason why ROM size is not significantly decreased is mainly due to that most mathematical functions realized by *math.h* of C library previously are implemented now by linear approximations or lookup tables instead. And this increases the ROM size.

Memory	Reduction Percentage
RAM	86.36%
ROM	3.36%

Table 7. 2 Comparing the resource requirement with the original source code

The ROM size is obtained by *objdump*, and the RAM size is obtained by the memory usage information reported by the Unix command “*top*” which provides an on-going look at processes in real time, including memory and CPU usage, etc.. Since it’s hard to precisely estimate the run-time memory usage for a software in the C code stage, however, the memory requirement is usually an important issue to embedded systems. Therefore, we employ “*top*” to give a simple rough estimation.

7.1.3 Encoding Speed

Since our proposed encoder is targeted at 96kbps encoding, all encoding speed is test under 96kbps bitrate settings.

Test	Length	Bitrate	Speed
Violoncello	0:30	96kbps	1.54 X
Soprano	0:23	96kbps	1.54 X
Bass	0:24	96kbps	1.44 X

Table 7. 3 Encoding speed

From the Table 7.3 we can see that our proposed encoder can at least run at the speed of 1X on our demo platform, PCM-7130 with StrongARM SA-1110 processor.

7.1.4 Quality Evaluation

To evaluate the audio quality of proposed AAC encoder, an objective evaluation is applied. The objective audio quality evaluation is done by using a software objective measurement tool for audio quality tool called EAQUAL, which stands for Evaluation of Audio QUality, and is implemented based on the recommendation ITU-R BS.1387. A brief introduction can be found in Appendix C.

The reference codec is traditional AAC encoder and the test codec is the proposed AAC encoder. Both of them are decoded by FAAD[15] implemented in floating-point. The "Diffgrade (DG)" is the objective rating given to the test item minus the rating given to the reference item. The DG scale can be divided into five ranges: " imperceptible (>0.00)", "perceptible but not annoying (0.00 ~ -1.00)", "slight annoying (-1.00 ~ -2.00)", "annoying (-2.00 ~ -3.00)" and "very annoying (-3.00 ~ -4.00)".

Bitrate	**Test Audio Sample**		
	Violoncello	Soprano	Bass
128kbps	0.67	0.60	-0.07
96kbps	0.15	-0.04	-0.43
64kbps	-0.05	-0.23	-0.51

Table 7. 4 The objective test results of proposed encoder

7.2 Data Embedded Method

After the addition of the data embedded feature, the performance analysis is discussed in the follows.

7.2.1 Resource Distribution

According to Table 7.5, we can see that only about 0.3% of computation resource is required by data embedded module. Thus it is quite less computation demanding.

Module	**Percentage**
Data Embedded	*0.3%*
Filter Bank	24.4%
TNS	6.6%
M/S Stereo	1.0%
Quantization	39.9%
Noiseless Coding	6.0 %
Bitstream Formatting	13.8%
Others	8.0%

Table 7. 5 Resource distribution of the proposed encoder plus data embedded method

7.2.2 Encoding Speed and File Size

Some comparisons based on experimental results are listed below, including file size, encoding speed and embedded bits count. Table 7.6 shows the resulting file size and encoding speed before and after data embedded method presented.

Test Sample	Length	Bitrate	File size		Speed	
			Before	After	Before	After
Always[24]	5:49	96kbps	3.98 MB	3.98 MB	1.33 X	1.32 X
Thank[24]	3:39	96kbps	2.50 MB	2.50 MB	1.32 X	1.31 X
Torn[24]	3:56	96kbps	2.74 MB	2.74 MB	1.28 X	1.28 X

Table 7. 6 The resulting file size and encoding speed before and after data embedded method presented

The results in Table 7.6 reveals the most attractive properties of this data embedded method. The file size remains the same, and only increases a small amount of computing complexity to the original proposed AAC encoder.

7.2.3 Embedded Data Size

Some resulting embedded data size of different files are listed in Table 7.7, and also presents the comparison with Huang's implementation of MP3 at 128kbps [32].

Test Sample	Length	Bitrate	Embedded bits	Huang's
Always	5:49	96kbps	46.2KB	N/A
		128kbps	110.0KB	163.1KB
Thank	3:39	96kbps	36.2KB	N/A
		128kbps	77.7KB	122.8KB
Torn	3:56	96kbps	47.4KB	N/A
		128kbps	113.0KB	162.0KB

Table 7. 7 The embedded bits count of different files

From Table 7.7, we find that, even at the same bitrate, i.e. 128kbps, the embedded bits count of AAC is still less than Huang's implementation of MP3 for about 50KB. One main reason is that this proposed AAC encoder exploits ABR encoding, thus no bit reservoir is presented. And observing the results from [32], bit reservoir often contributes 30~50KB data embedding space. Another reason is that AAC system does not provides "count 1 region data embedded method" as mentioned in Huang's thesis. Though there's no so-called "count 1 region" defined in AAC, inheriting the same concept from MP3, we should still apply the same method. However, the so called "count 1 region" in AAC, usually starts at around 7.9 kHz, which is much lower in frequency range than MP3 (~12kHz). It could cause great damage if applying "count 1 region data embedded method" as what MP3 does. Also, due to the presentation of bandwidth control tool, the nonzero range has been narrowed, to preserve the music quality, this method is not utilized finally.

7.2.4 Quality Evaluation

We use the same objective audio quality evaluation method as that in pure AAC encoder case, that is by applying EAQUAL. The reference codec is also the ISO method implemented encoder and then decoded by FAAD[15]. We use the "Diffgrade (DG)" here to evaluate audio quality again. The resulting DG of both pure proposed encoder and the proposed encoder with the addition of data embedded method are listed below for comparison.

Test Sample	Length	Bitrate	None	Data embedded
Always	5:49	96kbps	-1.26	-1.31
		128kbps	-0.67	-0.52
Thank	3:39	96kbps	-0.88	-0.92
		128kbps	-0.57	-0.42
Torn	3:56	96kbps	-1.85	-1.92
		128kbps	-0.22	-0.49

Table 7. 8 The objective test results of proposed encoder plus data embedded feature

Experimental results reveal that this data embedded method which based on psychoacoustic characteristic of human hearing only causes small degradation of music quality.

CHAPTER 8. Conclusions and Future Works

The conclusions of this thesis and future possibilities are shown in the follows.

8.1 Conclusions

In this thesis, we give a brief introduction of MPEG-2 AAC encoding algorithm, a proposed fast algorithm including:

- Removal of block switching
- Adopting fast MDCT algorithm
- Simplified TNS with a early decision mechanism
- Simplified Mid/Side stereo coding
- Faster quantization with scalefactor prediction and simplified QuantizeBand() function
- Simplified implementation of math functions

In the fixed-point stage, to further control the error caused by the approximating fixed-point arithmetic, a dynamic 32/64 bit implementation of FFT and a bandwidth control module are applied.

For additional feature, a data embedded method is also applied to AAC file. Both AAC encoders with and without data embedding are realized on a 32-bit RISC processor, Intel® StrongARM SA-1110. Finally, the performance analysis, the subjective and objective sound quality test results, and the comparison with other implementations are presented.

Experimental result shows that the 32-bit fixed-point implementation of proposed algorithm can perform at least 1X encoding by the processor, Intel® StrongARM SA-1110 which is capable of running at up to 206 MHz. And comparing with the original ISO implemented source code, 86.36% of RAM requirement and 3% of ROM requirement are reduced.

8.2 Future Works

The proposed encoder in this thesis is mainly concentrate in C code level, to be more adapted to embedded system applications, converting it into assembly code and perform further optimization accordingly is necessary. In this way, the proposed AAC encoder can get even better performance. Also, at present, we haven’t paid much attention to the memory usage optimization. However, this is quite important in those resource limited systems. Thus, these are two essential works in the future.

We can see that selling music in digitalized format through Internet rather than selling CDs in record stores seems to be an irresistible trend. Though AAC itself has the advantages of smaller file size and better quality, to convince traditional music company taking AAC format as the distributing standard is not enough. Thus, corresponding encryption should be implemented as well. Together with data embedded algorithm and encryption, AAC can be the first choice of standard format delivery music through Internet.

To further expand the application of this thesis, the concepts proposed can be applied to many other developed modern audio compression formats, such as Dolby

AC-3, MPEG-1 Layer III, Microsoft® WMA, and Ogg Vorbis, to increase their performance, too. Especially, MPEG-2 AAC constitutes the kernel of MPEG-4 General Audio (GA). MPEG-4 GA generally based on MPEG-2 AAC structure and with some enhancement and refinement. What have been standardized in MPEG-2 AAC, including low complexity (LC) profile, main profile and scalable sampling rate (SSR) profile, are also presented in MPEG-4 AAC. Thus, the proposed algorithm is most suitable for optimizing MPEG-4 audio.

REFERENCES

[1]. K. Brandenburg, M. Bosi, S. Quackenbush, L. Fielder, K. Akagiri, H. Fuchs, M. Dietz, J. Herre, G. Davidson and Y. Oikawa, "ISO/IEC MPEG - 2 Advanced Audio Coding", *J. Audio Eng. Soc.*, October 1997, pp. 789 – 811.

[2]. ISO/IEC 13818 – 7, "Information Technology – Generic Coding of Moving Pictures and Associated Audio, Part 7: Advanced Audio Coding," 1997.

[3]. N. Jayant and P. Noll, "Digital Coding of waveforms", Prentice-Hall, Englewood Cliffs, NJ, 1984.

[4]. J. Herre and J. D. Johnston, "Enhancing the Performance of Perceptual Audio Coders by Using Temporal Noise Shaping (TNS)," *101st AES convention*, Preprint 4384.

[5]. MPEG Audio FAQ [online]

URL: http://www.tnt.uni-hannover.de/project/mpeg/audio/faq/

[6]. J. D. Johnston and A. J. Ferreira, "Sum-Difference Stereo Transform Coding," *Proc. IEEE ICASSP*, 1992, pp. 569 – 572.

[7]. T. T. Sandel, D. C. Teas, W. E. Feddersen and Jeffress, " Localization of Sound From Single and Paired Sources," *J. Audio Eng. Soc. Am. 27*, 1955, pp.842 – 852.

[8]. B. C. J. Moore, "An Introduction to the Psychology of Hearing, " 3rd ed., Academic Press, NY, 1989.

[9]. M. L. Honig, and D. G. Messerschmitt , "Adaptive Filters: Structures, Algorithms, and Applications, " Kluwer Academic, 1984.

[10]. C. F. N. Cowan, P. M. Grant and P. F. Adams, " Adaptive Filters, " Prentice-Hall, Englewood Cliffs, 1985.

[11]. H. Fuchs, "Improving MPEG Audio Coding by Backward Adaptive Linear Stereo Prediction," *99st AES convention*, Preprint 4086.

[12]. S. R. Quackenbush and J. D. Johnston, "Noiseless Coding of Quantized Spectral Components in MPEG-2 Advanced Audio Coding," *IEEE ASSP*, 1997, pp. 1 – 4.

[13]. R.G. v. d. Waal and R. N. J. Veldhuis, "Subband Coding of Stereophonic Digital Audio Signals," *IEEE ICASSP*, 1991, pp. 3601 – 3604.

[14]. J. Herre, K. Brandenburg , and D. Lederer, "Intensity Stereo Coding," *96st AES convention*, Preprint 3799.

[15]. FAAC – Freeware Advanced Audio Coder [online]

URL: http://www.audiocoding.com

The proposed source code is modified based on FAAC's implementation.

[16]. E. Zwicher and H. Fastl, "Psychoacoustics: Facts and Models," Springer-Verlag, 1990.

[17]. T. Painter and A. Spanias, "A Review of Algorithms for Perceptual Coding of Digital Audio Signals," *DSP '97 Conference*, 1997, pp. 179 – 209.

[18]. E. Terhardt, "Calculating Virtual Pitch," Hearing Research, pp. 155-182, 1979.

[19]. Multimedia and Streaming [online]

URL: http://www.liacs.nl/~joostd/WebTech/Day6/slides/multimedia.html

[20]. R. Gluth, "Regular FFT-Related Transform Kernels for DCT/DST-based polyphase filter banks," *IEEE ICASSP* 1991, vol.3, pp. 2205 – 2208.

[21]. E. Kurniawati, C. T. Lau, B. Premkumar, J. Absar and S. George, "New Implementation of Techniques of an Efficient MPEG Advanced Audio Coder," *IEEE Transactions on Consumer Electronics*, Vol. 50, No. 2, MAY 2004, pp. 655 – 665.

[22]. H. oh, J. Kim, C. Song, Y. Park and D. Youn, "Low Power MPEG/Audio Encoders Using Simplified Psychoacoustic Model and Fast Bit Allocation," *IEEE Transactions on Consumer Electronics*, Vol. 47, No. 3, August 2001, pp. 613 – 621.

[23]. SQAM – Sound Quality Assessment Material: EBU SQAM disc tracks.

URL: http://www.tnt.uni-hannover.de/project/mpeg/audio/sqam/

[24]. Test Audio Sample Description –

Elliott: Artist/Elliott Smith, Album/From a Basement on the Hill, Title/A Fond Farewell, Label/Anti.

Jeff: Artist/Jeff Buckley, Album/Grace, Title/So Real, Label/Columbia.

Radio:Artist/Radiohead, Album/The Bends, Title/High and Dry, Label/ Parlophone.

Devic: Artist/The Devics, Album/The Stars at Saint Andrea, Title/Red Morning, Label/Bellaire

Sandee: Artist/Sandee Chan, Album/When We All Wept in Silence, Title/Track 03, Label/Music 543.

Always: Artist/Bon Jovi, Album/Cross Road, Title/Always, Label/Mercury.

Thank: Artist/Dido, Album/No Angel, Title/Thank You, Label/Arista.

Torn:Artist/Natalie Imbruglia, Album/Left of the Middle, Title/Torn, Label/RCA.

[25]. S. Cramer and R. Gluth, "Computationally Efficient Real-Valued Filter Banks Based on a Modified O^2DFT," Signal Processing V, Elsevier Sc. Publ., Proc. EUSIPCO 90, Barcelona, 1990.

[26]. G. Bonnerot and M. Bellanger, "Odd-Time Odd-Frequency Discrete Fourier Transform for Symmetric Real-Valued Series," *IEEE Proceedings*, March 1976, pp. 392 – 393.

[27]. ISO/IEC 11172-3, "Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s, Part 3: Audio," 1992.

[28]. Advanced RISC Machines Ltd. [online]

URL: http://www.arm.com/

[29]. Advanced RISC Machines Ltd., "Application Note 34: Writing Efficient C for ARM," 1998.

[30]. Intel Corporation [online]

URL: http://www.intel.com/

[31]. Y. S. Lin, "MPEG-1 Layer III Audio Codec Optimization and Implementation on a DSP Chip," Master thesis submitted to department of Electrical and Control Engineering, National Chiao Tung University, July 2004.

[32]. R. H. Huang, "A Study of Data Embedded Method on MPEG/Audio and

Implementation of Data Embedded Decoder on the ADSP-2181 DSP Processor," Master thesis submitted to department of Electrical and Control Engineering, National Chiao Tung University, July 2004.

[33]. Apple Computer, Inc. [online]

URL: http://www.apple.com/

[34]. Fraunhofer Institute [online]

URL: http://www.iis.fraunhofer.de/

[35]. Advantech PCM-7130 User Manual

[36]. Advantech PCM-7130 Data Sheet

[37]. Advantech Co., Ltd [online]

URL: http://www.advantech.com/

[38]. Thilo Thiede, William C. Treurniet, Roland Bitto, Christian Schmidmer, Thomas Sporer, John G. Beerends, Catherine Colomes, Michael Keyhl, Gerhard Stoll, Karlheinz Brandenburg and Bernhard Feiten, "PEAQ – The ITU Standard for Objective Measurement of Perceived Audio Quality," *J. Audio Eng. Soc.* Vol. 48, No.1/2, Jan/Feb 2000. pp. 3 – 29.

[39]. Proseminar Redundanz, Fehlertoleranz und Kompression [online]

URL: http://goethe.ira.uka.de/seminare/rftk/mp3/

APPENDIX A.   Advantech PCM-7130 SBC

The PCM-7130 is an Intel® StrongARM low-power RISC processor single board computer that is designed to serve power/environment critical applications. Fig. A.1 shows the appearance of the entire system.

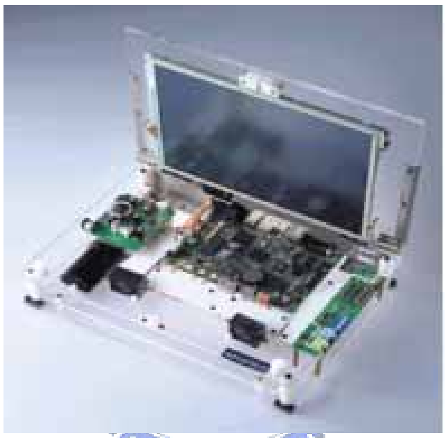


Fig. A. 1 The appearance of the Advantech PCM-7130 SBC [37]

The brief specifications are shown as following [35]:

- CPU: Intel® StrongARM SA-1110, 206 MHz
- Flash memory: up to 32 MB flash memory on board
- Memory: 64 MB SDRAM on board
- Watchdog timer: Dallas DS1670
- Audio: AC’97 stereo audio interface
- Dimensions: 145 x 102 mm
- Gross weight: 0.2 kg (0.4 lb.)
- SSD: 1 type-II CompactFlash™ card slot

- DIO: 8 digital inputs, 8 digital outputs
- Ethernet: 1 RJ-45 10Base-T port
- GPIO: 8
- IrDA: 1 IrDA interface
- PCMCIA: 1 type-II PCMCIA slot
- PS/2 port: 1 PS/2 port for keyboard/mouse
- Serial ports: 2 full RS-232 ports and 1 RS-485 port with automatic data flow control
- USB ports: 1 USB host and 1 USB client ports
- Display Chipset: Epson S1D13806 VGA controller
- LCD interface: 18-bit TFT active color LCD/16 bit DSTN passive color LCD; 20-pin header for 18-bit LVDS interface
- TV-out: supports both NTSC and PAL output
- Touchscreen: supports 4-wire resistive touchscreen via SPI (Serial Peripheral Interface)
- FCC Class A certified
- CE certified

Fig. A.2 gives a more detailed view of PCM-7130 with its peripheral interface.

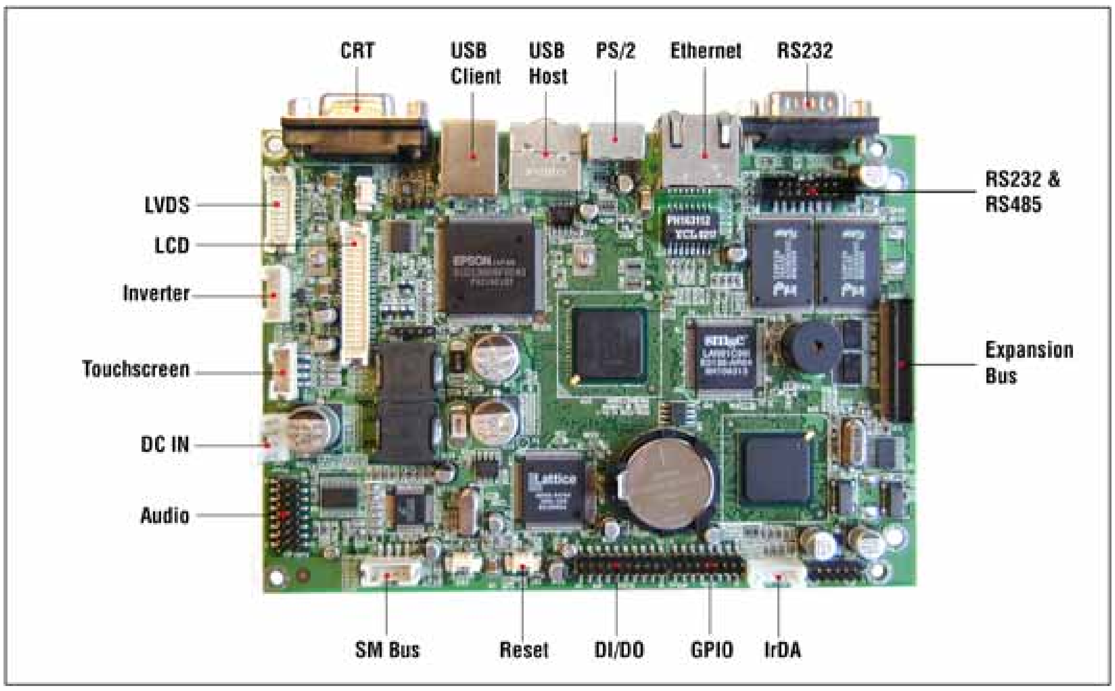


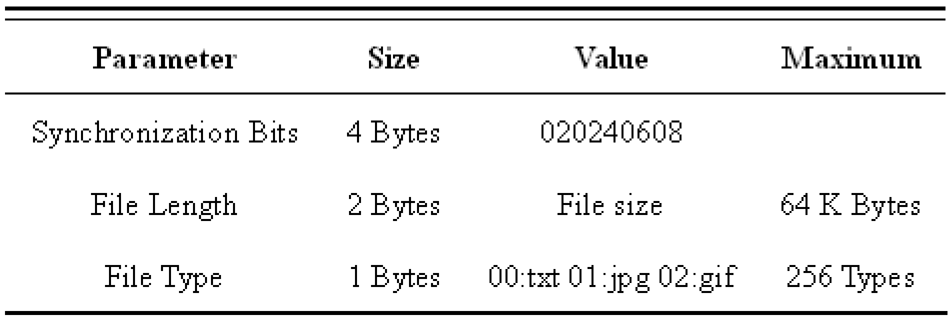
Fig. A. 2 Advantech PCM-7130 SBC [37]

APPENDIX B. Data Embedded Codec

B.1 Package File

The embedded data usually contain several files. These files should be embedded in series. Interleaving data of different files is not allowed. To avoid mix-up of data from different files, all embedding files will be bundled into one file, called package file, before entering encoding routine.

There are headers in the package file to recognize each file, and this information will be used while decoding. Each header consists of three parameters: synchronization bits, file type, and file length.



Parameter	Size	Value	Maximum
Synchronization Bits	4 Bytes	020240608	
File Length	2 Bytes	File size	64 K Bytes
File Type	1 Bytes	00:txt 01:jpg 02:gif	256 Types

Table B. 1 Parameters of the header in package file[32]

- *Synchronization Bits*: It is defined as “020240608”. And it is used to notify decoder that this is a start of new file.
- *File Length*: 2 bytes are used to record the file size.
- *File Type*: 1 byte is used to identify the file type. And it is very important when perform reconstruction in the decoder.

Fig. B.1 shows the file structure of package file:

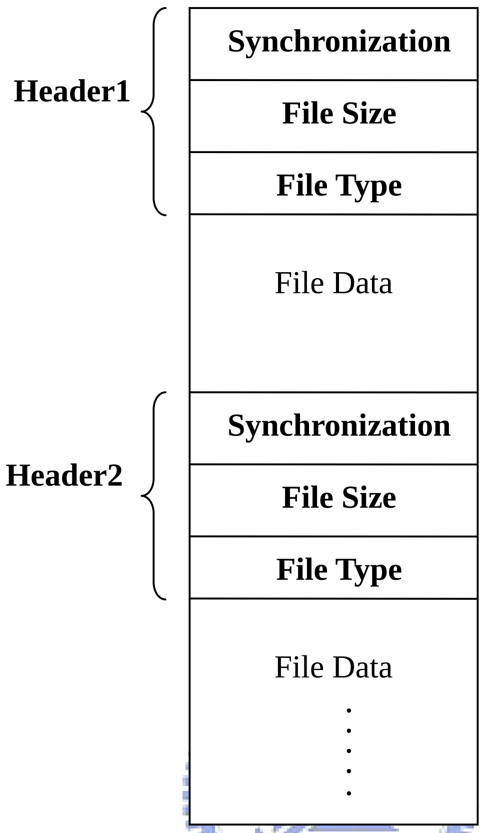


Fig. B. 1 The structure of package file [32]

To make sure the correctness of synchronous lyrics display, lyrics file must be embedded first. Thus, a sorting must be done before generating the package file. Fig. B.2 shows the flowchart of generating package file.

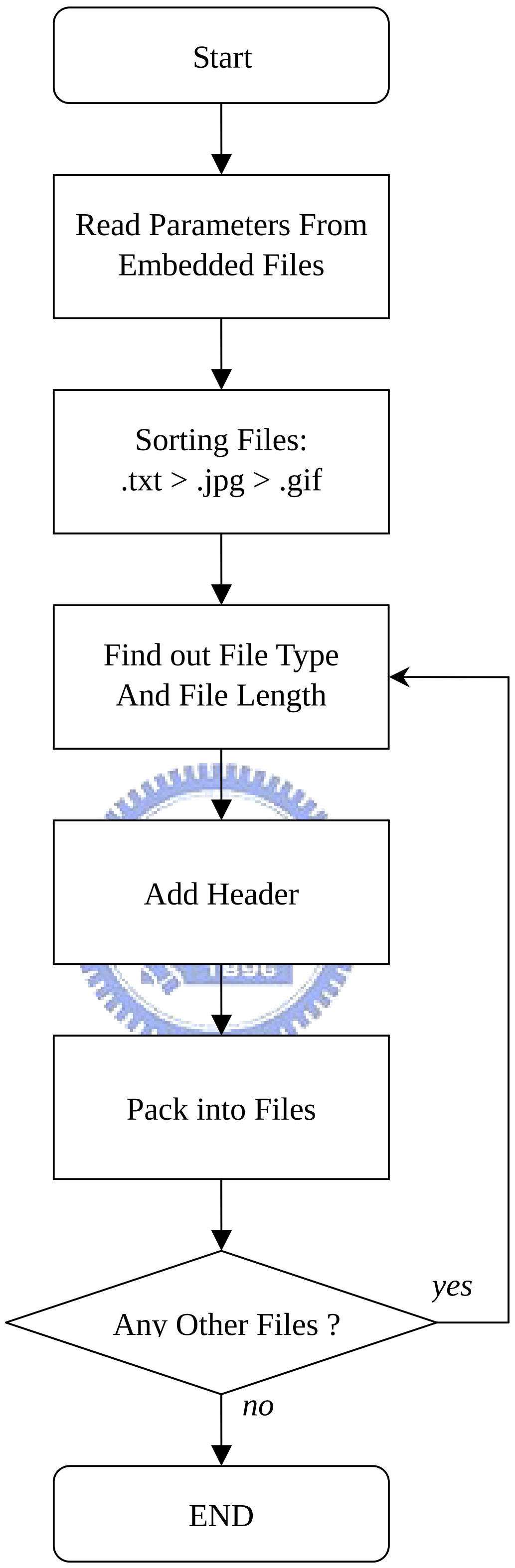

Start
Read Parameters From Embedded Files
Sorting Files: .txt > .jpg > .gif
Find out File Type And File Length
Add Header
Pack into Files
Any Other Files ?
yes
no
END

Fig. B. 2 The flowchart of generating package file [32]

B.2 Data Stream Analyzer

The data stream analyzer is used to analyze the data stream which is extracted by the data embedded decoder. The data stream is a series of signal of “0” and “1”, it must be analyzed and reconstructed to the original files by the data stream analyzer.

The flowchart of the data stream analyzer is shown as Fig. B.3 The purpose of stream analyzer is for data stream analyzing, including identifying synchronization, file length and file type, and processing every different files type.

“ Synchronization bits” is composed of 4 bytes. So at first 4 bytes should be read to check if they are synchronization bits. If not, left-shifting one byte and replenish one byte for checking, this process will continue until the synchronization bits are found. Afterwards, both file length and file type will be read. If the file type is identified as a lyrics file, those data will be saved to a lyric buffer, preparing to be shown synchronously on the screen while the song is playing. Other file types will be saved as files, then finishing the analysis of data stream.

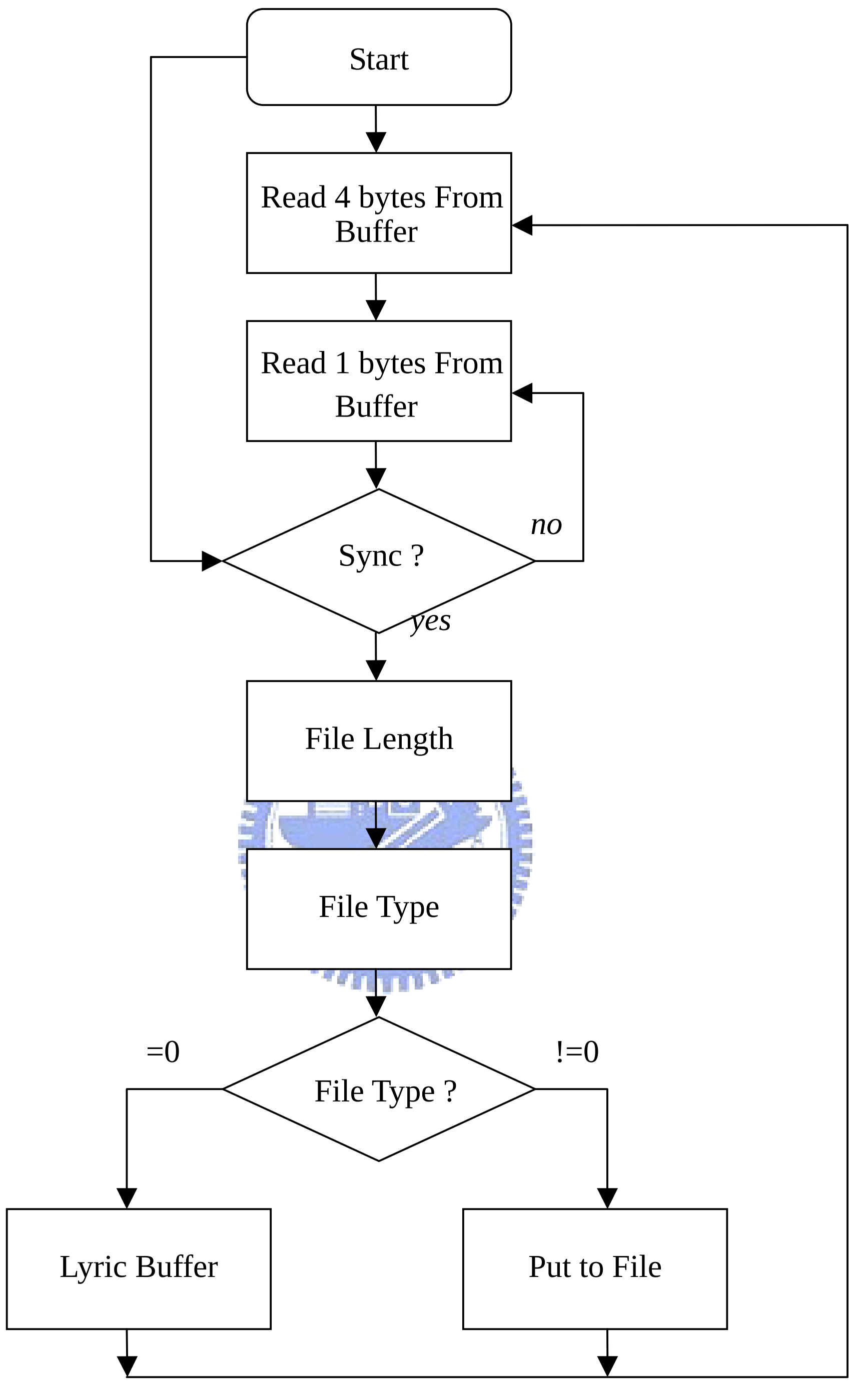


Fig. B. 3 The flowchart of data stream analyzer[32]

B.3 Lyrics Analyzer

Lyrics analyzer is used to analyze the lyrics file, and print lyrics synchronously while playing music, which is the same as what we see lyrics shown in KTV screen. In data stream analyzer, if the file type is identified as a lyrics text file, then those data will be saved temporarily to lyric buffer for analyzing by lyrics analyzer.

The lyrics file format is defined as ”*[mm:ss] the lyrics of a line*”, Fig. B.5 shows one example of lyrics file. “[” represents the beginning of the lyrics in every line, ”mm” represents the showing minutes of the lyrics, ”:” is for partition, and “ss” represents the showing seconds of the lyrics. From ”]” to Carriage Return/Linefeed (CR/LF) characters ”0D 0A”, they represent the contents of one line of the lyrics.

At first, one byte will be read to check if it’s the beginning of one line, that is ”[”. After finding that, the next five bytes will be analyzed to get the showing time of the lyrics. Following the “]” character is the content of current line of lyrics. The result will be saved to print buffer. Then, the analysis of next line of lyrics will begin again. This process will continue until the whole lyrics file are analyzed and all results are saved to print buffer. Fig. B.4 shows the flowchart.

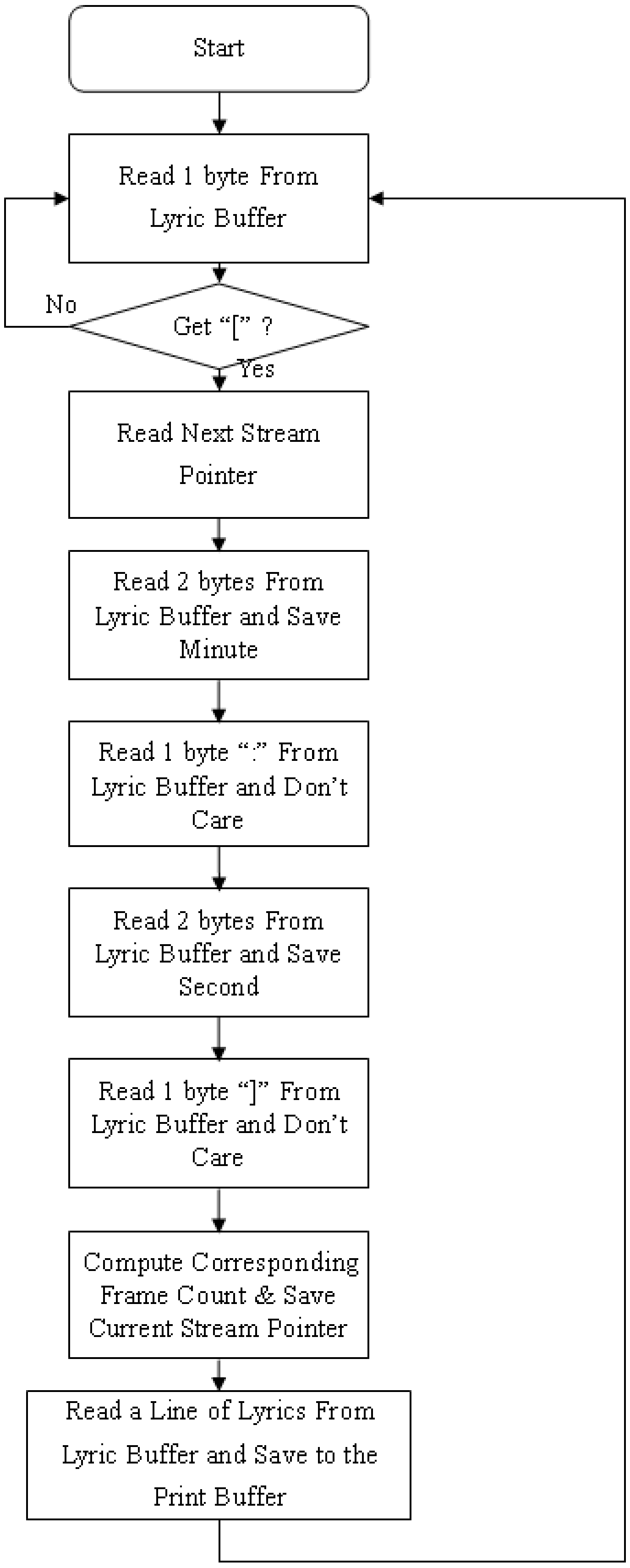
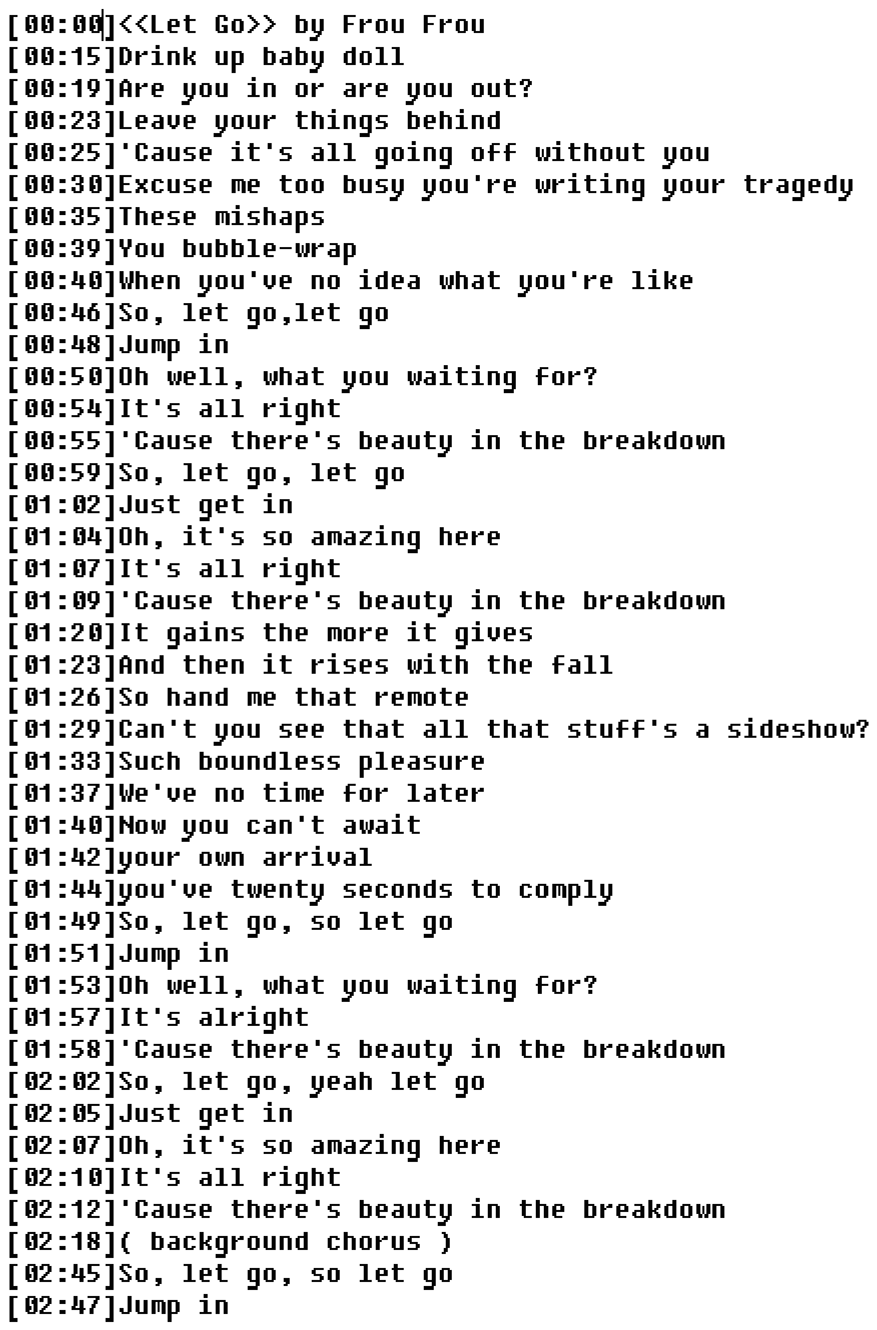

Start
Read 1 byte From Lyric Buffer
No
Get "[" ?
Yes
Read Next Stream Pointer
Read 2 bytes From Lyric Buffer and Save Minute
Read 1 byte ":" From Lyric Buffer and Don't Care
Read 2 bytes From Lyric Buffer and Save Second
Read 1 byte "]" From Lyric Buffer and Don't Care
Compute Corresponding Frame Count & Save Current Stream Pointer
Read a Line of Lyrics From Lyric Buffer and Save to the Print Buffer

Fig. B. 4 The flowchart of lyrics analyzer[32]



```
[00:00]<<Let Go>> by Frou Frou
[00:15]Drink up baby doll
[00:19]Are you in or are you out?
[00:23]Leave your things behind
[00:25]'Cause it's all going off without you
[00:30]Excuse me too busy you're writing your tragedy
[00:35]These mishaps
[00:39]You bubble-wrap
[00:40]When you've no idea what you're like
[00:46]So, let go,let go
[00:48]Jump in
[00:50]Oh well, what you waiting for?
[00:54]It's all right
[00:55]'Cause there's beauty in the breakdown
[00:59]So, let go, let go
[01:02]Just get in
[01:04]Oh, it's so amazing here
[01:07]It's all right
[01:09]'Cause there's beauty in the breakdown
[01:20]It gains the more it gives
[01:23]And then it rises with the fall
[01:26]So hand me that remote
[01:29]Can't you see that all that stuff's a sideshow?
[01:33]Such boundless pleasure
[01:37]We've no time for later
[01:40]Now you can't await
[01:42]your own arrival
[01:44]you've twenty seconds to comply
[01:49]So, let go, so let go
[01:51]Jump in
[01:53]Oh well, what you waiting for?
[01:57]It's alright
[01:58]'Cause there's beauty in the breakdown
[02:02]So, let go, yeah let go
[02:05]Just get in
[02:07]Oh, it's so amazing here
[02:10]It's all right
[02:12]'Cause there's beauty in the breakdown
[02:18]( background chorus )
[02:45]So, let go, so let go
[02:47]Jump in
```

Fig. B. 5 An example of lyrics file format

APPENDIX C. EAQUAL

The EAQUAL, which stands for Evaluation of Audio QUALity, is an audio quality evaluation software. It is implemented based on the recommendation ITU-R BS.1387.

In general, it compares a signal that has been processed in some way with the corresponding time-aligned original signal. And it extracts perceptually relevant features, which are used to compute a measure of quality. Also, a number of intermediary model output variables (MOVs) are available.

A selected set of MOVs are mapped to a final output score, called objective difference grade (ODG). The mapping was established by minimizing the difference between the distribution of objective measurements and the corresponding distribution of mean subjective qualities for an available data set.

The block diagram of EAQUAL is shown in Fig. C.1. It includes an ear model based on the fast Fourier transform (FFT). The model output values are based partly on the masked threshold concept and partly on a comparison of internal representations. The model outputs the partial loudness of nonlinear distortions, the partial loudness of linear distortions (signal components lost due to an unbalanced frequency response), a noise to mask ratio, measures of alterations of temporal envelopes, a measure of harmonics in the error signal, a probability of error detection and the proportion of signal frames containing audible distortions.

Selected output values are mapped to a single quality indicator by an artificial neural network with one hidden layer. For details, please see [38].

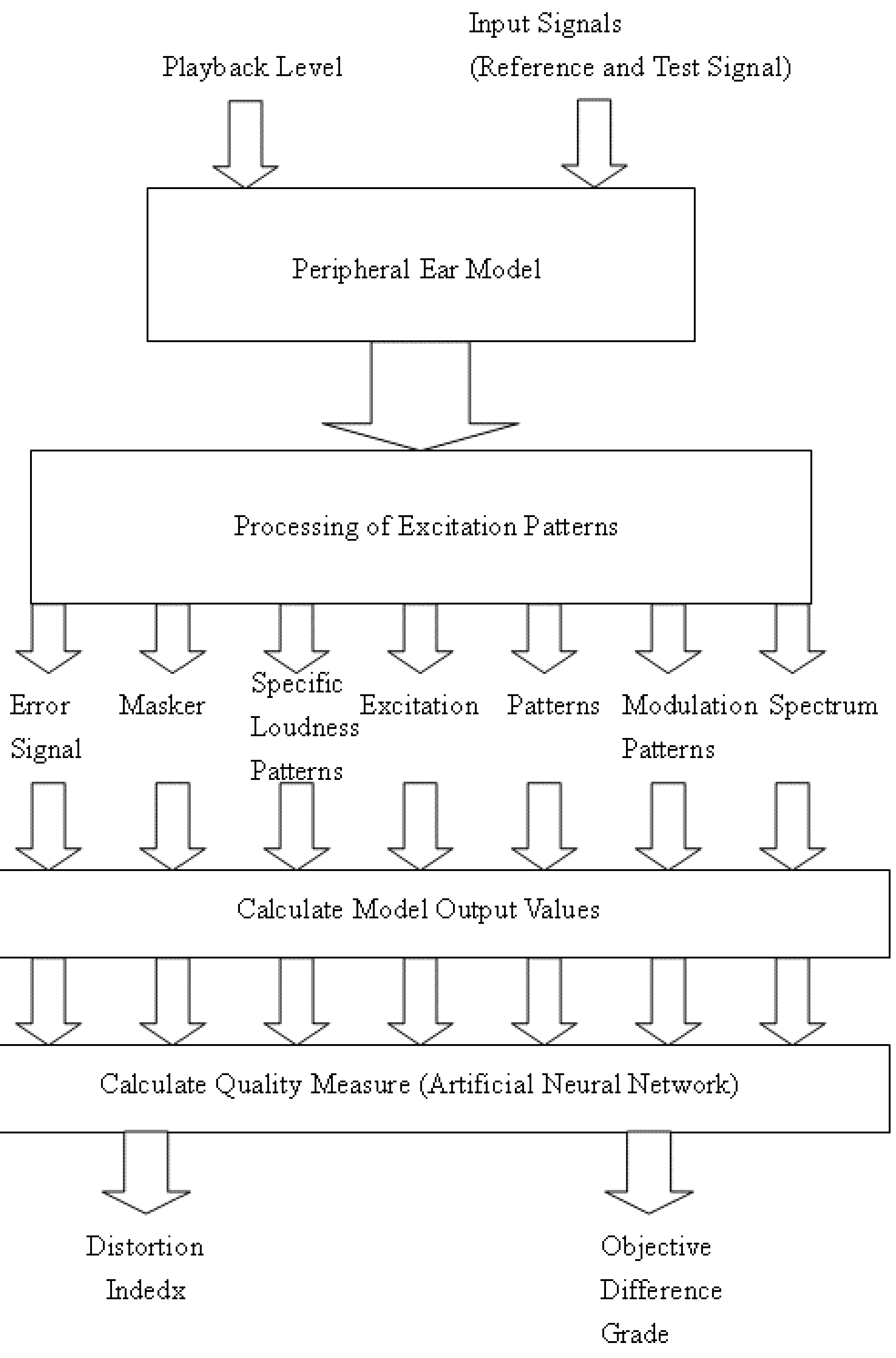

Playback Level
Input Signals (Reference and Test Signal)
Peripheral Ear Model
Processing of Excitation Patterns
Error Signal
Masker
Specific Loudness Patterns
Excitation
Patterns
Modulation Patterns
Spectrum
Calculate Model Output Values
Calculate Quality Measure (Artificial Neural Network)
Distortion Indedx
Objective Difference Grade

Fig. C. 1 Block diagram of measurement scheme[38]