國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩 士 論 文

MPEG-4 高效率音訊編解碼器之增速及其在 TI DSP 平台上的實現

MPEG-4 High Efficient AAC Codec Acceleration and Implementation on TI DSP

研 究 生：黃育彰

指導教授：杭學鳴 博士

中 華 民 國 九 十 五 年 六 月

MPEG-4 高效率音訊編解碼器

之增速及其在 TI DSP 平台上的實現

MPEG-4 High Efficient AAC Codec Acceleration and Implementation on TI DSP

研究生：黃育彰　　Student: Yu-Chang Huang

指導教授：杭學鳴　　Advisor: Dr. Hsueh-Ming Hang

國　立　交　通　大　學
電　子　工　程　學　系
碩　士　論　文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of Master

in

Electronics Engineering

June 2006

HsinChu, Taiwan, Republic of China

中華民國九十五年六月

MPEG-4 高效率音訊編解碼器之增速及其在 TI DSP 平台上的實現

研究生: 黃育彰　　　　　　　　　　　　　　　　指導教授: 杭學鳴 博士

國立交通大學

電子工程學系 電子研究所碩士班

摘要

由於數位音訊編碼技術的成熟發展，與音訊相關的產品，如 MP3 播放器、手機，在我們生活中扮演了一個很重要的角色。而所有的音訊壓縮標準中，MPEG-4 高效能音訊編碼(HE-AAC)提供了非常高的壓縮效率與不錯的音訊品質。在本篇論文中，針對 HE-AAC 編碼器的模組，我們提供了較快速的演算法，並且符合 DSP 系統的加速。我們也把 HE-AAC 編碼器實現在德州儀器公司(TI)的數位訊號處理器(DSP)上。

我們首先在 DSP 系統上針對 HE-AAC 編碼器分析其複雜度。我們發現 QMF bank(正交鏡像濾波器)、降頻濾波器、暫態訊號偵測器、心理學模式、量化模組在 DSP 系統上耗費了大部分的運算量。因此我們針對這些模組去研究，並且提供了快速演算法來降低其複雜度。

針對頻帶複製(SBR)編碼器部分，我們提供了快速的暫態訊號偵測器與有效率的降頻濾波器結構。針對 AAC 編碼器部分，我們提供了查表的方式來減少心理學模式的複雜度，並且也提出了簡化的視窗轉換、簡化的 TNS、快速的量化模組與簡化的短視窗分組方法。此外針對 DSP 架構，我們使用了很多加速的方法像是定點數運算、TI DSP 的特殊指令群、單個指令存取多筆資料、迴圈的分解與巨集指令。最後修改過的 HE-AAC 編碼器版本在 DSP 系統上執行的週期，在相同的最佳化設定之下，比最原始的改善了大約 55% 並且還能維持相同的音訊品質。

關鍵字：MPEG-4 HE-AAC、aacPlus、AAC、頻帶複製技術、DSP 系統加速

MPEG-4 High Efficient AAC Codec Acceleration and Implementation on TI DSP

Student: Yu-Chang Huang Advisor: Dr. Hsueh-Ming Hang

Department of Electronic Engineering &
Institute of Electronics
National Chiao Tung University

Abstract

Due to the recent advances of digital audio coding technology, audio coding related devices play an important role in our daily life such as MP3 players and mobile phone hand set. MPEG-4 High Efficient AAC (HE-AAC) provides a very high compression ratio and a good audio quality among all known audio coding standards. In this thesis, we propose several fast algorithms to speed up the MPEG-4 HE-AAC encoder for the DSP platform. Their implementations on the Texas Instrument (TI) TMS320C6416T fixed-point DSP are also presented.

We first analyze the complexity of HE-AAC encoder on a DSP system. We find that the QMF bank, transient detector, downsampling filter, psychoacoustic model (PAM), and quantization modules require the most operational cycles on DSP. Hence, we study and suggest several fast algorithms to reduce their complexities.

For the SBR encoder part, we propose a fast transient detector and an efficient decimation structure. For the AAC encoder part, we use a look-up table method to reduce the complexity of PAM and propose simplified block switching, simplified TNS, fast quantization and simplified short window grouping methods. We also adopt several DSP techniques to speed up the DSP, such as using fixed-point operation, intrinsic function, single instruction multiple data (SIMD), loop unrolling and macro function. Comparing to the original 3GPP HE-AAC encoder, the modified HE-AAC encoder save about 55% operational cycles under the same compiler optimization level and still maintains about the same audio quality.

Key words: MPEG-4 HE-AAC, aacPlus, AAC, Spectral Band Replication、DSP system acceleration.

誌謝

這篇論文能夠順利完成，最重要感謝的人是我的指導教授 杭學鳴 老師。在這二年的研究生涯中，老師不僅在做研究上給予指導，讓我在知識的探索上獲益良多，同時也關心我們的日常生活。老師除了豐富的學識和研究，謙虛、認真的待人處事態度，也是我景仰與學習的目標。

另外，要感謝的是與我在 audio 領域之中一起討論、研究的德宣與繼大學長，謝謝你們熱心地幫我解決了許多 audio 方面相關的疑問。也感謝從大學到研究所都一直共同打拼的鴻志、家賢、旻弘、崇諺，在修課中遇到問題時，大家互相討論，解決難題，讓我在這之中成長許多。此外，也感謝俊榮學長與家揚學長在我研究遇到困難時，給予很大的協助，使我在研究的過程中不至於徬徨不知方向。也感謝實驗室的其他成員，在碩士這兩年中帶給我歡樂、成長，在生活上給我支持、勉勵，使我充滿信心。也感謝通訊電子與訊號處理實驗室(commlab)，提供了充足的軟硬體資源，讓我在研究中不虞匱乏。

而我還想要感謝認識了 20 多年的好朋友，鈞平，在論文完成的最後階段，給予我在英文寫作上很大的幫助，也願你能在澳洲順利取得博士學位。還有要感謝玥含的陪伴，在生活中給予我關懷與鼓勵，使我在研究的過程中不孤單。

最後，要感謝的是我的媽媽、姊姊，不論在生活上、求學上給予我最大的鼓勵，他們的支持讓我能夠心無旁騖的從事研究工作。也要感謝我在天國的父親，沒讓你看到我穿碩士服實在有點可惜，願你在天國能分享到我完成碩士學位的喜悅。

在此僅以這篇論文獻給所有幫助過我，陪伴我走過這一段歲月的師長，同學，朋友與家人，謝謝!

誌於 2006.7 風城交大

育彰

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Introduction and Motivation

Due to the recent advances of digital audio coding technology, the audio related devices play an important role in our daily life such as hand set and MP3 player. In the last decade, CD-quality digital audio has essentially replaced analog audio. However, the emerging digital audio applications for digital broadcasting, storage, real-time communication, wireless system, and network faces a series of problems such as limited storage, narrow channel bandwidth and the real-time encode/decode implementation. Therefore, high quality audio coding at lower bitrate becomes necessary. In response to these requirements, many audio standards are proposed such as MPEG-1 Layer III (MP3), Microsoft WMA, MPEG-2/4 AAC, and MPEG-4 HE-AAC for high quality audio coding.

MPEG, which stands for Moving Pictures Experts Groups**,** is the name of family of standards used for coding audio-video information in a digital compressed format. It is a group work under the directives of the International Standard Organization (ISO) and the International Electro-technical Commission (IEC). The works of this group concentrate on defining the standards for coding moving pictures, audio and related data. MPEG-1 Layer III (MP3) which is proposed by MPEG is the most popular international audio coding standard in the market so far. Many MP3 portable devices having audio playing and recording functionality come into our daily life. For better audio quality at lower bitrates, the next generation audio coding standard, MPEG-2/4 Advanced Audio Coding (AAC) and its extension, High Efficient Advanced Audio Coding (HE-AAC), have been introduced.

AAC provides higher coding efficiency, multi-channel support, and high-quality at bitrates around 128k bps or lower. HE-AAC is a combination of AAC with the Spectral Band Replication (SBR) technology. SBR is known as the bandwidth extension technique, and the high frequency part of the signal bandwidth is reconstructed from the received low frequency part at the receiver. HE-AAC is able to deliver high-quality audio at bitrate down to 48k bps for stereo audio. However, the complexity of HE-AAC is higher due to the new technologies employed. The higher complexity has restricted the applications in a few ways. For portable device, the processing power of the devices is supplied by battery. If we can reduce the complexity of an audio algorithm, we can save the battery power. Also, it reduces the hardware cost. Therefore, we believe reducing the complexity of the HE-AAC and providing higher coding efficiency for the embedded system is very important for many applications and is an interesting challenge.

Many applications have already appeared in the market based on MPEG-4 HE-AAC. Digital radio broadcasting systems such as Digital Radio Mondiale (DRM) [8] are among the newest consumer audio products. They offer a more efficient use of the limited bandwidth available for broadcasting. The DRM system chooses the HE-AAC as the basic coding for their codec since it offers high audio quality at lower bitrates. HE-AAC is also recommended by 3GPP [9] forum for sound transmission as well as DVB [10] forum for DVB-H. For commercial examples, such as 3G mobile phone or iPod, they use the HE-AAC coding due to the limitation of storage and bandwidth constraints.

1.2 Overview of the thesis

This thesis concentrates on developing fast methods for improving encoding speed. The thesis is organized as follows. In Chapter 2, we discuss the algorithms of the low complexity profile (LC) of the MPEG-2 AAC encoder. Chapter 3 explains the MPEG-4 HE-AAC system and the algorithms of the SBR encoder module. In Chapter 4, we describe the DSP development environment and the acceleration methods for the TI C6416T DSP system. Chapter 5 describes our proposed algorithms to accelerate the HE-AAC and the implementation of the HE-AAC encoder on the DSP system. Several experiments are conducted to verify the proposed acceleration methods. In Chapter 6, we give conclusion remarks and possible future work.

Chapter 2
MPEG-2 Advanced Audio Coding

In this chapter, we will describe the basic concepts and the modules of MPEG-2 Advanced Audio Coding (AAC). MPEG-4 High Efficient AAC (HE-AAC) is a combination of Spectral Band Replication (SBR) tool and MPEG-2 AAC LC profile. In this chapter, we will explore MPEG-2 ACC in depth, whereas SBR tool will be introduced in the latter Section. The details of MPEG-2 ACC and SBR can be found in [1] and [2] respectively.

2.1 MPEG-2 Advanced Audio Coding

Starting from 1994, the Moving Pictures Expert Group (MPEG) audio standardization committee launched a higher quality multi-channel standard that is designed specifically to maintain MPEG-1 backward compatibility. This standard is known as MPEG-2 Backward Compatible (MPEG-2 BC). This is the predecessor of MPEG-2 AAC. In 1997, the formal specification of MPEG-2 AAC was formulated by the MPEG committee. Since MPEG-2 AAC is not backward compatible to the MPEG-1, MPEG-2 AAC is also known as the MPEG-2 Non-Backward Compatible (NBC). The aim of MPEG-2 AAC is to achieve indistinguishable audio quality at data rate of 384 kbps (or lower) for five full-bandwidth channel audio signals. AAC is the first codec system to fulfill the requirements of the International Telecommunication Union, Radio-communication Bureau (ITU-R) for the indistinguishable quality at 128 kbps for stereo.

Like all other perceptual coding schemes, MPEG-2 AAC compresses audio signals by removing the redundancy between the samples and the irrelevant audio signals. It uses time-frequency analysis for removing the redundancy between samples, and makes use of the signal masking properties of human hearing system to remove irrelevant audio signals. AAC combines the several coding efficiency tools like filter bank, temporal noise shaping (TNS), prediction techniques, gain control, quantization and huffman coding. In order to select between the quality and the memory/processing power for the different audio requirements, MPEG-2 AAC system offers three profiles:

- Main Profile (Main)

  The Main Profile provides the highest quality for applications where the amount of random access memory (RAM) requirement is not limited. It removes the prediction and gain control tools from AAC system. It also reduces the order of the Temporal Noise-Shaping (TNS) tool, thereby effectively alleviates the system complexity.
- Low-Complexity Profile (LC)

  The LC profile is intended to use when the computing cycles and memory requirements are constrained.
- Scalable Sampling Rate Profile (SSR)

  The SSR profile is in use when a scalable decoder is required. It adds the gain control tool to the LC profile.

MPEG-4 HE-AAC combines MPEG-2 AAC LC profile with SBR tool. We implement MPEG-4 HE-AAC on TI C64 DSP platform. Figure 2.1 shows the block diagram of MPEG-2 AAC encoder. Figure 2.2 shows the MPEG-2 AAC LC profile encoder block diagram. We will describe the MPEG-2 AAC LC profile coding tools in the following Section.

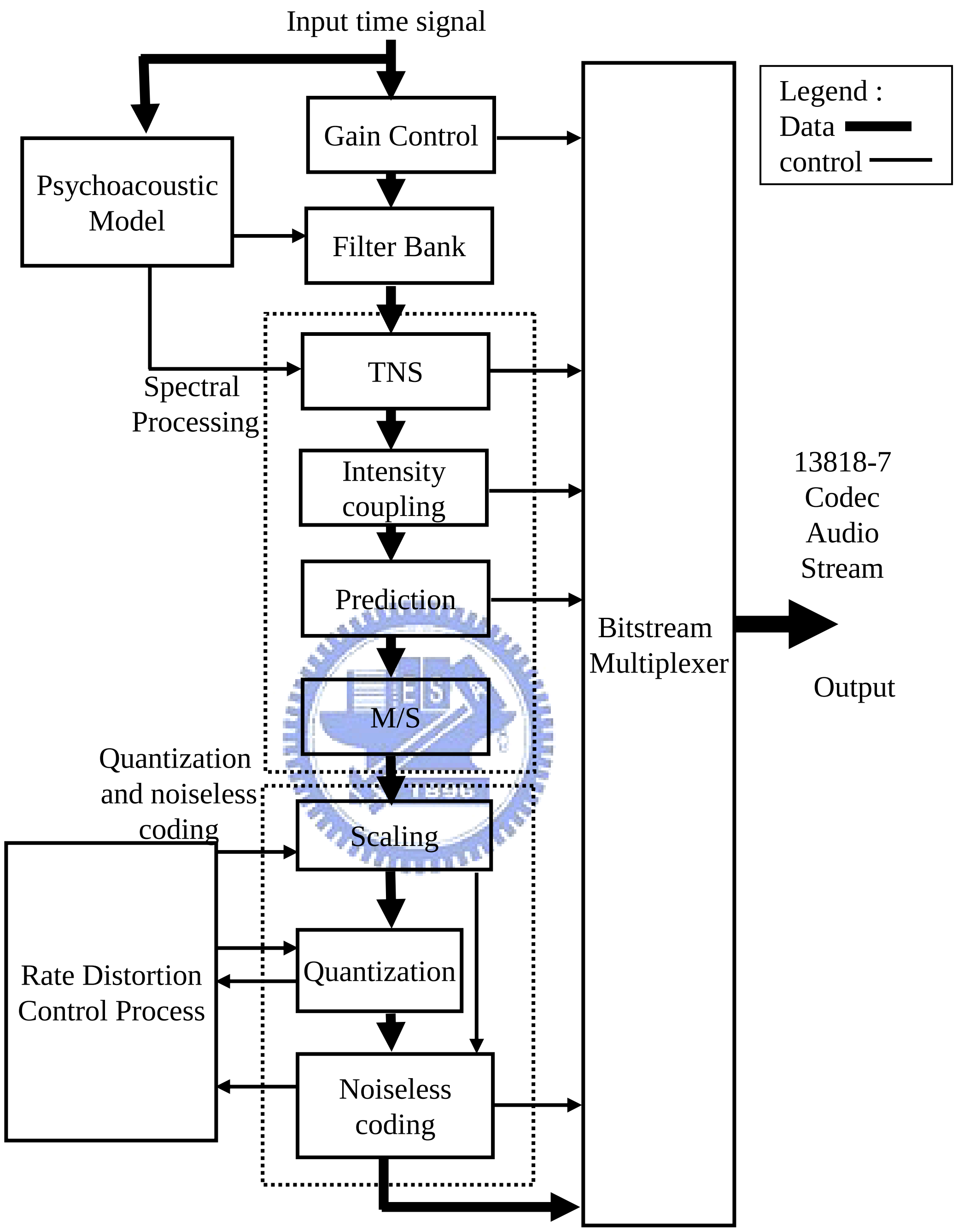


Figure 2.1 Block diagram for MPEG-2 AAC encoder [1].

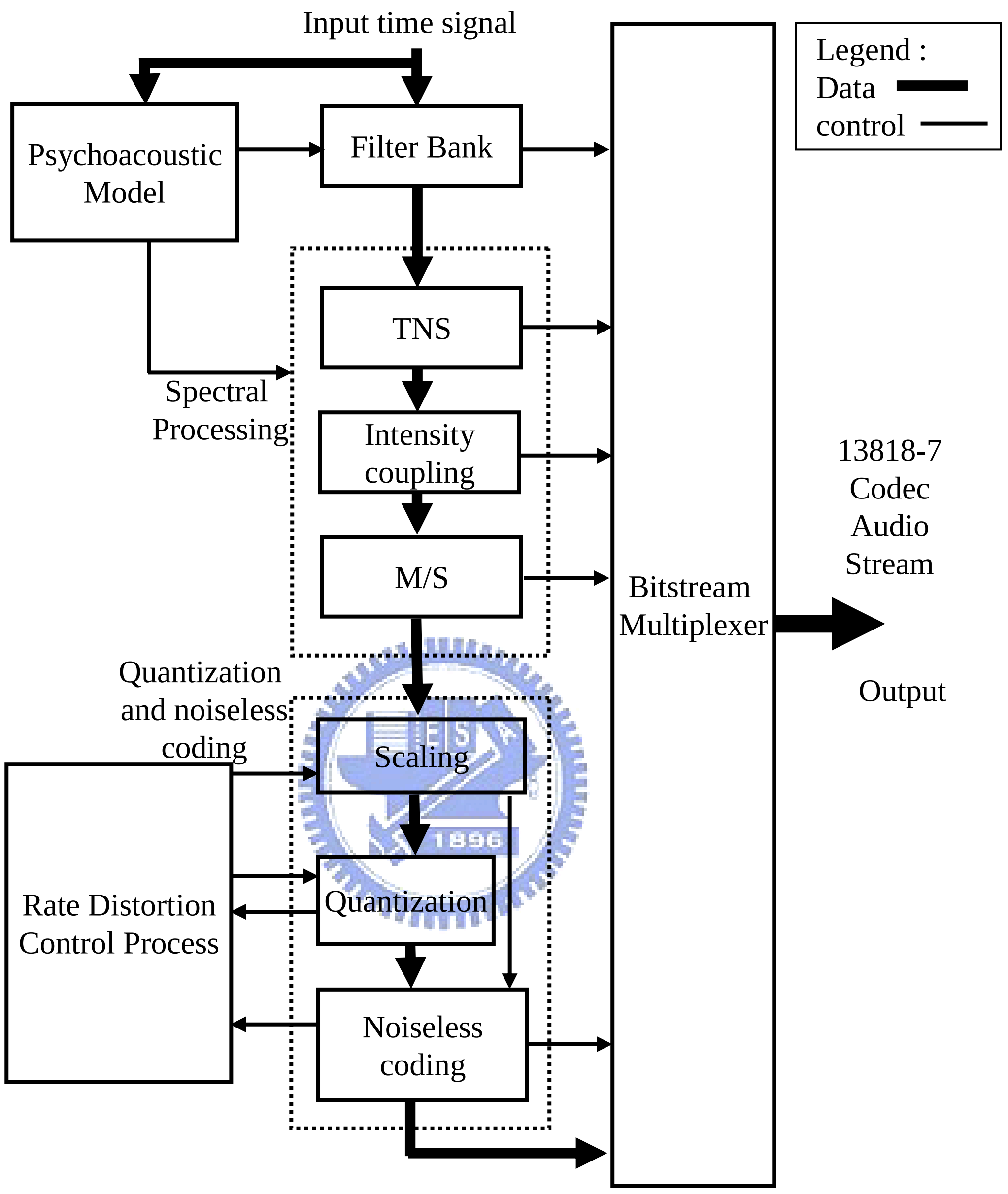


Figure 2.2 Block diagram for MPEG-2 AAC Low Complexity encoder [1].

2.2 Psychoacoustic Model

Characteristics of human hearing system are very important for audio compression. Psychoacoustic model has made significant progress toward characterizing human hearing system. The job of the psychoacoustic model is to analyze the input audio signal and determine to what extent the level of the spectrum quantization noise is allowable. By exploiting this principle of the psychoacoustic model, audio coder can use the signal energy to mask the noises which are generated in the quantization. There are some factors needed to calculate the masking threshold, they are: absolute threshold of human hearing in quiet, masking effect, critical band.

2.2.1 Absolute threshold of hearing in quiet

Absolute threshold of hearing in quiet is the threshold that one will perceive the signal in a noiseless environment. Figure 2.3 shows the absolute threshold of hearing in quiet. It is representative of a young listener with acute hearing. Sound waves with frequencies between 20 Hz and 20 kHz are called the audible sound frequencies. Human ear is most sensitive in middle frequency signals, especially from 2k to 4k Hz, but not sensitive to higher or lower frequency signals. Approximation function of the absolute threshold of hearing in quiet is as the following formula.

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

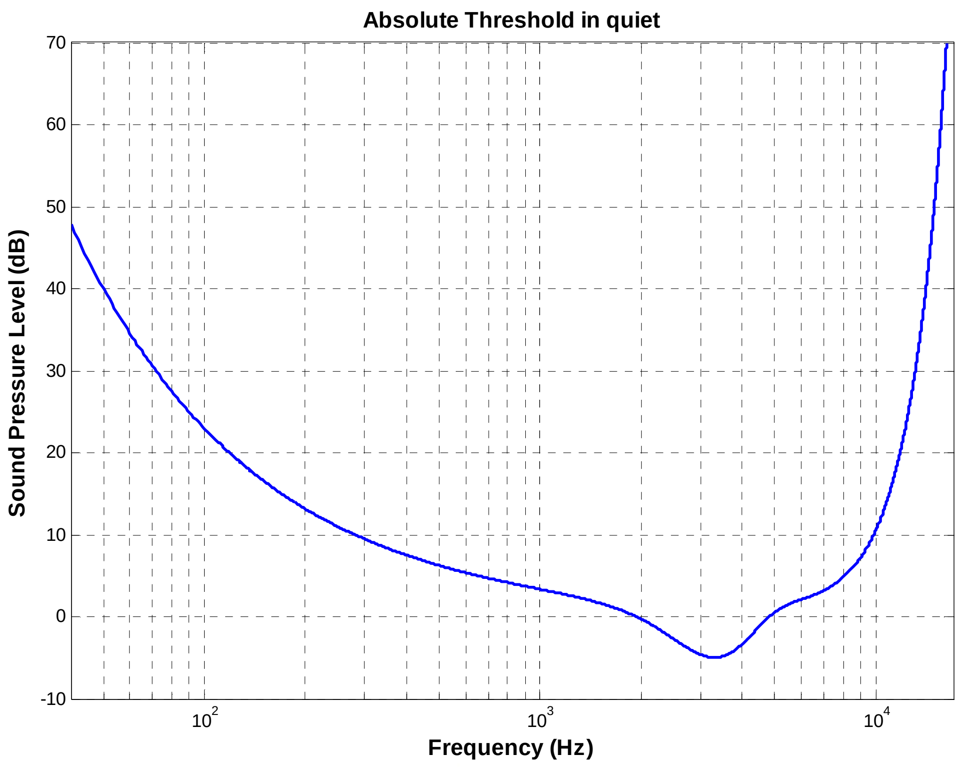


Figure 2.3 Absolute threshold of hearing in quiet.

2.2.2 Masking effect

Masking effect is an important factor to calculate the threshold. It is an important characteristic of human ear for compression. This phenomenon is that one sound (maskee) will be masked by another adjacent sound (masker). ‘Adjacent sound’ can be meant to the one closed to maskee in time domain or frequency domain.

(A) Frequency Masking

Figure 2.4 illustrates the frequency masking. The loud signals mask two other signals at nearby frequencies. The curve marks the “masking threshold” representing the audibility threshold for signals in the masking signal (masker). Other signals that are below the curve will not be heard when the masker is present. It does not need to transmit the signals which below the masking threshold, and allow much inaudible quantization noise.

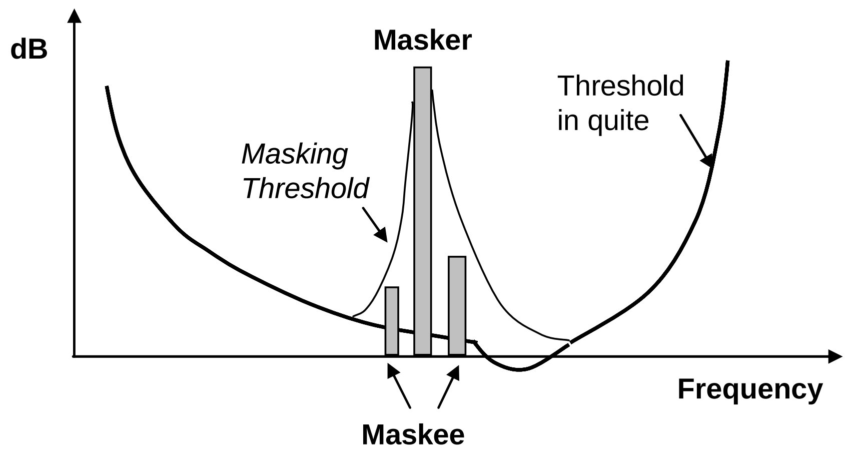


Figure 2.4 Frequency Masking.

(B) Temporal Masking

Figure 2.5 illustrates the temporal masking. Masking can occur prior to and after the presence of the masker. It is pre-masking and post-masking. Pre-masking takes place before the masker. Post-masking takes place after the masker is removed. The loud signals mask two other signals at nearby time. Figure 2.5 shows that post-masking can last for a long time, and pre-masking last for a short time.

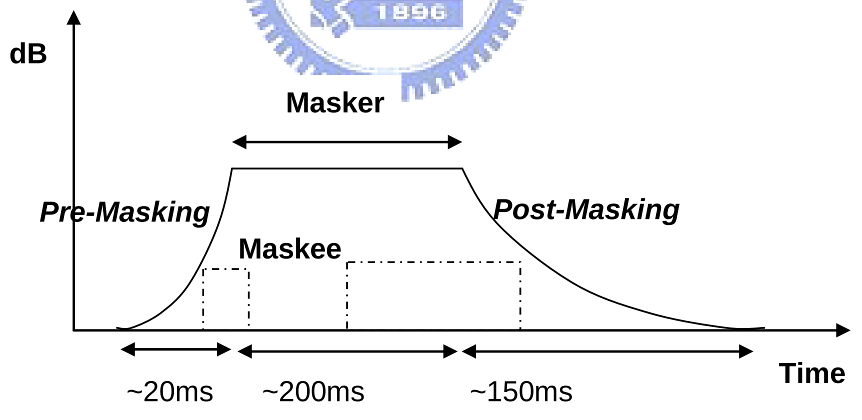


Figure 2.5 Temporal Masking.

2.2.3 Critical band

Human hearing performs a “frequency to place” mapping to analyze the spectrum of audio signals. Human ears have different sensitivities to audio signals in different frequency bands. These frequency bands are called the critical bands. The concept of critical band is that a masker exhibits a constant level of masking threshold regardless of the type of masker. The width of constant level masking is critical band. The unit of the critical band is “Bark”.

The MPEG-2 AAC psychoacoustic model based on the previous factor and calculates the maximum distortion energy (masking threshold). The threshold calculation has three inputs. Describing them in following :

1. The shift length for the threshold calculation process is called *iblen*. This *iblen* must remain constant over any particular application of the threshold calculation process. For long FFT *iblen* = 1024, for short FFT *iblen* = 128.
2. For each FFT type, the newest *iblen* samples of the signal, with the samples delayed (either in the filterbank or psychoacoustic calculation) such that the window of the psychoacoustic calculation is centered in the time-window of the codec time/frequency transform.
3. The sampling rate. There are sets of tables that will be used in the calculation threshold process, and the tables are provided for the standard sampling rates. Sampling rate must necessarily remain constant over the threshold calculation process.

The outputs of the psychoacoustic model are:

1. A set of Signal-to-Mask Ratios (SMR) and thresholds, which are to be used by the encoder.
2. The delayed time domain data (PCM samples), which are to be used by MDCT.
3. The block type (short or long) for the MDCT.
4. An estimation of the coding bits should be used for encoding. It compares to the average available bits (bit-rate).

Unlike the psychoacoustic model 1, this model does not make a dichotomous distinction between tonal and non-tonal components. Instead the spectral data is transformed to a “partition” domain and the fractions of the tonal and non-tonal components are estimated in each partition. This fraction ultimately determines the amount of masking threshold. Figure 2.6 shows the block diagram for the psychoacoustic model in the MPEG-2 AAC encoder. For more detailed procedures for calculation, please see [1].

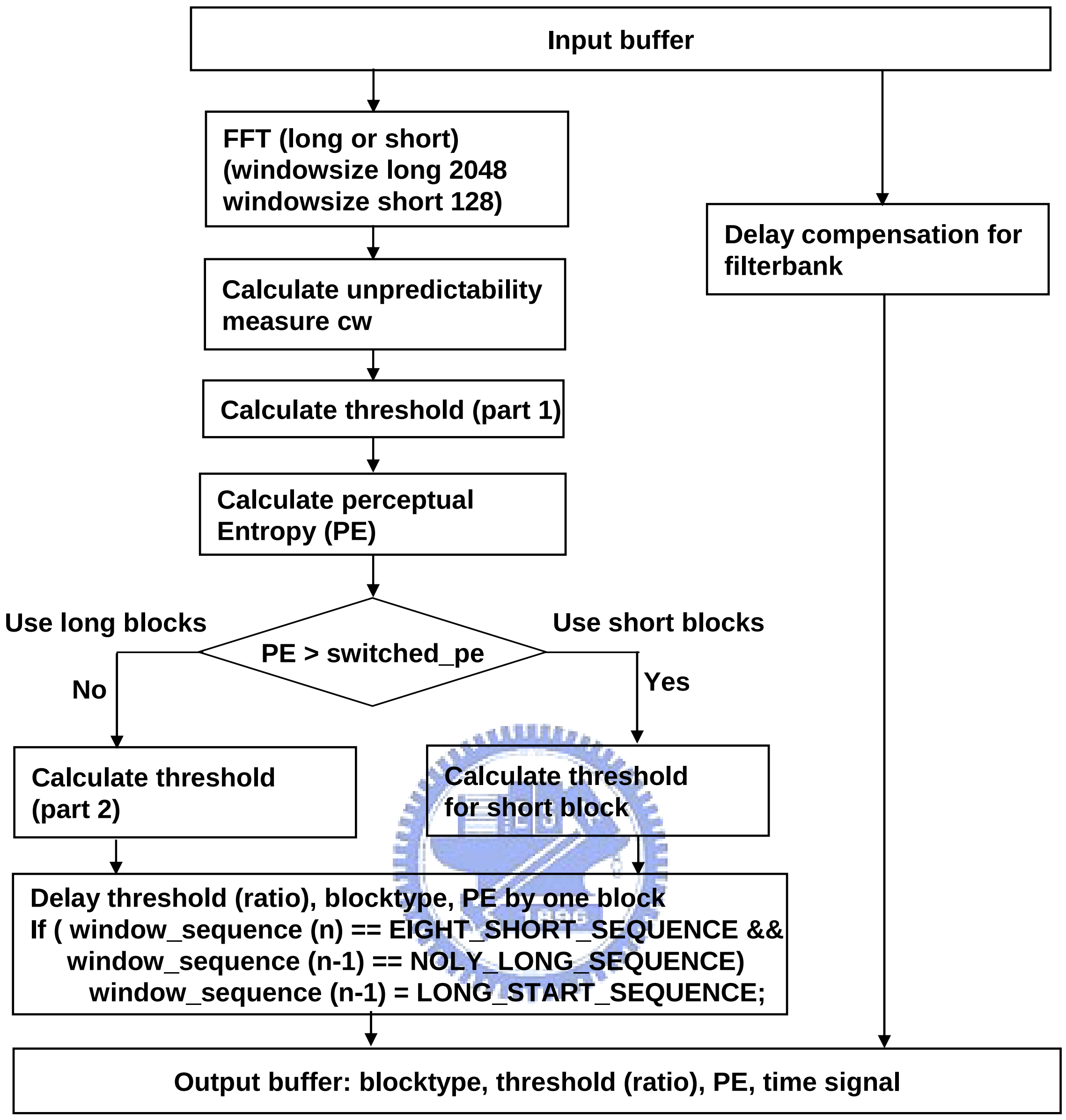

Input buffer
FFT (long or short) (windowsize long 2048 windowsize short 128)
Delay compensation for filterbank
Calculate unpredictability measure cw
Calculate threshold (part 1)
Calculate perceptual Entropy (PE)
Use long blocks
PE > switched_pe
Use short blocks
No
Yes
Calculate threshold (part 2)
Calculate threshold for short block
Delay threshold (ratio), blocktype, PE by one block
If (window_sequence (n) == EIGHT_SHORT_SEQUENCE &&
window_sequence (n-1) == NOLY_LONG_SEQUENCE)
window_sequence (n-1) = LONG_START_SEQUENCE;
Output buffer: blocktype, threshold (ratio), PE, time signal

Figure 2.6 Block diagram of psychoacoustic model [1].

2.3 Filter Bank and Block Switching

Filterbank tool transforms the time domain input samples into coefficients in frequency domain by the modified discrete cosine transformation (MDCT) technique. MDCT use the concept of subband coding. It adopts a special analytical filterbank to decompose the input data. As its name implied, analytical filterbank is a cosine multiplied with window sequence. In the MPEG-2 AAC encoder, the filterbank takes in the appropriate block of time samples, and modulates the time samples by an appropriate window function, and performs the MDCT. Each block of input samples is overlapped by 50% with the immediately preceding block and following block in order to reduce the boundary artifact. The expression is as following:

$$X_{i,k} = 2\sum_{n=0}^{N-1} x_{i,n} w(n) \cos\left[\frac{2\pi}{N}(n+n_0)(k+\frac{1}{2})\right] \ , \ k = 0,1...,\frac{N}{2}-1, \tag{2.2}$$

where

n	=	sample index,
N	=	window length of the one transform window based on the window sequence, 2048 for long window and 256 for short window,
i	=	block index,
k	=	spectral coefficient index,
n_0	=	$n_0 = \frac{N/2+1}{2}$,
$w(n)$	=	window function (KDB or Sine function).

Since the window function has a significant effect on the filterbank frequency response, the filterbank has been designed to allow a change in window length and shape to match to the input signal characteristics. There are two resolutions in AAC, one with 1024 spectral coefficients (one long window) and one with eight sets of 128 coefficients (eight short windows) and the switching between them is supported through the use of transition windows. The encoder selects the optimal shape for each of these windows between the Kaiser-Bessel-derived window (KBD) with improved far-off rejection and the sine window with a wider main lobe. Both of them provide perfect reconstruction. Figure 2.7 shows the example of window shape switching. The labeled A-B-C is the process using the KDB windows, and the labeled D-E-F is the process that the filter bank switches to sine window and then switches back to KDB window.

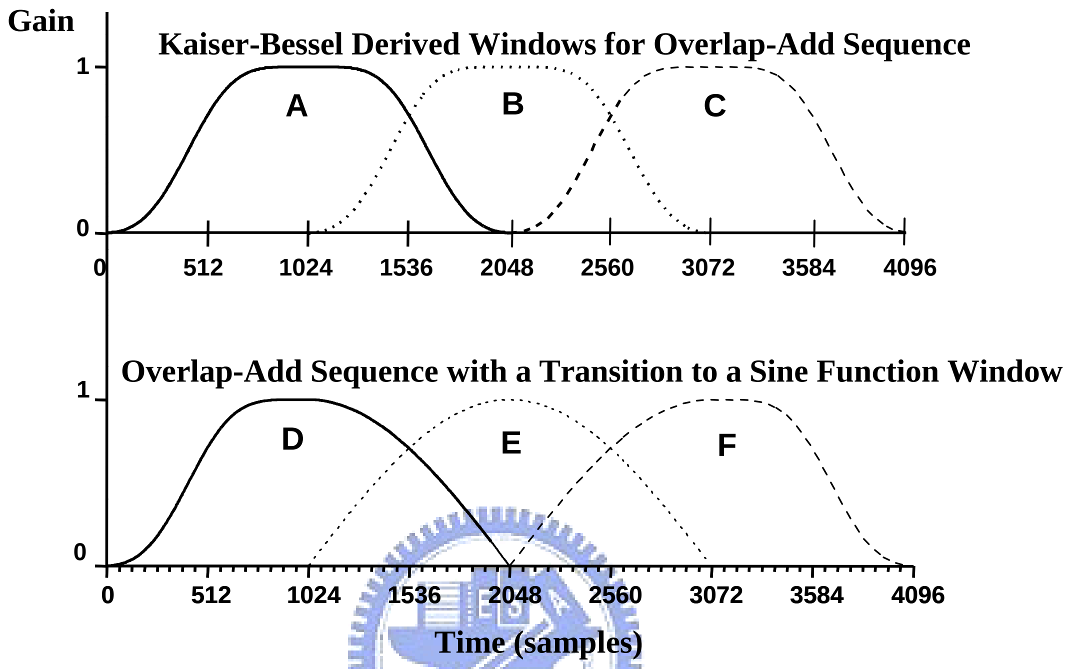
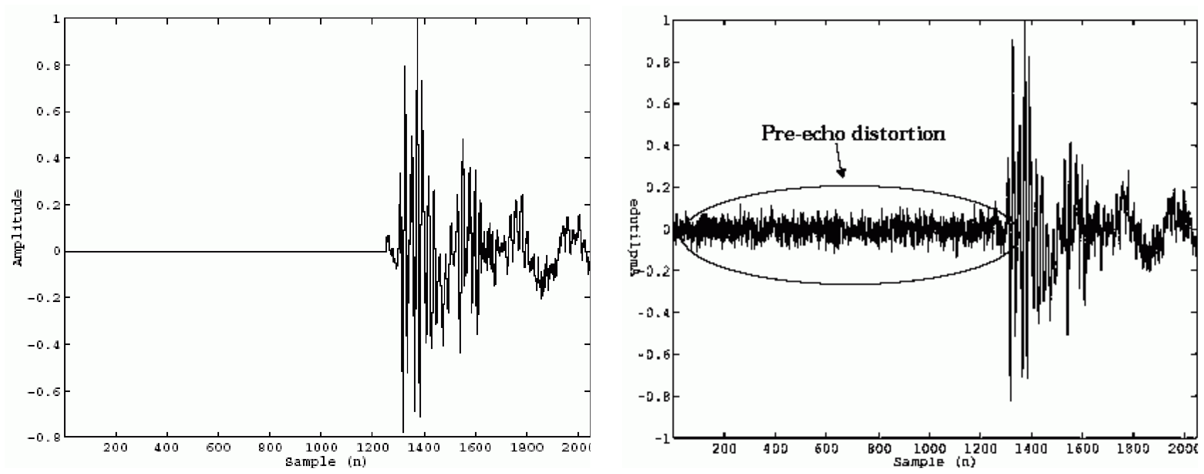


Figure 2.7 Window shape adaptation process [1].

In transform audio coding, using long window type usually provides higher coding efficiency. But this may have problem for transient signals. The quantization noise extends to the area before the occurrence of the transient signals and can not be masked by itself. This phenomenon is called pre-echo. Figure 2.8 shows the pre-echo phenomenon. AAC solves the pre-echo by switching the block length between 2048 and 256. If the psychoacoustic model detects the transient signals, the filter bank switches to short block sequence. If it detects the steady-state signals, the filter bank switches to long block sequence.



(a) (b)

Figure 2.8 Pre-echo example: (a) original wave spectrum, (b) transform spectrum, 2048-point.

Because of two different type of window length (2048 or 256), the problem of block synchrony between the different windows is occurred. In order to maintain the block alignment and the time domain aliasing cancellation properties of MDCT, the "long start" and "long stop" windows is used during the long-short window transitions. Figure 2.9 show the window overlap-add process for both steady-state and transient conditions.

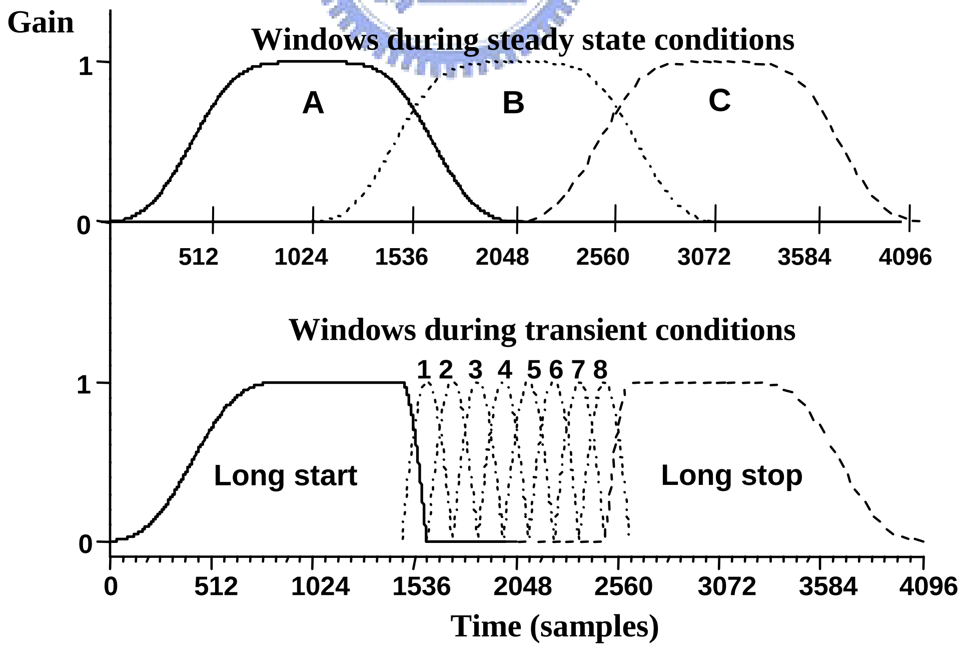


Figure 2.9 Block switching during steady-state and transient signal conditions [1].

2.4 Temporal Noise Shaping (TNS)

Pre-echo happens when a large signal rises abruptly from quiet or nearly quiet. Most codec will choose an appropriate length of the window to perform time-frequency analysis. The temporal noise shape (TNS) is used to solve pre-echo phenomenon. It is used to control the temporal shape of the quantization noise within each window of the transform. It maintains the masking effect in the reproduced audio signals.

The concept of TNS uses the duality between the time domain and frequency domain to extend linear predictive coding (LPC) techniques. The signals with an "un-flat" spectrum can be coded efficiently either by directly coding the spectral coefficients or by applying predictive coding method to the time domain signals. According to the duality property, the signals with an "un-flat" time structure, like transient signals, can be coded efficiently either by directly coding time-domain samples or applying predictive coding to the spectral coefficients. In addition, if predictive coding is applied to spectral coefficients, the temporal noise will adapt to the temporal signal when decoded. Hence the quantization noise is put into the original signal, and in this way, the problem of temporal noise in transient or pitched signals can be avoided. The tool can provide considerable enhancement to the audio quality for the speech and transient signals.

Figure 2.10 shows the block diagram of the TNS encoder filtering. Immediately after the filter bank module TNS filtering is inserted. It performs an in-place filtering operation on the spectral values, and replaces the target spectral coefficients with the prediction residual.

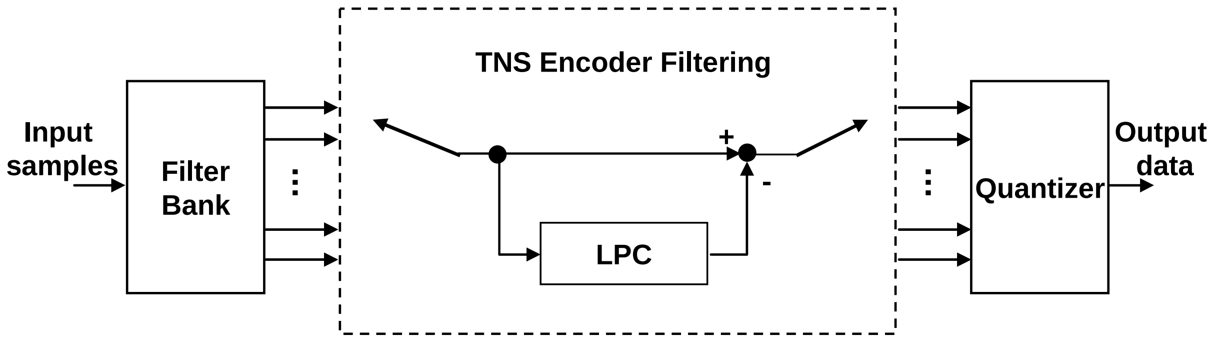


Figure 2.10 Block diagram of TNS encoder filtering.

2.5 Joint Stereo Coding

AAC joint stereo coding reduces the needed bitrate for stereo or multichannel signals more efficiently than separate coding of several channels. There are two different joint stereo methods that can be selected for coding of different frequency bands to optimize the resulting bitrate: M/S stereo coding and intensity stereo coding.

2.5.1 Middle/Side Stereo Codin

There are two different choices to code each pair of the multi-channel signals, the original left/right (L/R) signals or the transformed middle/side (M/S) signals. M/S stereo coding is very efficient for near monophonic signals, because it use a sum (M) and a difference (S) channel instead of left and right channels. The relation between L/R and M/S shows as the following expression.

$$\begin{bmatrix} m \\ s \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \times \begin{bmatrix} l \\ r \end{bmatrix}. \qquad (2.3)$$

If the left and right channel signals have high correlation, the require bits to code this signals will be less. Because the difference signals is very small. Hence in the encoder, the M/S stereo coding will operate when the left and right signals' correlation is higher than a threshold. The M/S tool transforms the L/R signals to M/S signals.

2.5.2 Intensity Coupling

The human hearing system is sensitive to low frequency signals which include amplitude and phase. It also sensitive to amplitude of high frequency signals, but insensitive to phase. The intensity coupling tool is used to exploit irrelevance between the high frequency signals of each pair of channels. For stereo channels, it does not encode all the coefficients in high frequency, and instead, it only sends coefficients of left channel. The structure of coefficients in high frequency in right channel is obtained from left channel, and an intensity value is transmitted to calculate the actual magnitude of each coefficient.

2.6 Quantization and Bit Allocation

The primary goal of the quantization is to quantize the spectral data in such a way that the quantization noise satisfies the demands of the psychoacoustic model. The bit allocation consists of two loops, the inner loop and the outer loop, which is also called rate-distortion control (R-D control). The inner loop quantizes the spectral coefficients and increases the value of global gain until fitting the bitrate requirement. The outer loop controls the distortion in each scalefactor band after the completion of inner loop. The scalefactor bands with more than allowed distortion are amplified by scalefactor. After the amplification, the outer loop calls the inner loop again.

2.6.1 Non-uniform Quantization

AAC coding scheme adopts the non-uniform quantization to quantize the MDCT coefficients. The formula of the non-uniform quantizer

$$ix(i) = \operatorname{int}\left(\left(\frac{|xr(i)|}{2^{\frac{1}{4}(stepsize_q)}}\right)^{\frac{3}{4}} + 0.4054\right). \qquad (2.4)$$

, where *xr(i)* is the MDCT coefficients, *ix(i)* is the quantized value, int(.) is the nearest integer operation and $stepsize_q$ is the quantizer stepsize of the qth scalefactor band.

$$stepsize_q = global\_gain - scalefactor_q . \qquad (2.5)$$

The stepsize of quantizer is set by *scalefactor* and *global_gain*. *Scalefactor* is used in the outer loop to scale the spectral coefficients in order to control the power of the quantization noise. *Global_gain* is used in the inner loop and is the bitrate controlling variable.

2.6.2 Bit Allocation

For reducing the required bits, bit-allocation will applied in the procedure of quantization.

The allowed distortion (masking threshold) is calculated by psychoacoustic model. The allowed distortion is distinct in each scalefactor band, so the required bits within each scalefactor band won't be the same. If the distortion does not fit the requirement of masking threshold, the spectral coefficients of the scalefactor band are amplified to obtain higher SNR. Bit-allocation is the method that dynamically allocates the bits required in each scalefactor band according to perceptual model.

2.6.3 Inner Loop

Figure 2.11 shows the flow chart of inner loop. It is the bitrate control loop. The inner loop increases the value of global gain and quantizes the spectral data. After the quantization, noiseless coding function is called to count the number of used bits to code the quantized value. If the available bits are less than the used bits, the inner loop process change the global gain and repeat the inner loop process.

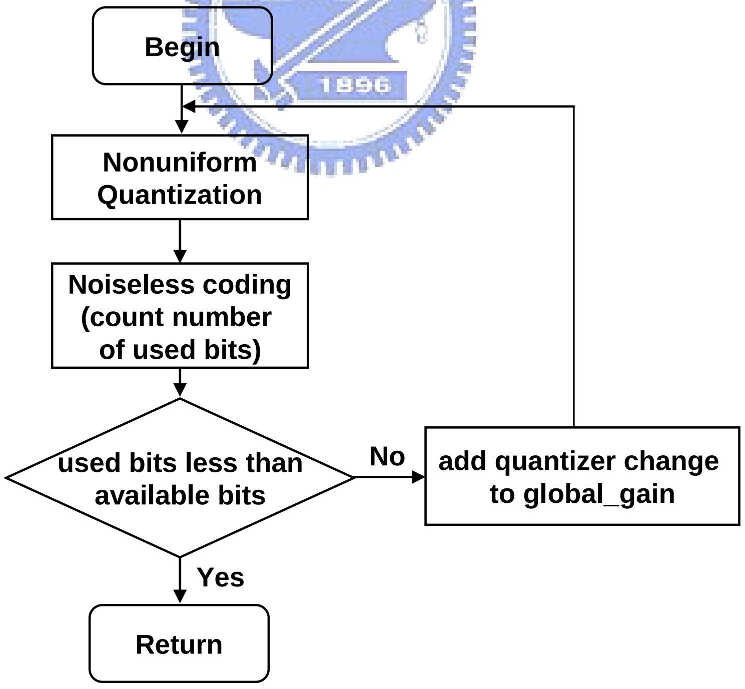

Begin
Nonuniform Quantization
Noiseless coding (count number of used bits)
used bits less than available bits
No
add quantizer change to global_gain
Yes
Return

Figure 2.11 AAC inner iteration loop [1].

2.6.4 Outer Loop

Figure 2.12 shows the flow chart of outer loop. It is distortion control loop. After inner iteration loop, the distortion in each scalefactor band is calculated by the following formula.

$$\begin{aligned} N_q &= \sum_{i \in q} \left(|xr(i)| - ix(i)^{\frac{4}{3}} \times 2^{\frac{1}{4} \times stepsize_q} \right)^2 \\ &= \sum_{i \in q} \left(|xr(i)| - ix(i)^{\frac{4}{3}} \times 2^{\frac{1}{4} \times (global\_gain - scalefactor_q)} \right)^2 \end{aligned} . \qquad (2.6)$$

, where N_q is the distortion of the qth scalefactor band, *xr(i)* is the MDCT coefficients, ix(i) is the quantized value, and $stepsize_q$ is the quantizer stepsize of the qth scalefactor band.

The task of the outer iteration loop is to amplify the scale factor bands in such a way that the demands of the psychoacoustic model are fulfilled. If the distortion is the best so far, the best scalefactor is stored. Otherwise, the scalefactor is increased to repeat the outer loop process. Normally the loops processing terminates, if there is no scalefactor band with an actual distortion above the allowed distortion. However this is not always possible to terminate the outer iteration loop by this condition. Therefore, there are two other conditions to terminate the outer iteration loop:

1. All scalefactor bands are amplified.
2. The difference between two consecutive scalefactors is greater than 60.

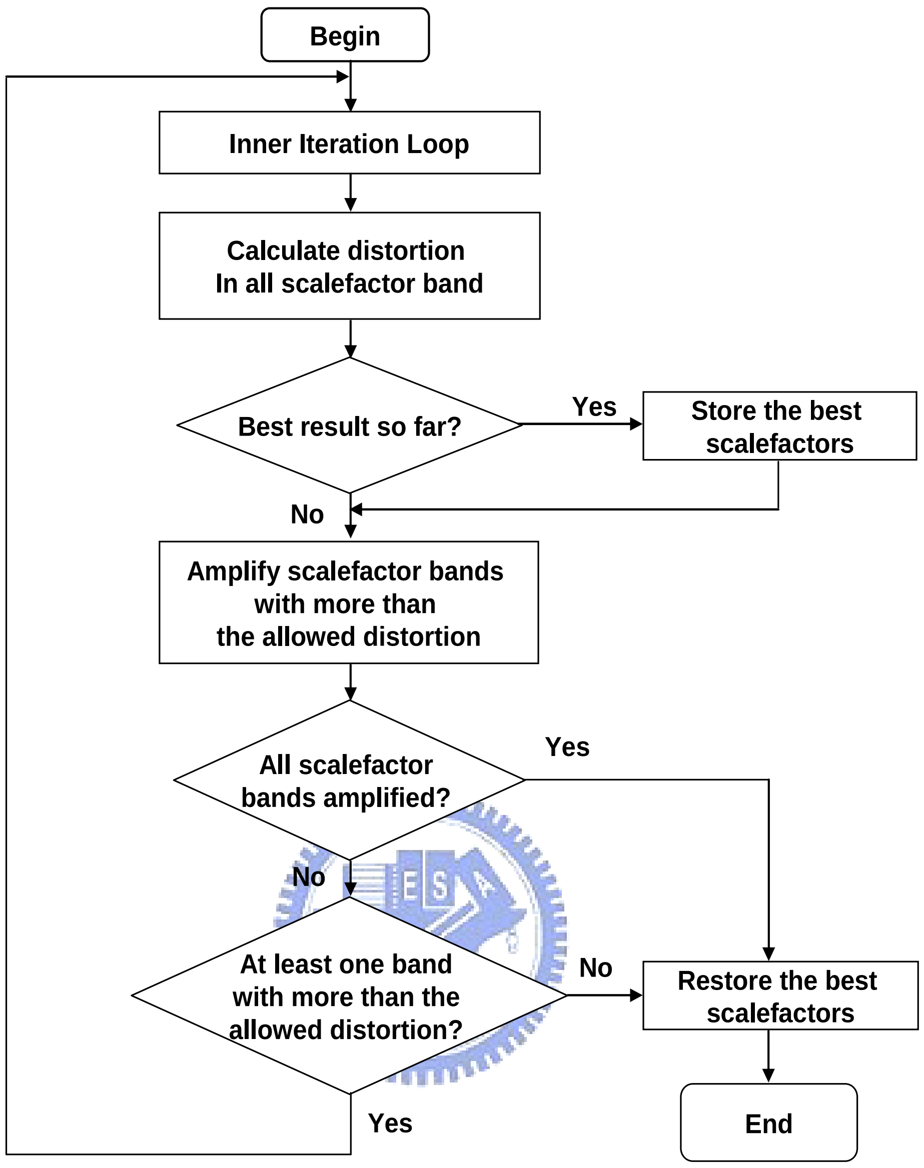

Begin
Inner Iteration Loop
Calculate distortion
In all scalefactor band
Best result so far?
Yes
Store the best scalefactors
No
Amplify scalefactor bands with more than the allowed distortion
All scalefactor bands amplified?
Yes
No
At least one band with more than the allowed distortion?
No
Restore the best scalefactors
Yes
End

Figure 2.12 AAC outer iteration loop [1].

2.7 Noiseless Coding

The input to the noiseless coding module is the set of 1024 quantized spectral coefficients. Since the noiseless coding is done inside the quantizer inner loop, it is part of an iterative process that converges when the total bit count achieves the available bit count. The noiseless coding uses sectioning and variable-length Huffman coding (entropy coding). It exploits statistical redundancy to efficiently encode the 1024 coefficients. Section technique is powerful technique by group 2 or 4 coefficients to reduce the bit-rate.

When there are eight short windows in a frame, grouping and interleaving mechanism are designed for better coding efficiency. The coefficients associated with contiguous short windows can be grouped to share scalefactors among all scalefactor bands within the group. In addition, the coefficients within a group are interleaved by interchanging the order of the scalefactor bands and windows.

In order to increase compression, scalefactors associated with the scalefactor bands that have zero-valued coefficients are ignored in the noiseless coding and do not have to be transmitted. Both the global gain and scalefactors are quantized in 1.5 dB steps. The scalefactors are normalized by the global gain. The global gain is coded as an 8-bit unsigned integer, and the scalefactors are differentially encoded relative to the previous scalefactor value.

The noiseless coding segments the set of 1024 quantized spectral coefficients, such that a single Huffman codebook is used to code each section. The Huffman coding is used to represent n-tuples of quantized coefficients, with 12 codebooks can be used. The spectral coefficients within n-tuples are ordered and the n-tuples size is two or four coefficients. Each codebook specifies the maximum absolute value that it can represent and the n-tuple size. Most codebooks represent unsigned values in order to save codebook storage.

Chapter 3
MPEG-4 High Efficient Advanced Audio Coding

In this chapter, we will introduce several basic concepts and major modules of the MPEG-4 High Efficient-AAC system and the Spectral Band Replication (SBR) tool. SBR is a unique bandwidth extension technique developed by Coding Technologies. It enables audio codec to operate at lower bit-rate without sacrificing sound quality. Details can be found in [2] and [4] respectively.

3.1 MPEG-4 High Efficient Advance Audio Coding

MPEG-4 High Efficient Advanced Audio Coding (HE-AAC) is a combination of MPEG AAC and the spectral band replication (SBR) tool. In December 2001, SBR has been submitted to MPEG and became the reference model of the MPEG-4 version 3 audio standardization process. SBR was finalized during the March 2003 MPEG meeting (14496-3:2001/Amd.1:2003). SBR is the bandwidth extension technology developed by Coding Technologies in Germany. It uses the concept that human ear is sensitive to low frequency signals but is insensitive to high frequency. At the encoder side, we encode the low frequency audio signals using regular method and the high frequency audio signals are represented by a small amount of side-information. At the decoder side, it uses the side-information to reconstruct the high frequency component of the audio signals. HE-AAC is also called aacPlus. It is able to deliver high quality audio signal at a 30% lower bit-rate with an increased complexity. It delivers good audio quality at 24 kbps for mono and 48 kbps for stereo signals. SBR is not a self-contained audio coder. It has been integrated to the different traditional audio or speech coders, such as MPEG-2/4-AAC, MPEG-Layer II and

MPEG-Layer III (mp3). Mp3Pro is the result of combining mp3 with SBR. Our audio codec, HE-AAC, is MPEG-2 AAC LC profile with SBR because of the memory consideration. Figure 3.1 shows the block diagram of SBR module and audio coder [4]. SBR acts as a pre-process to the audio encoder, and as a post-process to the core decoder. We will describe the SBR tool in the Section 3.3 , and demonstrate how this tool can achieve good coding efficiency.

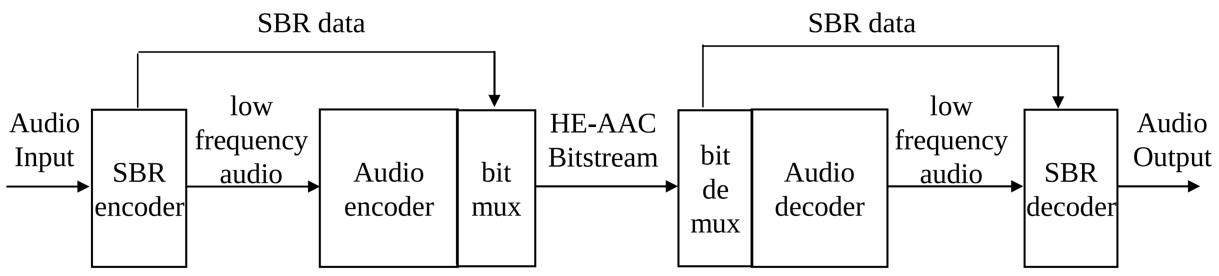


Figure 3.1 The block diagram of SBR module and audio coder [4].

3.2 Spectral Band Replication

3.2.1 Why SBR Improves Audio Coding

Research on perceptual audio coding started about twenty years ago. As a consequence the MP3 and AAC were developed with high compression efficiency. In Figure 3.2, the encoder estimates the masking threshold and tries to shape the quantization noise in the frequency domain to be lower than the masking threshold. This can achieve fine audio quality at low bitrates.

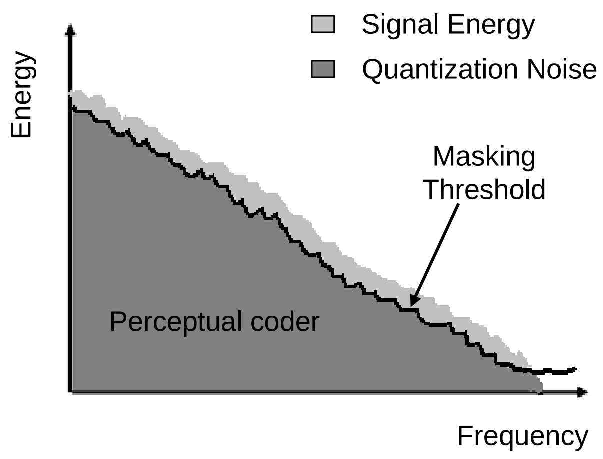
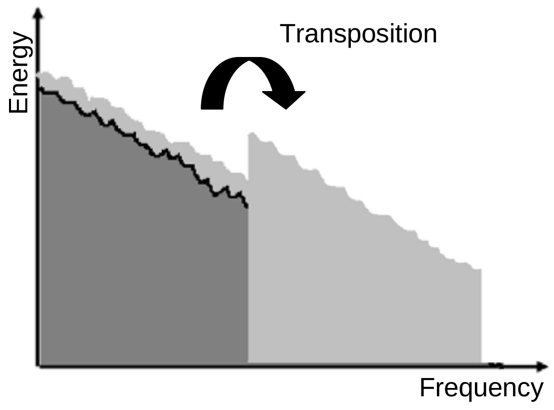


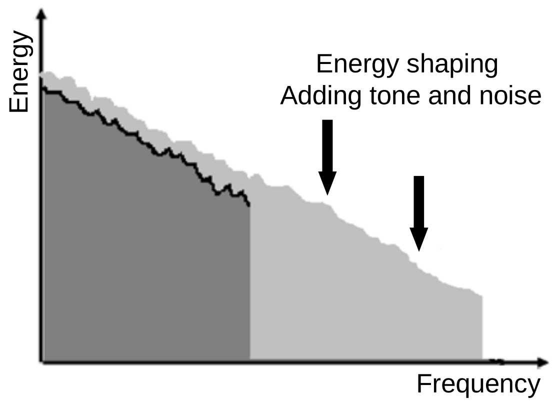
Figure 3.2 Ideal perceptual audio coding [4].

Although today’s perceptual waveform codecs already achieve good compression, the efficiency is not high enough to fulfill the bandwidth limitation for broadcasting systems and wireless systems. If the bitrate of afore mentioned audio codecs is significantly lower, the maximum distortion would exceed the masking threshold. One way to solve this problem is to limit the audio bandwidth to achieve lower bitrate. In this case, the high frequency signals are generated using a little side information. Since there is no high frequency signals to be encoded, more bits are available for encoding the remainder of the spectrum (lowband signals). HE-AAC encodes the lowband signals on encoder side, and decodes the full frequency audio signals on the decoder side with the help of the SBR technique. We thus have good audio quality on lower bitrate.

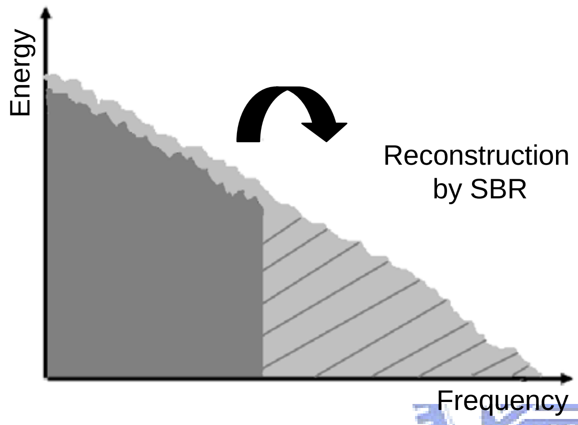
The SBR technology can be combined with any perceptual audio codec in a backward compatible way, which is shown in the Figure 3.1. It is based on the fact that there are usually high correlations between the lower and higher frequency part of audio signals. Hence, we can use lowband signals to reconstruct the highband signals. Only small amount of the side information is required to carry in the bitstream in order reconstruct of the highband signals. On the decoder side, the highband signals are reconstructed by a high quality transposition algorithm. Figure 3.3(a) shows the transposition from lowband signal to highband signal. But transposition itself is insufficient for reconstructing highband signals. It also uses the side information sent from the encoder to adjust the highband signals, such as energy envelope, inverse filtering to cancel tones, and the noise and sine addition to maintain the tonal-to-noise ratio shown in Figure 3.3(b). Figure 3.3(c) shows that high frequency reconstruction through SBR.



(a)



(b)



(c)

Figure 3.3 (a) Creation of highband by transposition. (b) Envelope adjustment of highband. (c) High frequency reconstruction through SBR.

In summary, SBR enhanced codecs perform better because:

(1) SBR allows the reconstruction of the high frequency part of signals using a small amount of side information. The high frequency signals are not encoded anymore. It results in a significant coding gain.

(2) The traditional audio codecs, such as AAC, encode the low frequency signals in which it can operate at the optimum sampling rate. However, the optimum sampling rate is usually different from the desired output sampling rate. On the other hand, the SBR decoder can convert the codec sampling rate to the desired output sampling rate.

3.2.2 How SBR Works

The SBR system is used as a dual-rate system. The SBR encoder operates at the original sampling rate, and the AAC encoder operates at half the original sampling rate. The AAC encoder just processes only the low frequency part of audio signals. It uses a downsampling filter to obtain the low frequency part of audio signals. The AAC encoder computation is lower because it processes half of input data. But the SBR encoder is complex because it uses many modules to extract the high frequency signals information. The following Section will briefly explain the SBR encoder system.

3.3 SBR Encoder

3.3.1 SBR Encoder

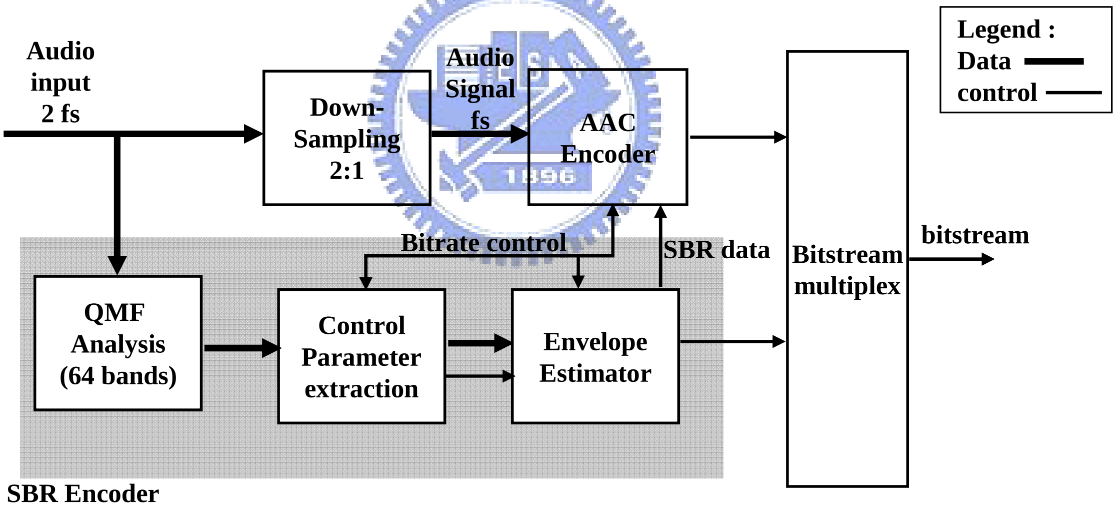


Figure 3.4 HE-AAC Encoder Overview [6].

Figure 3.4 shows the block diagram of the 3GPP HE-AAC encoder system. We can notice that the SBR encoder works in parallel with the AAC encoder. The important parameters are extracted by the SBR encoder in order to ensure an accurate high frequency reconstruction at the decoder. The input signal is fed to a 64-channel Analysis Quadrature Mirror Filter (AQMF) which will be described in Section 3.3.2 . The output from the filter banks are complex-valued subband signals. Then the complex-valued subband signals are used to choose the appropriate time/frequency resolution (T/F grid) of the current SBR frame.

The spectral envelopes of the current frame are estimated over the time segment and with the frequency resolution given by the time/frequency grid. In order to achieve optimal quality, given the high frequency generator which used in the decoder, several additional parameters apart from the spectral envelope are extracted. When the lowband signals are be transposed to the highband signals, it may have the situation that lowband constitutes a strong harmonic series but the highband constitutes random signal. Or the strong tonal components are present in the original highband but not in the lowband. To handle the inconsistence of the tonal-to-noise ratio of the original spectral bands and the replicated spectral bands, the adding of noise or sinusoids with suitable energy is considered. Then the SBR data and other parameters are coded by entropy coding (Huffman coding). SBR data and AAC data information is exchange between the system in order to determine the optimal cutoff frequency between the AAC encoder and the SBR band. Finally the HE-AAC encoder multiplexes the SBR bitstream into the AAC bitstream. Figure 3.5 shows the block diagram of the SBR Encoder. Details can be found in [14].

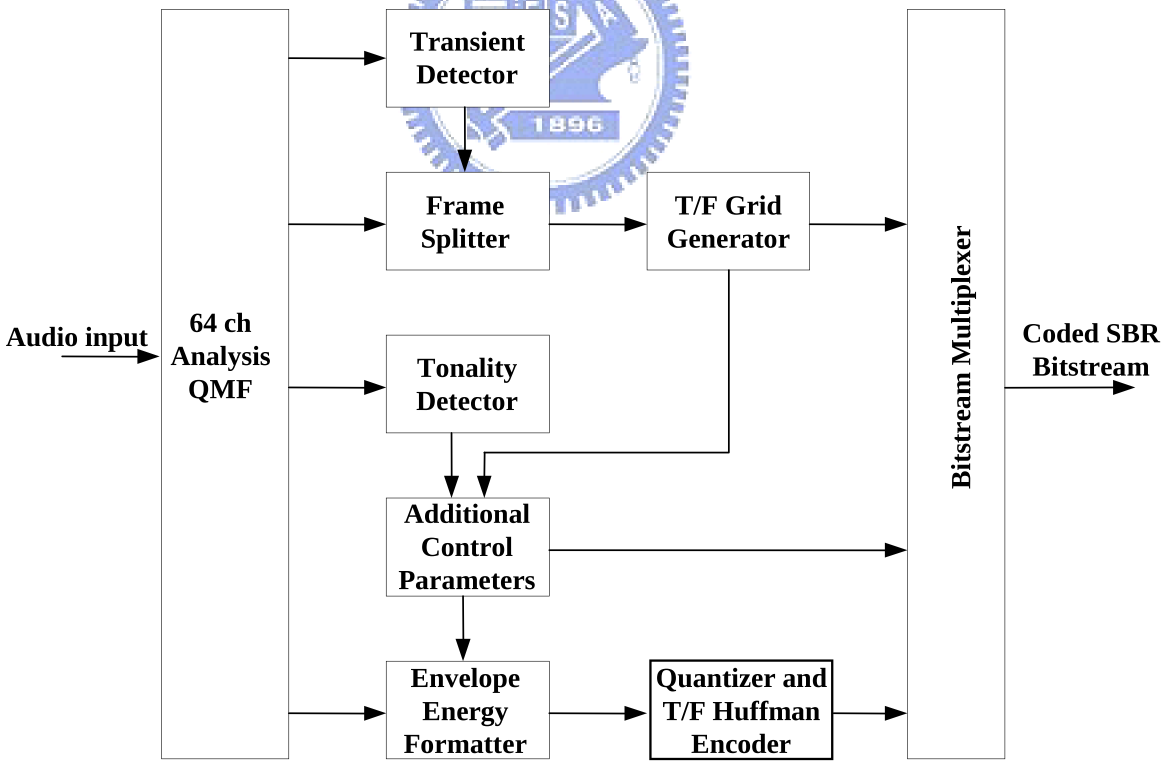


Figure 3.5 SBR encoder block diagram [14].

3.3.2 Analysis Quadrature Mirror Filter (AQMF) Bank

On the SBR encoder side, subband filtering of the input signal is done by a 64-subband QMF bank. The outputs from the filterbank are complex-valued. The filtering comprises the following steps, in which an array **x** consisting of 640 time domain input samples are assumed. Higher indices into the array correspond to older samples. Figure 3.6 shows the QMF analysis window.

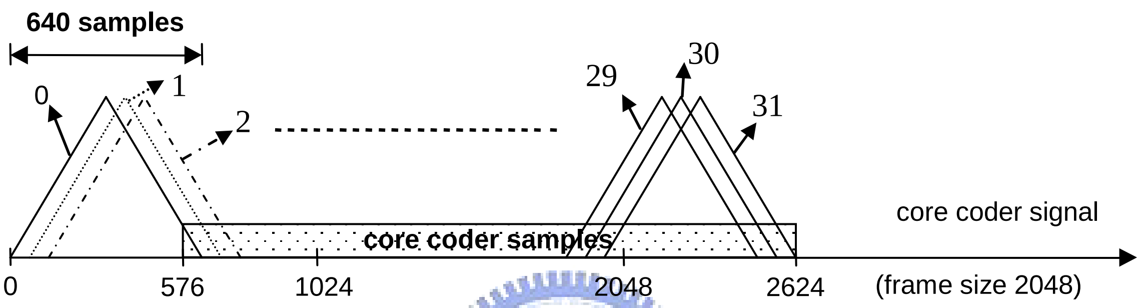


Figure 3.6 HE-AAC QMF analysis windowing [2]. Index 0 to 31 represent different window.

The QMF process is described:

1. Shift the samples in the array **x** by 64 positions. The oldest 64 samples are discarded and 64 new samples are stored in positions 0 to 63.
2. Multiplying the samples of array **x** by window **c** is array **Z** **(** $Z[n] = x[n] \times c[n]$, for n =0 to 639**)**. The 640 window coefficients (***c***) are showed in Figure 3.7.
3. Sum the samples according to the formula, $u[n] = \sum_{j=0}^{4} Z[n+128j]$, n=0 to 127, to create the 128-element array ***u***.
4. Calculate 64 new subband samples by the matrix operation ***X* = *Mu***, where

$$M(k,n) = \exp\left(\frac{i\pi(k+0.5)(2n+1)}{128}\right), \quad \begin{cases} 0 \le k < 64, \\ 0 \le n < 128. \end{cases} \tag{3.1}$$

X[k][j] corresponds to the jth subband sample QMF subband k.

In the equation, exp() denotes the complex exponential function and *i* is the imaginary unit.

Every loop produces 64 complex-valued subband samples, representing the output from one filterbank subband. For every SBR frame the filterbank produce 32 subband samples from every filterbank subband, corresponding to a time domain signal of length 2048 samples.

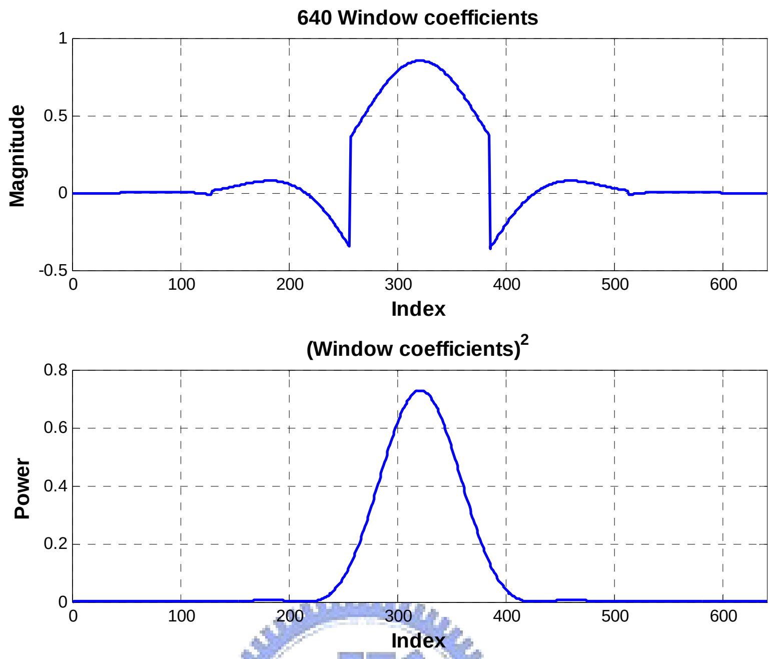


Figure 3.7 Coefficients of the QMF bank window.

3.3.3 Frequency Band Tables

On the SBR encoder side, the SBR encoder uses the following frequency band tables: a high frequency resolution table ($f_{TableHigh}$), a low frequency resolution table ($f_{TableLow}$), the noise floor frequency tables ($f_{TableNoise}$) and the master frequency band table (f_{Master}), which are defined according to subclause 4.6.18.3.2 in [2]. The parameters needed to define all frequency band tables are transmitted in the SBR bitstream header. The frequency band tables contain the frequency borders for each frequency band, represented as QMF subbands. Each frequency band is defined by a start frequency border and a stop frequency border. For SBR header bitstream elements either **bs_header_extra_1** or **bs_header_extra_1,** there are default values and a transmission of these elements are only needed if they differ from the default value. Default values are defined in subclause 4.5.2.8.1 in [1]. The SBR header parameters are regarded as tuning parameters since they are strongly bitrate and sampling frequency dependant. Throughout the tuning work for 3GPP submission several bitrate and sampling frequency dependant tunings have been created.

3.3.4 Time-Frequency Grid Generation

Information obtained from the analysis QMF bank is used to choose the appropriate time/frequency resolution of the current SBR frame. On the encoder side, the T/F grid generation algorithm calculates the start and stop time broder of the SBR envelopes and the noise floors in the current SBR frame. The T/F grid generation algorithm divides the current SBR frame into four classes, FIXFIX, FIXVAR, VARFIX and VARVAR. They use to determine the time broder of each SBR frame.

On the SBR decoder part, the T/F grid part of the bitstream payload describes the number of SBR envelopes and noise floors as well as the time segment associated with each SBR envelope and noise floor. Furthermore, it describes what frequency band tables to use for each SBR envelope. Four different SBR frame classes, FIXFIX, FIXVAR, VARFIX and VARVAR, are used, and each of which has different capabilities with respect to time-frequency grid selection. Figure 3.8 shows the example of the time-frequency grid. Detail can be found in [2], subclause 4.B.18.3.

On the SBR encoder part, the SBR encoder of 3GPP HE-AAC employs three tools for the T/F grid generation: the transient detector, the frame splitter, and the frame generator, that will be described in the following.

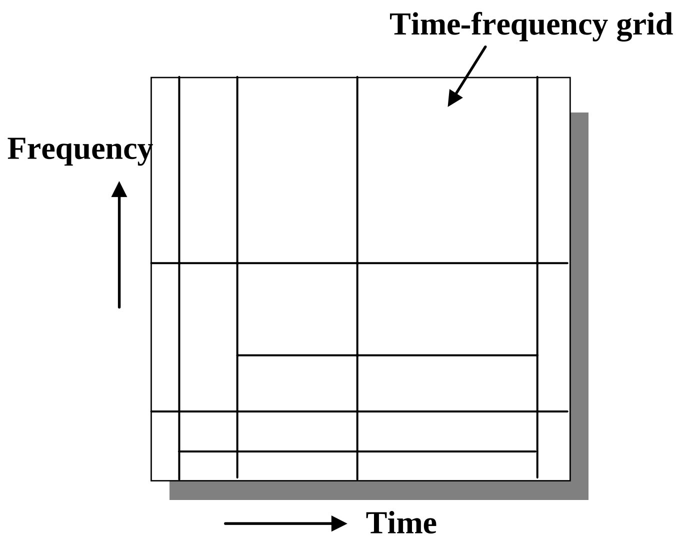


Figure 3.8 Example of the time-frequency grid

(A) Transient Detector

The transient detection is performed on subband samples of one frame length. The outputs from the transient detector are the variables *tranFlag* and *tranPos*. The first is a boolean indicating whether there is a transient in the processed frame, and the second specifies the position (in time slots) for the on-set of the transient. The time / frequency grid generation module uses the output from the transient detector and the stored transient detection output from the previous frame to perform its operations. Figure 3.9 shows the flow chart of transient detector.

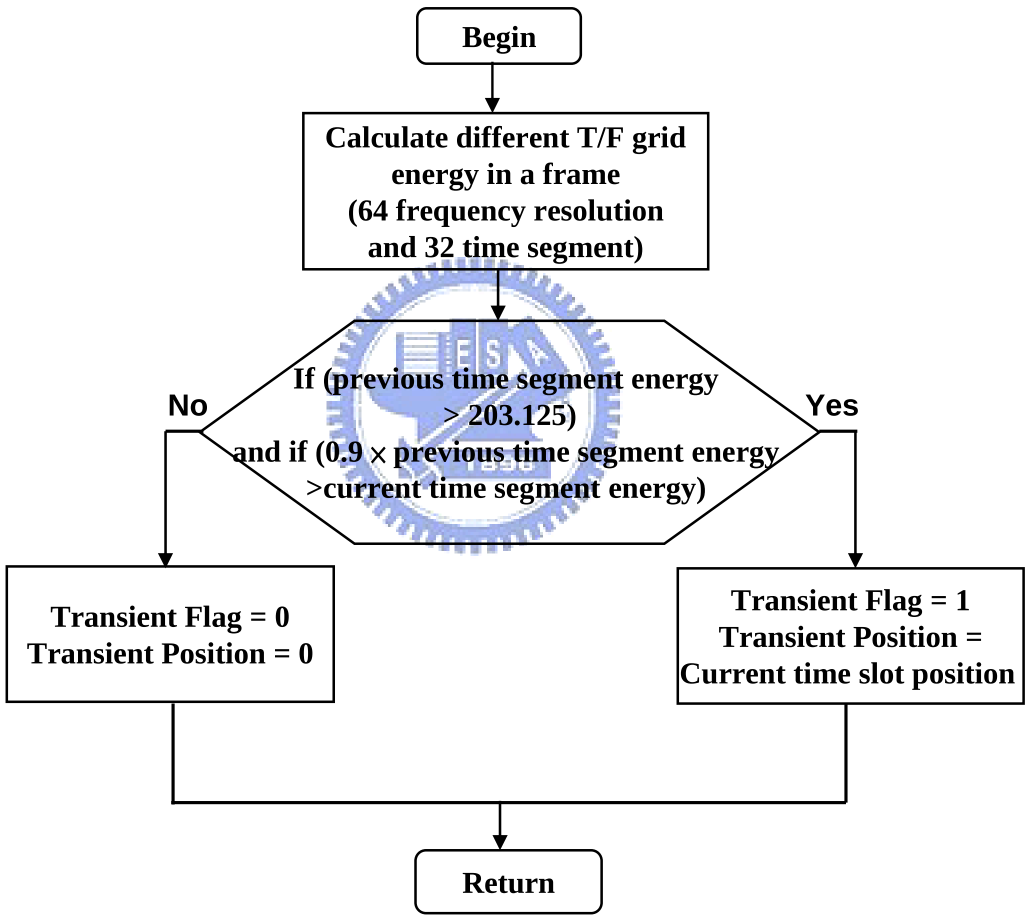
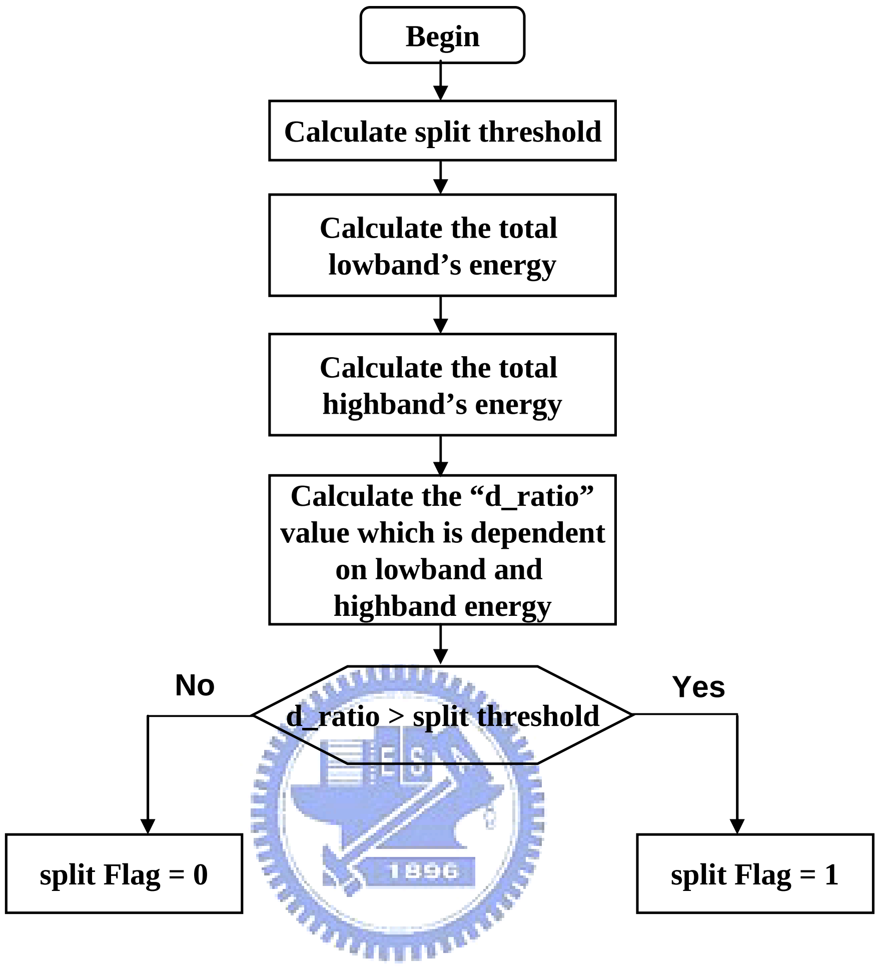

Begin
Calculate different T/F grid energy in a frame (64 frequency resolution and 32 time segment)
If (previous time segment energy > 203.125) and if (0.9 × previous time segment energy >current time segment energy)
No
Yes
Transient Flag = 0
Transient Position = 0
Transient Flag = 1
Transient Position = Current time slot position
Return

Figure 3.9 The flow chart of transient detector.

(B) Frame Splitter

The frame splitting is only active when the transient detector has detected the absence of a transient in the current frame (i.e. when transient Flag = 0). It operates on subband samples of one and a half frame length starting from subband sample 0. The output from the frame splitter is the variable splitFlag, which indicates whether the current frame (free from transients) should be divided into two envelopes of equal size. Figure 3.10 shows the flow chart of frame splitter.



Begin
Calculate split threshold
Calculate the total lowband's energy
Calculate the total highband's energy
Calculate the "d_ratio" value which is dependent on lowband and highband energy
No
d_ratio > split threshold
Yes
split Flag = 0
split Flag = 1

Figure 3.10 The flow chart of transient detector. The split threshold depends on the sampling rate and bitrate.

(C) Frame Generator

The frame generator creates the time/frequency grid for one SBR frame. Input signals are provided by the transient detector and the frame splitter. The frame generator produces two outputs: The sbr_grid() portion of the bitstream, and an internal representation of the time/frequency grid to be used by the envelope and noise floor estimators.

When no transients are present (tranFlag = 0), FIXFIX class frames are used. The frame splitter decides whether to use one or two envelopes in the FIXFIX frames (splitFlag = 0 or splitFlag = 1 respectively). "Sparse" transients (separated by one or more frames with tranFlag = 0) are coded by means of FIXVAR-VARFIX sequences. "Tight" transients (tranFlag = 1 for two or more consecutive frames) are handled by inserting VARVAR class frames. We do not show the block diagram of frame generator ,and details can be found in [14], subclause 5.4.3.

3.3.5 Envelope Estimator

On the SBR encoder side, the spectral envelopes of the current SBR frame are estimated over the time segment and with the frequency resolution given by the time/frequency grid represented by $\mathbf{t}_E$ and $\mathbf{r}$. The SBR envelope is estimated by averaging the squared complex subband samples over the given time/frequency regions.

$$\mathbf{E}(k-k_x,l)=\frac{\sum_{i=RATE\times t_E(l)}^{RATE\times t_E(l+1)-1}\sum_{j=k}^{k_h-1}\left|X(j,i)\right|^2}{\left(RATE\cdot t_E(l+1)-RATE\cdot t_E(l)\right)\cdot(k_h-k_l)},$$

$$k_l\le k<k_h,\begin{cases}k_l=\mathbf{F}\left(p,\mathbf{r}(l)\right)\\k_h=\mathbf{F}\left(p+1,\mathbf{r}(l)\right)\end{cases},0\le p<\mathbf{n}\left(\mathbf{r}(l)\right),0\le l<L_E. \tag{3.2}$$

, where RATE = 2, **F = [** $f_{TableLow}$ **,** $f_{TableLow}$ **],** L_E=Number of SBR envelopes
$\mathbf{n}=[N_{Low},N_{High}]$= Number of frequency bands for low and high frequency resolution

In the case of stereo and coupling the energy is calculated according to:

$$\mathbf{E}_{Left}\left(k-k_x,l\right)=\frac{\sum_{i=RATE\times\mathbf{t}_E(l)}^{RATE\times\mathbf{t}_E(l+1)-1}\sum_{j=k_l}^{k_h-1}\left(\left|\mathbf{X}_{Left}\left(j,i\right)\right|^2+\left|\mathbf{X}_{Right}\left(j,i\right)\right|^2\right)}{2\cdot\left(RATE\times\mathbf{t}_E\left(l+1\right)-RATE\times\mathbf{t}_E\left(l\right)\right)\times\left(k_h-k_l\right)},$$

$$\mathbf{E}_{Right}\left(k-k_x,l\right)=\frac{\varepsilon+\sum_{i=RATE\times\mathbf{t}_E(l)}^{RATE\times\mathbf{t}_E(l+1)-1}\sum_{j=k_l}^{k_h-1}\left|\mathbf{X}_{Left}\left(j,i\right)\right|^2}{\varepsilon+\sum_{i=RATE\times\mathbf{t}_E(l)}^{RATE\times\mathbf{t}_E(l+1)-1}\sum_{j=k_l}^{k_h-1}\left|\mathbf{X}_{Right}\left(j,i\right)\right|^2}, \tag{3.3}$$

For $k_l\le k<k_h,\begin{cases}k_l=\mathbf{F}\left(p,\mathbf{r}(l)\right),\\k_h=\mathbf{F}\left(p+1,\mathbf{r}(l)\right),\end{cases}\quad 0\le p<\mathbf{n}\left(\mathbf{r}(l)\right),\ 0\le l<L_E.$

3.3.6 Additional Control Parameters

In order to achieve optimal results in the decoder, several additional parameters apart from the spectral envelope are needed. The noise floor is estimated for the current SBR frame. It is defined as the ratio between the energy of the noise and the energy of the High Frequency (HF) generator signal. The energy of the noise that should be added to a particular frequency band in order to obtain a similar tonal to noise ratio.

The noise floor is estimated once or twice per SBR frame dependent on the number of spectral envelopes estimated for the SBR frame (indicated by $\mathbf{t}_Q$). The frequency resolution for the noise floor scalefactor is calculated according to the same algorithm subsequently used in the SBR decoder and described in [2] subclause 4.6.18.3. The start and stop time borders of the different noise floors are given from the time grid on the SBR encoder.

The level of the inverse filtering applied in the decoder is estimated for different frequency ranges. The inverse filtering estimation algorithm compares the original tonality and the tonality which will be produced by the High Frequency (HF) generator in the decoder. The ratio between the two is mapped to four different inverse filtering levels, off, low, mid and high. These levels correspond to different chirp factors in the HF generator as outlined in [2] subclause 4.6.18.5.

On SBR encoder side, additional control parameters include three factors: noise, inverse filtering and sine signal. The estimation noise is added in the decoder to obtain the same tonal to noise ratio. The inverse filtering is used to flat the reconstructed tonal signal on decoder side when original signal does not have tonal. The estimation sine signal is added in the decoder. If the high frequency reconstructed signals in the decoder miss the sinusoidal signal, the frequency bands will add a strong sinusoidal component. These three factors are all calculated by the output of tonal estimation. Therefore, we only describe the tonality estimation in the follows.

3.3.7 Tonality Estimation

The following detection modules produce their output based on a tonality estimate calculated in the tonality estimation module: Noise-floor estimation, Inverse filtering estimation, Additional sines estimation. These three modules are calculated by the output of the tonality estimation. Therefore, we will only describe tonality estimation module in this Section, because the complexity of the tonality estimation module is higher than the other three modules. The Noise-floor estimation, Inverse filtering estimation, Additional sines estimation modules can be found in [14], subclause 5.6.3.

The tonality is derived from the prediction gain of a second order linear prediction performed in every QMF subband. The linear predictive coding (LPC) is calculated using the covariance method, and for every frame two tonality estimates are calculated for every subband. In the equation 2.10, **X** is the matrix holding the most recently available complex QMF subband samples. The tonality values are calculated and stored in the ***T*** and ***Tsbr*** matrices. The ***Tsbr*** values are obtained from the ***T*** values by patching the tonality values similarly to the patching of the subband channels in the high frequency reconstruction modules in the decoder.

Since the subband signals are complex valued, this results in complex filter coefficients. The prediction filter coefficients are obtained from the covariance method.

$$\begin{bmatrix} \phi_x(1,1) & \phi_x(1,2) \\ \phi_x(2,1) & \phi_x(2,2) \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} \phi_x(0,1) \\ \phi_x(0,2) \end{bmatrix},$$

$$\phi_k(i,j) = \sum_{n=2}^{15} X(k,n-i)X^*(k,n-j)\ ,\ \begin{cases} 0 \le i < 3, \\ 1 \le j < 3, \end{cases} \tag{3.4}$$

where k is the subband index.

Based on the covariance elements the coefficients $\alpha_0^l(k)$ and $\alpha_1^l(k)$ used to calculate the tonality estimates for the subbands are calculated as:

$$
\begin{aligned}
&d(k) = \phi_k(2,2)\times\phi_k(1,1) - \frac{1}{1+10^{-6}}\left|\phi_k(1,2)\right|^2, \\
&\alpha_0(k) = \begin{cases} -\dfrac{\phi_k(0,1)+\alpha_1(k)\times\phi_k^*(1,2)}{\phi_k(1,1)} &, \ \phi_k(1,1)\neq 0, \\ 0 &, \ \phi_k(1,1)=0, \end{cases} \\
&\alpha_1(k) = \begin{cases} \dfrac{\phi_k(0,1)\times\phi_k(1,2)+\phi_k(0,2)\times\phi_k(1,1)}{d(k)} &, \ d(k)\neq 0, \\ 0 &, \ d(k)=0. \end{cases}
\end{aligned}
\tag{3.5}
$$

The tonality values are calculated based on the above coefficients according to:

$$
\mathrm{T}(k,l+2) = \frac{re\left\{\alpha_0(k)\times\phi_k^*(0,1)+\alpha_1(k)\times\phi_k^*(0,2)\right\}}{re\left\{\phi_k(0,0)\right\} - re\left\{\alpha_0(k)\times\phi_k^*(0,1)+\alpha_1(k)\times\phi_k^*(0,2)\right\}}, \tag{3.6}
$$

$$
\mathrm{Nrg}(l+2) = \frac{\sum_{k=0}^{N-1} re\left\{\phi_k(0,0)\right\}}{N}, \tag{3.7}
$$

where l = 0(lower half frame) or 1(upper half frame).

The tonality values are patched similarly to the patching of the QMF subbands in the decoder during high frequency reconstruction. Hence, it is possible to compare tonality of a simulated SBR signal and the original signal on the encoder side. The patch used is built in accordance to the flowchart in Figure 4.46, subclause 4.6.18.6.3 in [2], where the output variable *numPatches* is an integer value specifying the number of patches. ***patchStartSubband*** and ***patchNumSubbands*** are vectors holding the data output from the patch decision algorithm. Hence, the tonality values for the SBR part is obtained according to:

$$\mathrm{Tsbr}(k, l+2) = \mathrm{T}(p, l+2), \quad \begin{cases} k = k_x + x + \sum_{q=0}^{i-1} \mathrm{patchNumSubbands}(q)\,, \\ p = \mathrm{patchStartSubband}\,(i) + \mathrm{x}\,, \end{cases} \tag{3.8}$$

where $0 \le \mathrm{x} < \mathrm{patchNumSubbands}(i)$,$0 \le \mathrm{i} < \mathrm{numPatches}$,

l= 0(upper half frame) or 1(upper half frame).

Chapter 4
DSP Implementation Environment

We select the DSP platform to implement the MPEG-4 HE-AAC encoder. Our DSP baseboard (SMT395) is made by Sundance which houses Texas Instruments' TMS320C6416T DSP chip and Xilinx Virtex-II Pro FPGA. In this chapter, our discussion will concentrate on the DSP system development environment, DSP chip and its features because our implementation is software-based on the DSP. Then, the software development tool, Code Composer Studio (CCS), is introduced. At the end, some important acceleration techniques and features which can reduce stalls or hazards on DSP system are also included.

4.1 DSP Baseboard (SMT395)

The block diagram of the Sundance DSP baseboard system (SMT395) is shown in Figure 4.1 [24]. SMT395 utilizes the signal processing technology to provide extreme processing flexibility and high performance. SMT395 has some features which are shown below.

- 1GHz TMS320C6416T fixed point DSP processor with L1, L2 cache and SDRAM.
- 8000MIPS peak performance.
- Xilinx Virtex-II Pro FPGA. XC2V920-6 in FF896 package.
- Two Sundance High Speed Bus (100MHz, 200MHz) ports which is 32 bits wide.
- Eight 2Gbit/sec Rocket Serial Links(RSL) for interModule.
- 8 MB flash ROM for configuration and booting.
- Six common ports up to 20 MB per second for inter DSP communication.
- JTAG diagnostics port.

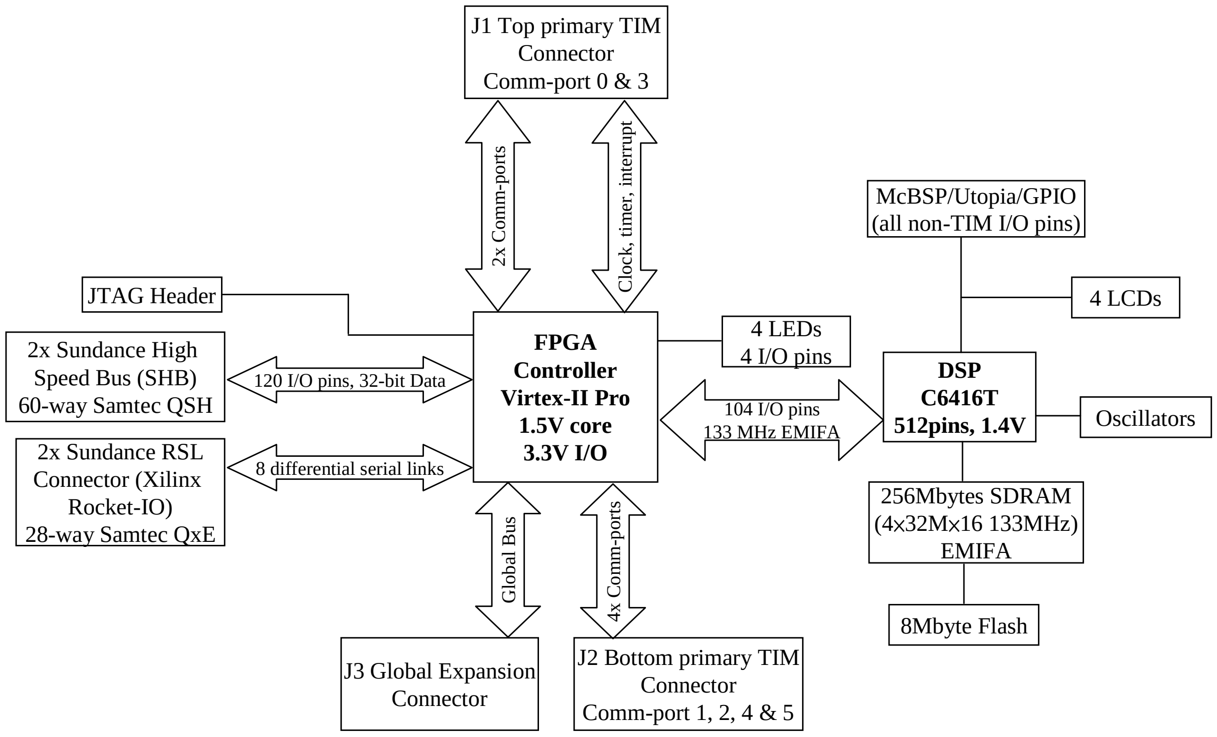

J1 Top primary TIM Connector Comm-port 0 & 3
2x Comm-ports
Clock, timer, interrupt
McBSP/Utopia/GPIO (all non-TIM I/O pins)
JTAG Header
4 LCDs
2x Sundance High Speed Bus (SHB) 60-way Samtec QSH
120 I/O pins, 32-bit Data
FPGA Controller Virtex-II Pro 1.5V core 3.3V I/O
4 LEDs 4 I/O pins
104 I/O pins 133 MHz EMIFA
DSP C6416T 512pins, 1.4V
Oscillators
2x Sundance RSL Connector (Xilinx Rocket-IO) 28-way Samtec QxE
8 differential serial links
256Mbytes SDRAM (4x32Mx16 133MHz) EMIFA
Global Bus
4x Comm-ports
8Mbyte Flash
J3 Global Expansion Connector
J2 Bottom primary TIM Connector Comm-port 1, 2, 4 & 5

Figure 4.1 The block diagram of the Sundance DSP Baseboard system

4.2 DSP Chip

TMS320C6416T DSP is using the VelociTI.2 architecture[25]. VelociTI.2 is a high performance, advanced very long instruction word (VLIW) architecture, which is an excellent choice for multi-channel, multi-functional, and performance-driven applications. VLIW architecture can achieve high performance through increased instruction-level parallelism, perform multiple instructions during a single cycle.

The DSP chip we adopt is one in the TMS320C64x series. According to [11], TMS320C64x series is also a member of the TMS320C6000 (C6x) family. The block diagram of the C6000 family is shown in Figure 4.2. The C6000 device is capable of executing up to eight 32-bit instructions per cycle. The detailed features of the C6000 family devices include:

- Advanced VLIW DSP core
- Eight independent functional units, including two multipliers and six arithmetic units (ALU).
- 64 32-bit general-purpose registers

- Instruction packing to reduce code size, program fetches, and power consumption.
- Conditional execution of all instructions.
- Non-aligned Load and Store architecture
- Byte-addressable (8/16/32/64-bit data), providing efficient memory support for a variety applications.
- 8 bit overflow protection

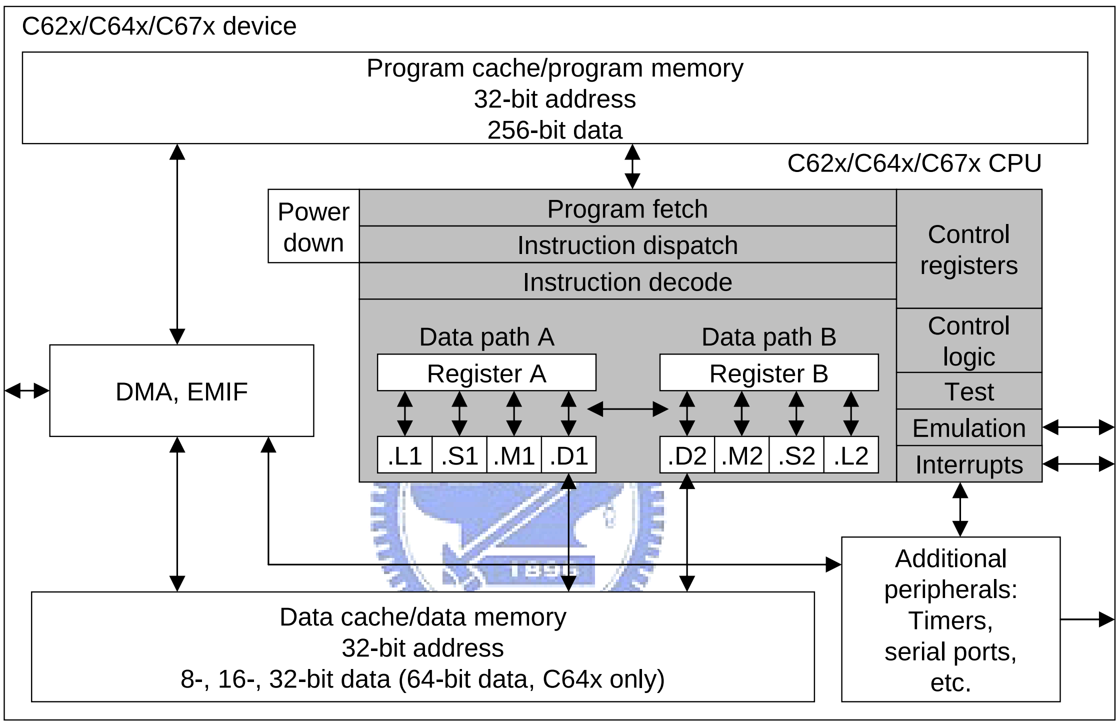


Figure 4.2 Block diagram of TMS320C6x DSP

Peripherals such as enhanced direct memory access (EDMA) controller, power-down logic, and two external memory interfaces (EMIFs) usually come with the CPU, while peripherals such as serial ports and host ports are on only certain devices. In the following Sections, C64x DSP chip is introduced further in the manner of three major parts: central processing unit (CPU), memory, and peripherals.

4.2.1 Central Processing Unit (CPU)

Besides the eight independent functional units and sixty-four general purpose registers that has been mentioned before, the C64x CPU consists of the program fetch unit, instruction dispatch unit, instruction decode unit, two data path (A and B, each with four functional units), interrupt logic, several control registers and two register files (A and B with respect to the two data paths). The DSP chip architecture is illustrated in Figure 4.3.

The instruction dispatch and decode units could also decode and arrange the eight instructions to eight functional units. The eight functional units in the C64x architecture could be further divided into two data paths, A and B as shown in Figure 4.3. Each path has one unit for multiplication operations (.M), one for logical and arithmetic operations (.L), one for branch, bit manipulation, and arithmetic operations (.S), and one for loading/storing, address calculation and arithmetic operations (.D). The .S and .L units are for arithmetic, logical, and branch instructions. All data transfers make use of the .D units. Two cross-paths (1x and 2x) allow functional units from one data path to access a 32-bit operand to the register file on another side. There are 32 general purpose registers, but some of them are reserved for specific addressing or are used for conditional instructions. All functional units which end in 1 (for example, .L1) write to register file A while all functional units which end in 2 (for example, .L2) write to register file B.

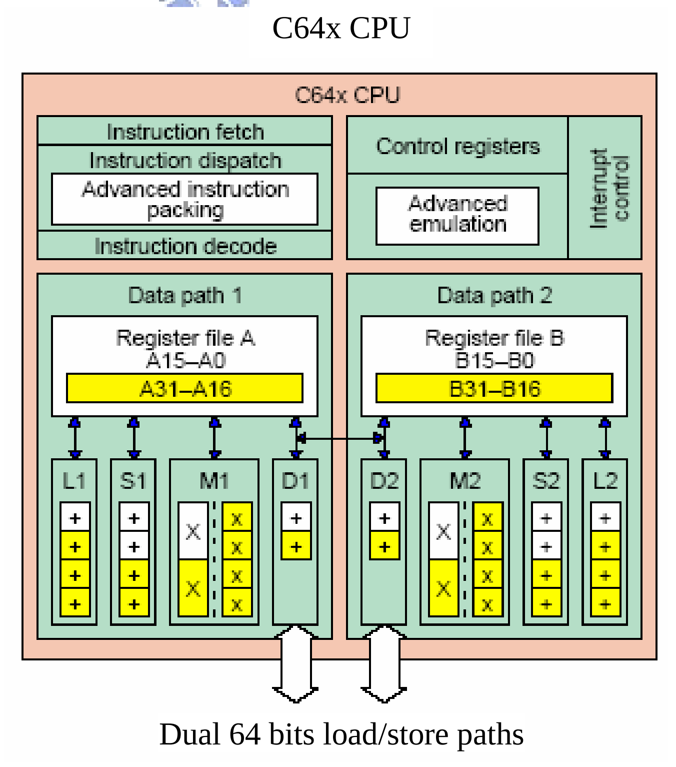


Figure 4.3 C64x DSP chip architecture

4.2.2 Memory and Peripherals

(A) Cache / Memory

The C6416T memory architecture consists of a two-level (L1/L2) internal cache-based memory architecture plus an external memory. Level 1 cache is split into L1 program cache and L1 data cache. Each of L1 cache is 16 kB. All caches and data paths are automatically managed by cache controller. Level 1 cache is accessed by the CPU without stalls. The size of L2 cache is 1 MB. L2 cache is configurable and can be split into L2 SRAM(addressable on-chip memory) and it is for caching external memory location. It also has one external memory which is a 256 MB SDRAM and operated at 133MHz.

(B) Peripherals

C64x DSP chips also contain some peripherals for supporting with off-chip memory options, co-processors, host processors, and serial devices. The peripherals are enhanced direct memory access (EDMA) controller, Host-Port interface (HPI), three 32-bit general purpose timers, IEEE-1149.1 JATG interface and some other units.

The DMA controller transfers data between regions in the memory map without the intervention by CPU. It could move the data from internal memory to external memory or from internal peripherals to external devices. It is used for communication to other devices.

The Host-Port Interface (HPI) is a 16/32-bit wide parallel port through which a host processor could directly access the CPUs memory space. It is used for communication between the host PC and the target DSP.

The C64x has three 32-bit general-purpose timers that are used to time events, count events, generate pulses, interrupt the CPU and send synchronization events to the DMA controller. The timer has two signaling modes and could be clocked by an internal or an external source.

4.3 TI DSP Code Development Environment

In this Section, the CCS tool is introduced, and we show that how a programmer can use it. Then, the code development flow is presented to show how to program a DSP code efficiently.

4.3.1 Code Composer Studio (CCS)

The Code Composer Studio (CCS) is a software integrated development environment (IDE) for building and debugging programs. The CCS extends DSP code development tools by integrating editor, debugger, simulator, and emulation analysis into one entity. We use CCS to develop and debug the projects. We briefly describe some of its features below. The details can be found in [21].

- Real time analysis
- Chip support libraries (CSL) to simplify device configuration.
- Provide debug options such as step over, step in, step out, run free.
- Compile codes and generate Common Object File Format (COFF) output file.
- Support optimized DSP functions such as FFT, filtering, convolution.
- Count the instruction cycles between successive profile-points.
- Arrange code/data to different memory space by linker command file

We mainly use the CCS tool for debugging, refining, optimizing, and implementing our C codes on DSP. The profile function helps us to determine whether the modifications of the codes are better or not. Figure 4.4 shows the software development flow.

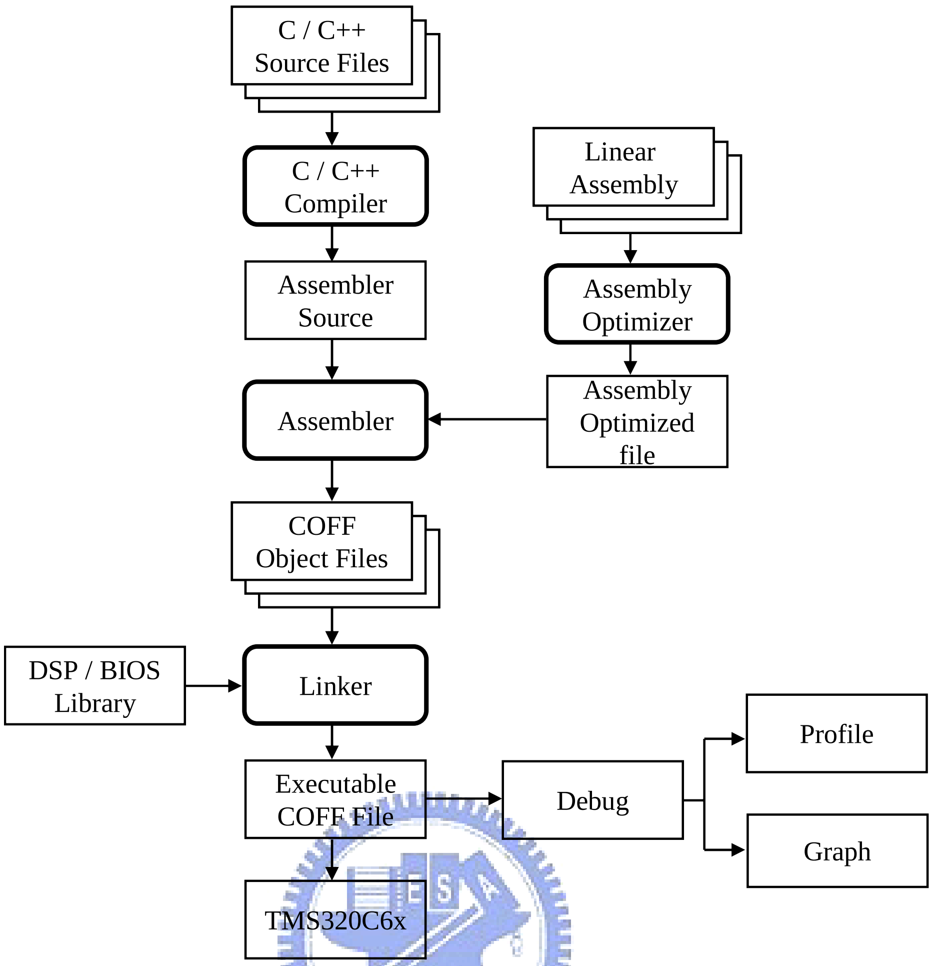


Figure 4.4 Software Development Tool Flow

4.3.2 Code Development Flow

Figure 4.5 illustrates the three phases in the code development flow [21]. Generally, we do not go to phase3 because the linear assembly will be too detailed. The recommended code development flow involves utilizing the C6000 code generation tools to aid optimization rather than forcing the programmer to write the code in assembly.

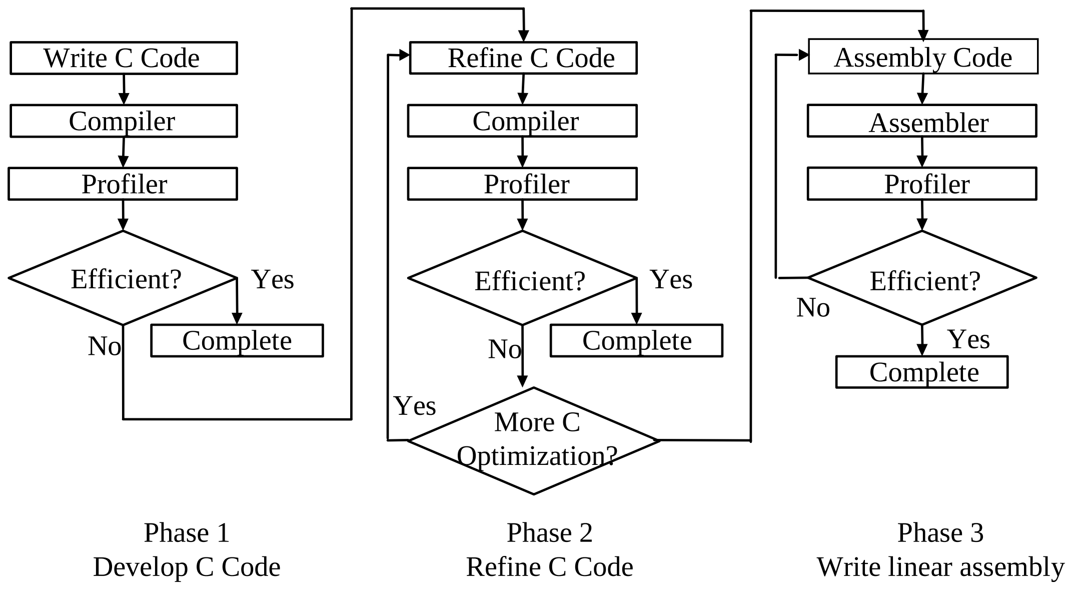


Figure 4.5 Code develop flow

4.4 DSP Code Acceleration Methods

Improving the execution cycles of the HE-AAC encoder is the main task of our system implementation. In this Section, we will describe several methods that can accelerate our code and reduce the execution time on the C64x DSP. Some of these methods are supported by the features of C64x DSP system.

4.4.1 Compiler Optimization Options

The CCS compiler offers high language support by transforming C code into more efficient assembly code. The compiler options can be used to reduce code size and improve executing time. Four optimization levels are provided: register (-o0), local (-o1), function (-o2), file level (-o3). File level (-o3) is the highest one of optimization available. With file level optimization, all our source files are compiled into one intermediate file giving the compiler complete program view during compilation. Various loop optimizations are performed, such as software pipelining, unrolling, and SIMD. It also reduces code size like: eliminating unused assignments, eliminating local and global common sub-expressions, and removing functions that are never called.

4.4.2 Fixed-point Coding

The C6000 compiler defines a size for each data type:

Table 4.1 The Size of Different Data Type.

Data Type	Char	Short	Int	Long	Float	Double
Size (bits)	8	16	32	40	32	64

The C64x DSP is a fixed-point processor, so it can only perform fixed-point operations. Although the C64x DSP can simulate floating-point processing, it takes a lot of extra clock cycles to do the same job. The “char”, “short”, “int” and “long” are the fixed-point data types, and the “float” and “double” are the floating-point data types.

4.4.3 Loop Unrolling and Packet Data Processing

Loop unrolling unrolls the loops so that all iterations of the loop appear in the code. It often increases the number of instructions available to execute in parallel. It is also suitable for use software pipeline. When our codes have conditional instructions, sometimes the compiler may not be sure that the branch will occur or not. It needs more waiting time for the decision of branch operation. If we do loop unrolling, some of the overhead for branching instruction will be reduced. Example 4.1 is the loop unrolling and Table 4.2 shows the cycles and code size.

(a)	(b)

/*Before unrolling*/
 	
int i,a=0,b=0;
for (i=0;i<8;i++)
{
a+=i;
b+=i;
}
	/*After unrolling*/
	
	int i=0,a=0,b=0;
	a+=i; b+=i; i++;
	a+=i; b+=i; i++;
	a+=i; b+=i; i++;
	a+=i; b+=i; i++;
	a+=i; b+=i; i++;
	a+=i; b+=i; i++;
	a+=i; b+=i; i++;
	a+=i; b+=i; i++;

Example 4.1 Loop unrolling.

Table 4.2 Comparison between Rolling and Unrolling

	(a)Before Unrolling	(b)After Unrolling
Execution Cycles	436	206
Code Size	116	479

We can see clearly that the clock cycle decreases after loop unrolling, but the code size is larger than the original.

Use a single load or store instruction to access multiple data that consecutively located in memory in order to maximize data throughput. It is so called the single instruction multiple data (SIMD) method. For example, if we can place four 8-bit data (char) or two 16-bit data (short) in a 32-bit space, we may do four or two operations in one clock cycle. If we use the SIMD method, then we can improve the code efficiency substantially. Some intrinsic functions enhance the efficiency in a similar way.

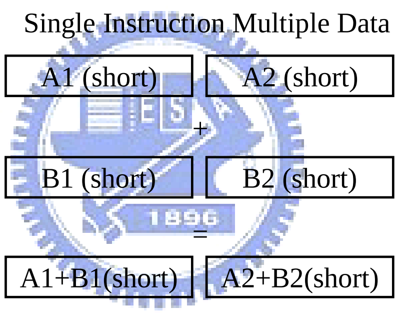


Figure 4.6 The block diagram of SIMD example. Use the word access for adding short data.

4.4.4 Register and Memory Arrangement

When the accessed data are located in the external memory, it may need more clock cycles to transfer data to CPU. We can use registers to store data in order to reduce transfer time in operation. In DSP code, the variables, pointer, malloc functions, C codes and so on will locate data in memory. We can arrange the link.cmd file which is the memory allocation file. We arrange different type of data in different memory space because of acceleration consideration. It also provide the “CODE_SECTION”, ”DATA_SECTION” key words which can allocate parts of C code or data in the internal memory in order to speed.

4.4.5 Macros Function and Intrinsic Function

Because the software-pipelined can not contain function calls, it takes more clock cycles to complete the function call. Changing the functions to the macro functions under some conditions is a good way for optimization. In addition, replacing the functions with the macro functions can cut down the code for initial function definition and reduce the number of branches. But macro functions are expanded each time when they are called, they increase the code size.

TI C6000 compiler provides many special functions that map C codes directly to inlined C64x instructions, to increase C code efficiently. These special functions are called intrinsic functions. If the instructions have equivalent intrinsic functions, we can replace them by intrinsic functions directly and the execution time will be decreased. The details of the intrinsic functions can be found in [21].

4.4.6 Linear Assembly

Assembly code is generated from CCS compiler or assembler optimizer. Sometimes the generated assembly codes are not efficiency because of a lot of stalls or hazards. Converting parts of the C codes into linear assembly codes is a good way to solve this problem. We rearrange the assembly codes to avoid the stalls and hazards by hand. Linear assembly codes can be more efficiency. But this process generally is too detail and very time consumption in practice. Hence, we will do this process at last if we have strict constrains in processor performance.

4.4.7 Other Acceleration Rules

Other rules like: reduce times for accessing memory, use bit shift for multiplication or division, declare variable or memory as constant, access the memory sequentially, and do not use conditional break or complex condition code in the loop.

Chapter 5
MPEG-4 HE-AAC Encoder Acceleration on DSP

In this chapter, we present several acceleration methods for the HE-AAC encoder. We adopt the HE-AAC source code provided by 3GPP [9]. It is the fixed-point version and thus is suitable for porting to an embedded system. Firstly, we analyze the complexity of the HE-AAC encoder, and then we determine which parts are required to accelerate. Then, we propose several fast and simplified methods to reduce the computational time. Finally, the speed improvement, RAM requirement and the audio quality due to the proposed modifications will be discussed.

5.1 HE-AAC Complexity Analysis

We use the following methods to measure the speed of the HE-AAC code:

- Use the profile mode of the stand-alone C6416T DSP simulator.
- If we are interested in only one or two functions or a region of code inside a function, the clock( ) function can be used to time the region specified.

For the purpose to find which parts take the most computational time, we choose the first method to compare all the function speeds in the HE-AAC program. We identify the parts of the HE-AAC encoder that consume the most execution time based on the profile data. We concentrate on the most critical area to accelerate. The complexity profiling of the HE-AAC encoder is shown in Figure 5.1. The test audio sequence is “glockenspiel”, which is a two channel sequence with a sample rate at 48k Hz. The bitrate is at 48k bps. It is extracted from European Broadcasting Union (EBU) [28].

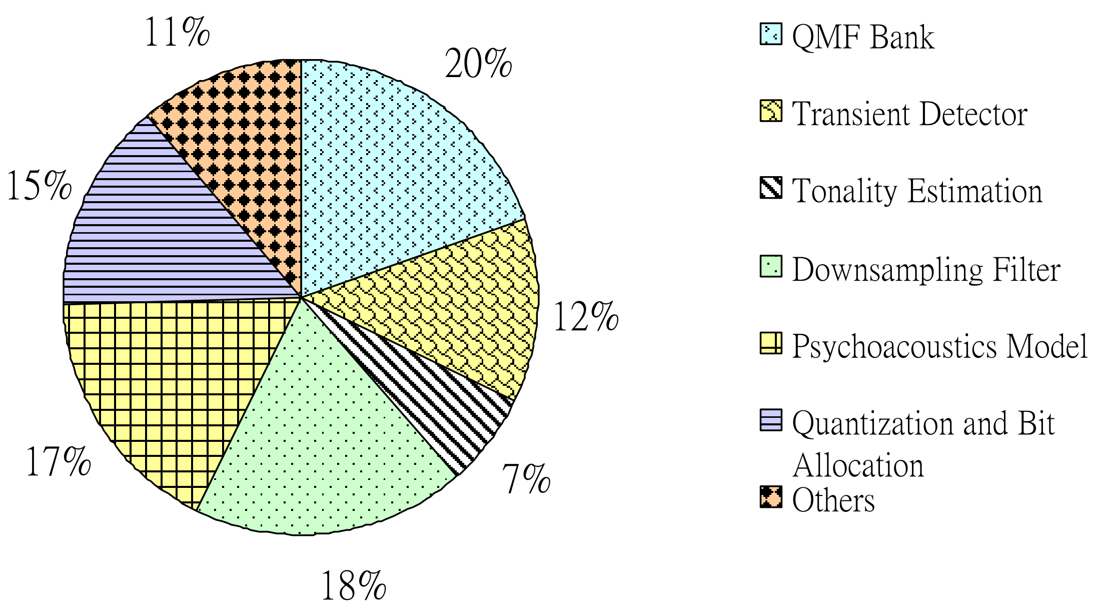


Figure 5.1 The complexity profiling of the HE-AAC encoder on the C6416T DSP.

From the profiling results, QMF bank, transient detector, tonality estimation, down-sampling filter, psychoacoustics model and quantization are the major complex parts of the HE-AAC encoder. The QMF bank, transient detector and tonality estimation are the modules related to the SBR. The psychoacoustics model and quantization are the modules only related to the original AAC. We propose several complexity reduction techniques in the following Sections. Our goal is to reduce the codec complexity while not to degrade its output audio quality.

We shortly describe the five audio sequences [28] to be used in the following Sections. These audio sequences are CD-quality sampled at 48k Hz with 16 bits for stereo, and codec at bitrate 48k bps. Table 5.1 shows the audio sequence and their characteristics. All of the compiler optimization level configure the “File” level (-o3).

Table 5.1 Test Audio Files and Their Characteristics.

Audio File	Time (sec)	Mode	Description
Gspi35_2	9	stereo	Glockenspiel. Melodious phrase.
vibr37	4	stereo	Vibraphone. Melodious phrase.
sopr44_1	12	stereo	Soprano. Vocal.
guit58	13	stereo	Guitar. Solo instrument.
edra70	14	stereo	Eddie rabbitt. Pop music.

5.2 Transient Detector Acceleration

As we mentioned in Sections 3.3.3 and 3.3.4 , the frequency band tables determine the AAC range and SBR range, and the time-frequency grid generation module uses the output from the transient detector to determine the time segment. The transient detector detects the transients and identifies the positions of the transients in each frame. However, from the profiling data, the transient detector is complex and time consuming. It calculates the energy of every gird in one frame in order to find out where the transient is, and the energy calculation in each frame is very time consuming. Therefore, after studying the transient detector algorithm, we propose an acceleration method to speed up the transient detector. Figure 5.2 and 5.3 show the spectrums of one audio frame. Figure 5.2 is the spectrum of the original audio, and Figure 5.3 is the spectrum of the HE-AAC compressed frame.

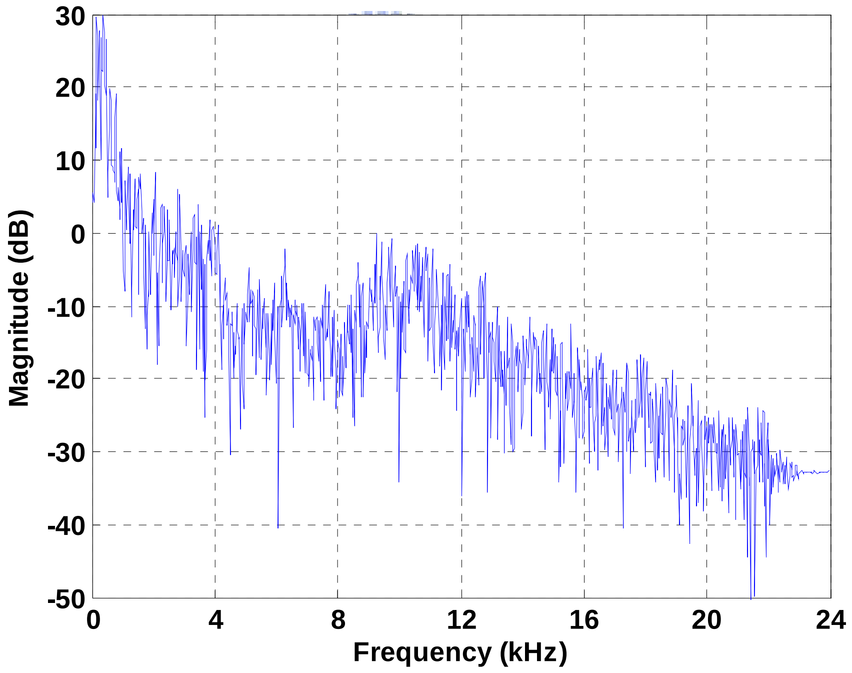


Figure 5.2 The spectrum of the original audio frame

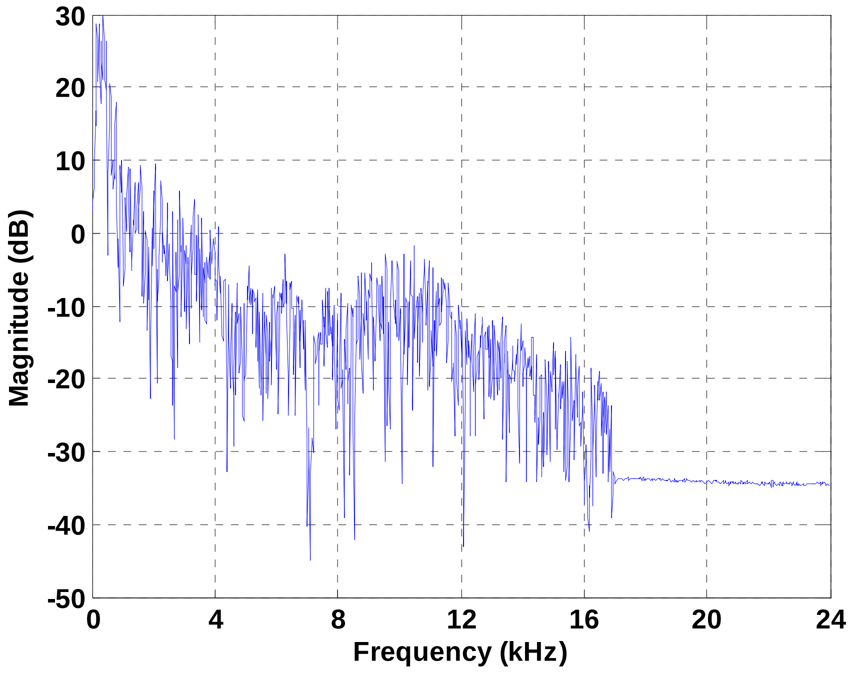


Figure 5.3 The spectrum of the HE-AAC decoded frame

From Section 3.3.3 , if the test sequence is a two-channel sequence with sampling frequency at 48 kHz, AAC is applied to the frequency range from 0 Hz to 8,250 Hz, and SBR is applied to the frequency range from 8,250 Hz to 16,875 Hz. The range from 16,875 Hz to 24,000 Hz is the high frequency range. Although the SBR module can extend further the bandwidth of a lowband signal in the decoder side, the reconstructed frequency range is usually lower than 17k Hz. The signal frequency above 17k Hz will be truncated in the decoder side.

However, the transient detector is time consuming and complex. The propose of the transient detector is to decides the proper time-frequency partition as discussed in Section 3.3.4 . The transient detection of the 3GPP HE-AAC encoder goes through the full frequency range to identify the transient signals. However, the power of the high frequency signal is generally small and the high frequency signal will be truncated in the decoder. Hence, in order to accelerate the encoder, we do not detect the transient signals of frequency above 17k Hz. Figure 5.4 shows the flow chart of the proposed simplified transient detector. We simply calculate the signal energy below 17k Hz. If the sampling frequency is not at 48k Hz, the AAC range and SBR range may be different. Details can be found in the Section 3.3.3 .

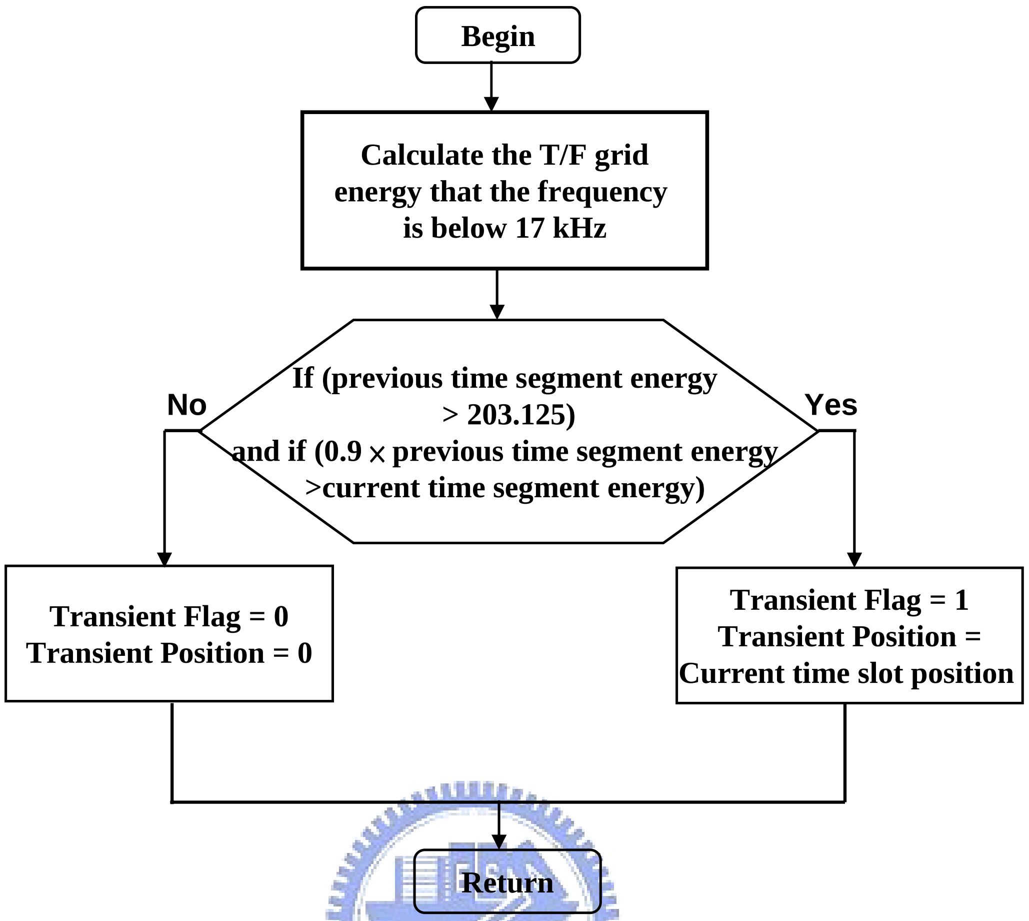

Begin
Calculate the T/F grid energy that the frequency is below 17 kHz
If (previous time segment energy > 203.125) and if (0.9 × previous time segment energy >current time segment energy)
No
Yes
Transient Flag = 0
Transient Position = 0
Transient Flag = 1
Transient Position = Current time slot position
Return

Figure 5.4 The flow chart of the proposed simplified transient detector

Table 5.2 shows the computing reduction ratio of the proposed simplified transient detector performed on the test sequences. We can notice that the operational cycles are about 24 % reduced in the transient detector function.

Table 5.2 Reduction Ratio of the Proposed Transient Detector

Test Sequences	Original Transient Detector (cycles)	Proposed Simplified Transient Detector (cycles)	Reduction Ratio %
Gspi35_2	3,441,739	2,616,269	24 %
vibr37	3,498,627	2,665,930	23.8 %
sopr44_1	3,395,772	2,575,065	24.2 %
guit58	3,381,267	2,543,060	24.8 %
edra70	3,327,745	2,454,540	26.3 %

5.3 Down-sampling Filter Complexity Analysis

The core AAC module encodes only the lowband signals, and highband signals are encoded by the SBR module. Therefore, the system for down-sampling by a factor of two is needed before the audio signals fed into the AAC module. The operation of reducing the sampling rate is called down-sampling. The input signal is initially fed to the down-sampling filter, and the output time domain signal enters the AAC encoder which has half the sampling rate. Then the AAC encoder operates on the half sampling rate of input signal, and the SBR encoder operates on the original sampling rate of input signal. However, from the profiling data, we notice that the down-sampling filter is time consuming and complex. This is due to a low-pass filter prior to the down-sampling filter in order to avoid the aliasing phenomenon. After down-sampling, the frequency spectrum would expand. Thus, the expanded spectrum may overlap and the aliasing occurs. The block diagram of the down-sampling filter module is shown in Figure 5.5. Before doing the down-sampling, the input signals convolve with the coefficients of the low-pass filter, and the low-pass filter is a FIR filter with 49 coefficients.

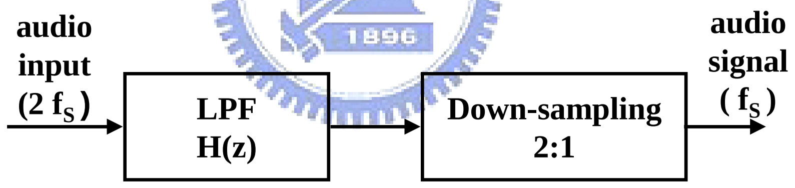


Figure 5.5 General system for down-sampling filter

However, the computations are not needed for those samples that are thrown away after the down-sampling process. Therefore, in order to reduce the complexity, we use an efficient decimation structure: the polyphase decomposition of the decimation filter. Figure 5.6 shows the efficient decimation structure based on the polyphase decomposition.

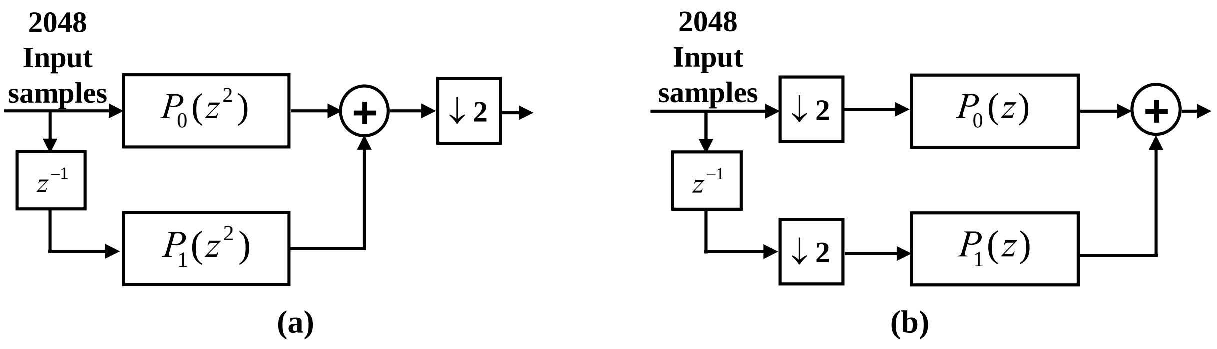


Figure 5.6 (a) The original decimation structure (b) The efficient decimation structure

The FIR low-pass filter: $H(z) = h[0] + h[1]z^{-1} + \cdots + h[48]z^{-48} = P_0(z^2) + z^{-1}P_1(z^2)$,

$$P_0(z^2) = \sum_{k=0}^{24} h[2k]\, z^{-2k}, \quad P_1(z^2) = \sum_{k=0}^{24} h[2k+1]\, z^{-2k},$$

$$P_0(z) = \sum_{k=0}^{24} h[2k]\, z^{-k}, \quad P_1(z) = \sum_{k=0}^{24} h[2k+1]\, z^{-k},$$

h[k] is the kth LPF coefficients, k is form 0 to 48.

If we use Figure 5.6(a), “an original structure”, the total number of real multiplication is 2048×49 and the total number of real addition is 2048×48. If we use Figure 5.6(b), “an efficient structure”, the total number of real multiplication is 1024×49 and the total number of real addition is 1024×48. Then because the coefficients of FIR filter are symmetrical, we only load half of the coefficients. Hence, we can reduce the load action to reduce the access of the memory, because the access of the memory is slow and power consuming.

5.4 Simplified Block switching

According to the 3GPP HE-AAC [13], the decision of whether to use long windows with a window length of 2048 samples or to use a sequence of eight short blocks with a window length of 256 samples is made in the time domain. This decision uses the transient detection algorithm to determine whether to use long or short window. The transient detection algorithm is based on the energy distribution. Because the transient is short sharp variations of sound, it induces a significant increase in the high frequency signal energy. To detect the transients, the input signal is first filtered by a high-pass filter. The transfer function of the high-pass IIR-Filter is shown below:

$$H(z) = \frac{0.7548 \times (1 - z^{-1})}{1 - 0.5095 z^{-1}} \tag{5.1}$$

After filtering, eight subblock energies are calculated by summing up the 128 consecutive squared samples (8×128=1024). These eight subblock energies represent the eight short windows of the next frame of subblocks. An attack is detected if one of these subblocks energy exceeds a sliding average of the previous energies by a constant factor of 10. This subblock energy is also greater than a constant energy level 10^6. If an attack is detected, a short window is used to encode this frame. Otherwise this frame will use the long window instead.

Figure 5.7 shows the magnitude response of the high-pass filter. The low frequency parts (below 4k Hz) of input signals are filtered.

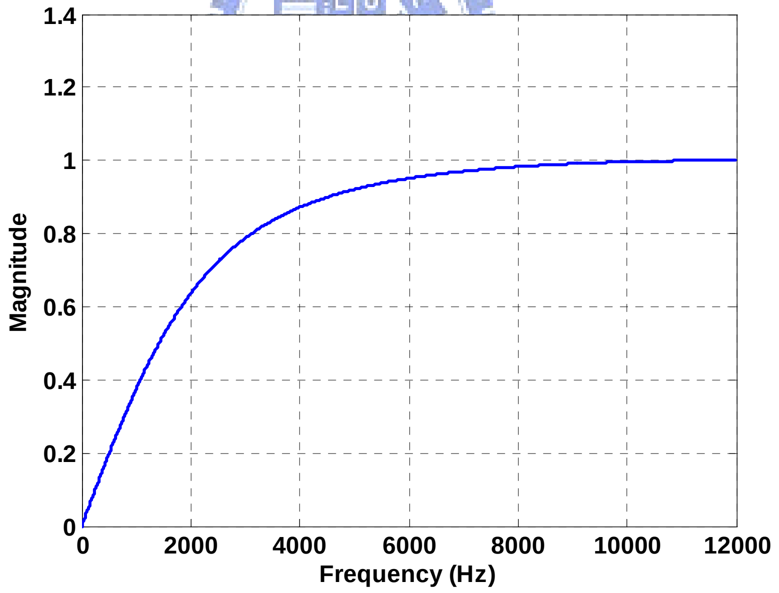


Figure 5.7 Magnitude response of high pass filter with sampling frequency at 24k Hz

However, in this filter process, the input samples convolve with the high-pass filter coefficients, and this convolution requires many computations. If the convolution can be avoided, the computational time can be reduced. Therefore, we propose a method to do the transient detection which does not include the high-pass filter. The proposed transient detection algorithm is also based on the energy distribution described previously but without the high pass filter. We know that long or short window detection is used to avoid pre-echoes. Removing the high-pass filter will affect the type of detection that is applied. If the wrong window is used, the pre-echo control may fail. However, if the wrong window is detected, the TNS module can be used to control the pre-echoes. TNS module can compensate the lack of using different window type. Hence, by removing the high-pass filter, the sound quality does not degrade and the operational cycles can be significantly reduced. Table 5.3 shows the reduction ratio of the block switching detection. We notice that the reduction ratio of the operational cycles is about 96 % in the block switching function.

Table 5.3 Reduction Ratio of the Block Switching Detection

Test Sequences	Original Block Switching Detection (cycles)	Accelerated Block Switching Detection (cycles)	Reduction Ratio %
Gspi35_2	437,742	13,752	96.8 %
vibr37	436,496	13,820	96.7 %
sopr44_1	436,100	13,247	97 %
guit58	436,213	13,567	96.8 %
edra70	435,939	13,279	97 %

5.5 Low Complexity Psychoacoustic Model

Based of the proposal of [16], we use some methods to provide a low complexity of the psychoacoustic model (PAM). The dominant calculation of spreading function, absolute threshold in quiet and other parameters that are sampling rate dependent are replaced by look-up tables.

From [13], the spreading function is used to calculate the neighboring masking threshold. The absolute threshold in quiet represents the smallest intensity that can be detected by human hearing. However, the calculation of the spreading functions, spreaded energy and absolute threshold in quiet are complicated. We find that the values of spreading functions, spreaded energy and absolute threshold in quiet are simply affected by the sampling rate and the block type used. Hence, we reduce the calculation of spreading functions, spreaded energies and absolute threshold in quiet by replacing them with a look-up table method. This method can significantly reduce the computing time at the cost of increasing the memory usage. The 3GPP HE-AAC already uses the MDCT-based PAM, which is similar to the one propose by [16].

To see the effect of the low complexity PAM, we simulated by profiling the computational cycles before and after accelerations. Table 5.4 shows the reduction ratio of the initial PAM function. The initial PAM function is used to calculate the spreading functions, spreaded energies and absolute threshold in quiet.

Table 5.5 shows the reduction ratio of the main PAM function. We rewrite the program of the main PAM function in order to accelerate the PAM part. We notice that the operational cycles of these two functions are significantly reduced.

Table 5.4 Simulated Reduction Ratio by Low Complexity PAM (Initial_PAM Function)

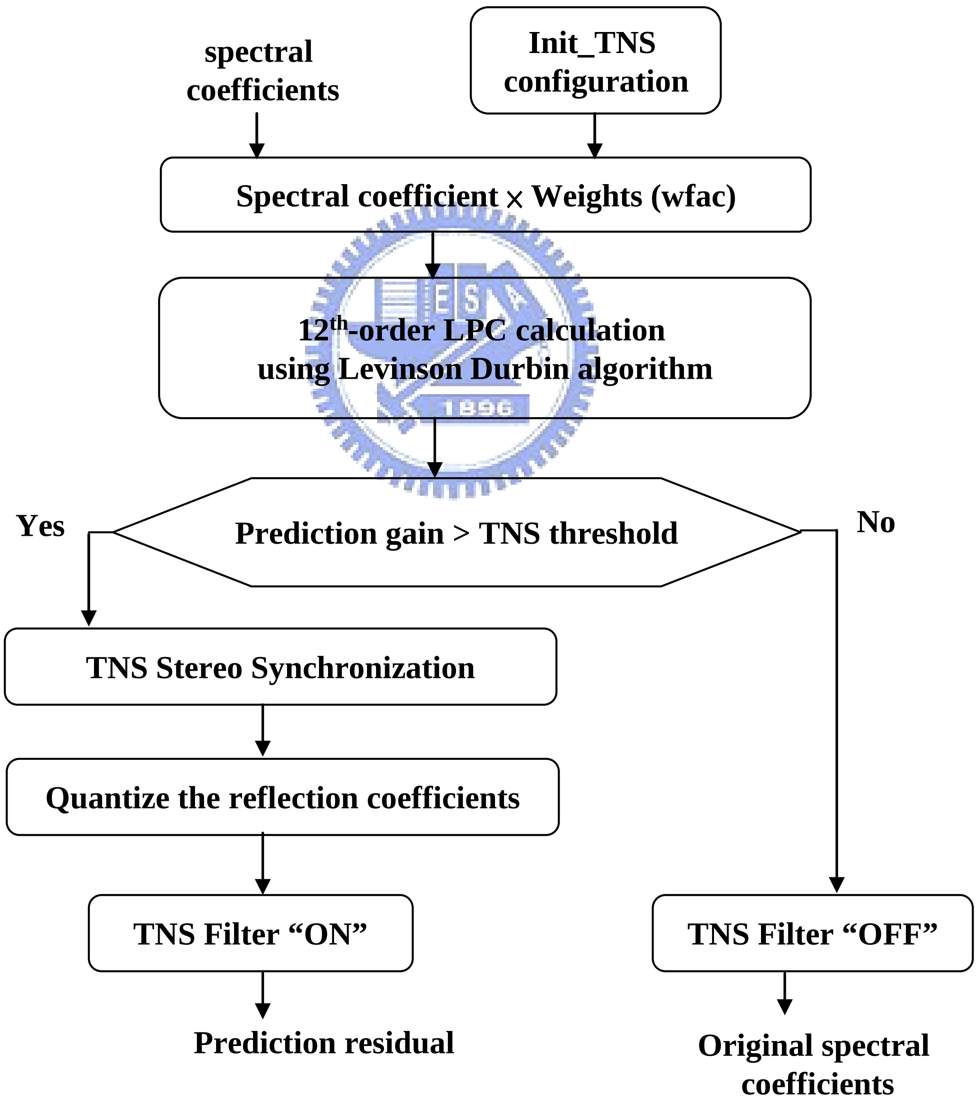
Test Sequences	Original Initial_PAM () (cycles)	Speed-up Initial_PAM () (cycles)	Reduction Ratio %
Gspi35_2	8,062,300	159,757	98 %
vibr37	8,062,300	159,757	98 %
sopr44_1	8,062,300	159,757	98 %
guit58	8,062,300	159,757	98 %
edra70	8,062,300	159,757	98 %

Table 5.5 Simulated Reduction Ratio by Low Complexity PAM (Main_PAM Function)

Test Sequences	Original Main_PAM () (cycles)	Speed-up Main_PAM () (cycles)	Reduction Ratio %
Gspi35_2	13,941,976	4,776,662	65.7 %
vibr37	11,966,594	4,864,070	59.4 %
sopr44_1	16,416,163	4,997,581	69.6 %
guit58	14,101,033	4,888,818	65.3 %
edra70	15,780,724	4,963,293	68.5 %

5.6 Simplified TNS Filter

TNS is used to control the temporal shape of the quantization noise. This is done by applying a TNS filter process to the part of the spectral data. The order of the TNS filter with the LC profile is 12 for long window and 5 for short window. TNS algorithm uses the linear predictive coding (LPC) method which is based on the Levinson-Durbin Recursion algorithm to calculate the prediction gain. TNS will be active only if the prediction gain is greater than a given threshold and the threshold is bitrate dependent and varies between 1.2 and 1.41. Figure 5.8 shows the flow chart of the TNS for the long window.



spectral coefficients
Init_TNS configuration
Spectral coefficient × Weights (wfac)
12th-order LPC calculation using Levinson Durbin algorithm
Prediction gain > TNS threshold
Yes
No
TNS Stereo Synchronization
Quantize the reflection coefficients
TNS Filter "ON"
TNS Filter "OFF"
Prediction residual
Original spectral coefficients

Figure 5.8 The flow chart of the TNS for long window. Weights (wfac) is defined below.

$$X_w = \text{Spectral coefficient} \times wfac(k)\ ,\ wfac(k) = \frac{1}{\sqrt{en(n)}}\ ,$$

$$en(n) = \sum_{k=kOffset(n)}^{k=kOffset(n+1)-1} X(k) \times X(k), \tag{5.2}$$

$kOffset(n)$: spectral line of scalefactor band n

However, the prediction gain is often smaller than TNS threshold, it wastes a lot of time to compute the 12th order prediction gain. We collect statistics and find that the percentage of TNS filter active for long window is pretty low. Table 5.6 shows the statistics. We find that the TNS active percentage for long window is pretty low except for some specific sequences.

Table 5.6 The Percentage of the TNS Filter Active for Long Window.

Test Sequences	TNS Filter Active Percentage for Long Window
Gspi35_2	9 %
vibr37	6.4 %
sopr44_1	4.6 %
guit58	15.5 %
ABBA69	9.6 %

In order to reduce the computational time, we use a simplified 6th order LPC to calculate the prediction gain. If this simplified 6th order prediction gain is greater than the threshold, the original TNS detection (12th order LPC procedure) will be turned on. This is an early termination mechanism. Figure 5.9 shows the flow chart of this simplified TNS for long window. This simplified TNS detection can reduce the operational time because the 6th order TNS detection is much faster than the 12th order one. We like to check whether this simplified method is appropriate or not.

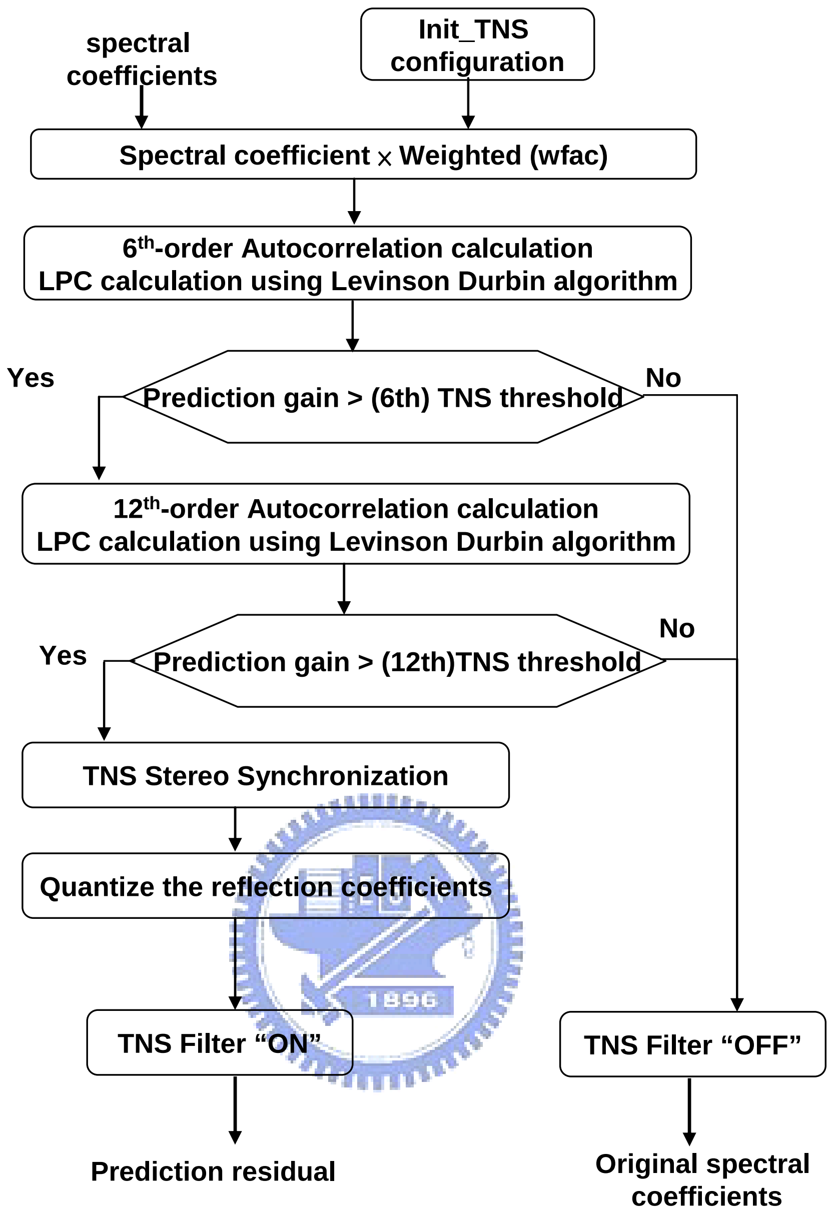

spectral coefficients
Init_TNS configuration
Spectral coefficient × Weighted (wfac)
6th-order Autocorrelation calculation
LPC calculation using Levinson Durbin algorithm
Yes
Prediction gain > (6th) TNS threshold
No
12th-order Autocorrelation calculation
LPC calculation using Levinson Durbin algorithm
Yes
Prediction gain > (12th)TNS threshold
No
TNS Stereo Synchronization
Quantize the reflection coefficients
TNS Filter "ON"
TNS Filter "OFF"
Prediction residual
Original spectral coefficients

Figure 5.9 The flow chart of the simplified TNS for long window

We have to check whether this simplified TNS is proper or not. According to the Figure 5.8 and Figure 5.9, the 6th order and the 12th order LPC processes produce four different conditions as follows:

(1) The 12th order TNS filter is "ON". The simplified 6th order TNS filter is "ON".

(2) The 12th order TNS filter is "ON". The simplified 6th order TNS filter is "OFF".

(3) The 12th order TNS filter is "OFF". The simplified 6th order TNS filter is "ON".

(4) The 12th order TNS filter is "OFF". The simplified 6th order TNS filter is "OFF".

Cases (1), (3), and (4) lead to the correct TNS detection. Only case (2) causes a different result as comparing to the original TNS detection. We collect statistics to show that case (2) is rarely happened. Table 5.7 shows the results.

Table 5.7 Cases (1), (2), (3) and (4) Percentage.

Test Sequences	Case (1) Percentage	Case (2) Percentage	Case (3) Percentage	Case (4) Percentage
Gspi35_2	9 %	0.3 %	0 %	90.7 %
vibr37	6.4 %	0 %	0 %	93.6 %
sopr44_1	4.6 %	0.4 %	0 %	95 %
guit58	15.5 %	0.2 %	0.9 %	83.4 %
ABBA69	9.6 %	0.1 %	0.7 %	89.6 %

We can say that the simplified 6th order TNS detection has about the same result as the original 12th order TNS detection, but it reduces the computing time significantly. The complexity of LPC calculation using Levinson-Durbin Recursion algorithm is known as $\boldsymbol{O}(N^2)$ arithmetic operations [30], where N is the order of the LPC. Therefore, the 6th order TNS detection is one-fourth the computing time comparing to the 12th order TNS detection. From Table 5.6, about 90% of total frame can reduce the complexity by this simplified TNS. However, this simplified TNS may attract extra computing time when the case (1) and case (3) happen. But comparing to case (4), case (1) and case (3) seldom happens. It means that this simplified TNS can reduce the complexity of TNS. Table 5.8 shows the reduction ratio of the window detection. We notice that the reduction ratio of the operational cycles is about 19 % in the TNS detection function.

Table 5.8 Simulated Reduction Ratio of the Simplified TNS

Test Sequences	Original TNS (cycles)	Simplified TNS (cycles)	Reduction Ratio %
Gspi35_2	145,168	121,939	16 %
vibr37	153,247	134,141	12.5 %
sopr44_1	162,460	134,003	17.5 %
guit58	114,419	80,972	29.2 %
edra70	126,968	103,239	18.7 %

5.7 Quantization Acceleration

As we described in Section 2.6 , the two iteration loops (R-D control) can provide the higher compression rate, but it is time consuming and complex. Many papers have suggested ways to accelerate the two iteration loops. The noise shaping method and fast bitrate rate control algorithm are proposed in [17] to reduce the complexity of the quantization module. The noise estimation method proposed in [19] is applied to derive single loop distortion control algorithm. It derives the distortion-free stepsize of each scalefactor band. According to equation 2.4 and 2.5, the formula of the non-uniform quantizer is as follows:

$$ix(i) = \operatorname{int}\left(\left(\frac{|xr(i)|^{\frac{3}{4}}}{\Delta_q}\right) + 0.4054\right), \tag{5.3}$$

where *xr(i)* is the MDCT coefficients, *ix(i)* is the quantized value, int(.) is the nearest integer operation. The definition of Δ_q is $\Delta_q = 2^{\frac{3}{16}(stepsize_q)} = 2^{\frac{3}{16}(global\_gain\text{-}scalefactor_q)}$

The quantization noise in each scalefactor band is calculated by the following formula.

$$\begin{aligned} N_q &= \sum_{i \in q} (|xr(i)| - ix(i)^{\frac{4}{3}} \times 2^{\frac{1}{4} \times stepsize_q})^2 \\ &= \sum_{i \in q} [|xr(i)| - (ix(i) \times \Delta_q)^{\frac{4}{3}}]^2 \end{aligned} \quad . \tag{5.4}$$

According to [19], the quantization noise can be rewritten as following.

$$xr_i = [(ix_i + \varepsilon_i)\Delta_q]^{4/3}$$

$$\tilde{xr}_i = (ix_i\Delta_q)^{4/3}$$

$$where\ \Delta_q = 2^{3/16(stepsize_q)}, stepsize = global\_gain - scalefactor_q,$$

xr_i : MDCT coefficient, ix_i : *quantized value,*

$\tilde{xr}_i$: requantized value, ε_i : *uniform random noise,* i : scalefactor band.

$$e_i = xr_i - \tilde{xr}_i = [(ix_i + \varepsilon_i)\Delta_q]^{4/3} - (ix_i\Delta_q)^{4/3} = [(1+\frac{\varepsilon_i}{ix_i})ix_i\Delta_q]^{4/3} - (ix_i\Delta_q)^{4/3},$$

Apply binomial theorem : (5.5)

$$e_i = (1+\frac{4}{3}\frac{\varepsilon_i}{ix_i})(ix_i\Delta_q)^{4/3} - (ix_i\Delta_q)^{4/3} = \frac{4}{3}\frac{\varepsilon_i}{ix_i}(ix_i\Delta_q)^{4/3} = \frac{4}{3}ix_i^{1/3}\varepsilon_i\Delta_q^{4/3},$$

$$E[e_i^2] = \frac{16}{9}\Delta_q^{8/3}E[ix_i^{2/3}\varepsilon_i^2], where\ E[\varepsilon_i^2] = \frac{1}{12},$$

$$E[e_i^2] = \frac{4}{27}\Delta_q^{8/3}E\left[\left(\frac{|xr_i|^{3/4}}{\Delta_q}\right)^{2/3}\right],$$

The quantization noise in each scalefactor band :

$$N_q \approx \frac{4}{27}\Delta_q^2\ E[|xr(i)|^{\frac{1}{2}}]$$

where N_q is the distortion of the qth scalefactor band.

This noise estimation builds up the relationship between quantization noise, MDCT coefficient, and *stepsize*$_q$. In order to achieve non-distortion scalefactor bands, the quantization noise have to be less than the masking threshold. We derive the non-distortion stepsize$_q$ of each scalefactor band below.

$$\begin{aligned}
&N_q \le M_q \\
&\rightarrow \frac{4}{27}\Delta_q^2\ E[|xr(i)|^{\frac{1}{2}}] \le M_q \\
&\rightarrow \frac{4}{27}\times\left(2^{3/16(stepsize_q)}\right)^2 \times E[|xr(i)|^{1/2}] \le M_q \\
&\rightarrow \left(2^{3/16(stepsize_q)}\right)^2 \le \frac{27}{4}\frac{M_q}{E[|xr(i)|^{1/2}]} \\
&\rightarrow \frac{3}{8}stepsize_q \times 0.301 \le \log_{10}(6.75M_q) - \log_{10}(E[|xr(i)|^{1/2}]) \\
&\rightarrow stepsize_q \le 8.8585\left(\log_{10}(6.75M_q) - \log_{10}(E[|xr(i)|^{1/2}])\right) \\
&\rightarrow stepsize_q \approx floor\left[8.8585\left(\log_{10}(6.75M_q) - \log_{10}(E[|xr(i)|^{1/2}])\right)\right],
\end{aligned} \tag{5.6}$$

where floor(x) is the nearest integer that is less than or equal to the number x and M_q is the masking threshold of the qth scalefactor band.

The non-distortion stepsize$_q$ is obtained in a single loop but it is an approximate value. Hence, the 3GPP HE-AAC uses the single loop distortion control algorithm [17] [18] to accelerate. We describe its quantization process flow in the following steps.

Step 1:

Use the single loop distortion control algorithm [19] to obtain the stepsize$_q$ of each scalefactor band.

Step 2:

The 3GPP HE-AAC attempts to increase or decrease the values of the stepsize$_q$ to find a lower distortion, thereby improves the Noise-to-Mask-Ratio (NMR) and quantization noise.

Step 3:

In order to decrease the side information bits, the 3GPP HE-AAC reduces the difference of the scalefactor between two adjacent scalefactor bands. Because the difference of the scalefactor will be encoded, a smaller difference between two adjacent scalefactor bands costs less bits. But decreasing the scalefactor will cost more bits in quantizing the spectrum. Hence, we should search for a single scalefactor band. While using a smaller scalefactor, the side information bits reduced are greater than the quantized spectrum bits increased. If such a scalefactor band is found and the quantization noise is smaller, then the new scalefactor is accepted.

Step 4:

Instead of improving only single scalefactor bands, the same procedure is applied to a complete region of the scalefactor bands.

Step 5:

We can obtain the *global_gain* by finding the maximum $stepsize_q$ of all possible scalefactor bands, and the scalefactor for each individual band is then equal to the *global_gain* minus the respective $stepsize_q$.

Step 6:

The 3GPP HE-AAC uses the non-uniform quantizer and Huffman coding to encode the spectrum coefficient and side information. It counts the required bits to verify the required bits have to be less than the available bits. If the required bits are greater than the available bits, the global_gain is increased, and returns to the step 6.

The flow chart of the 3GPP HE-AAC quantization is shown in the Figure 5.10.

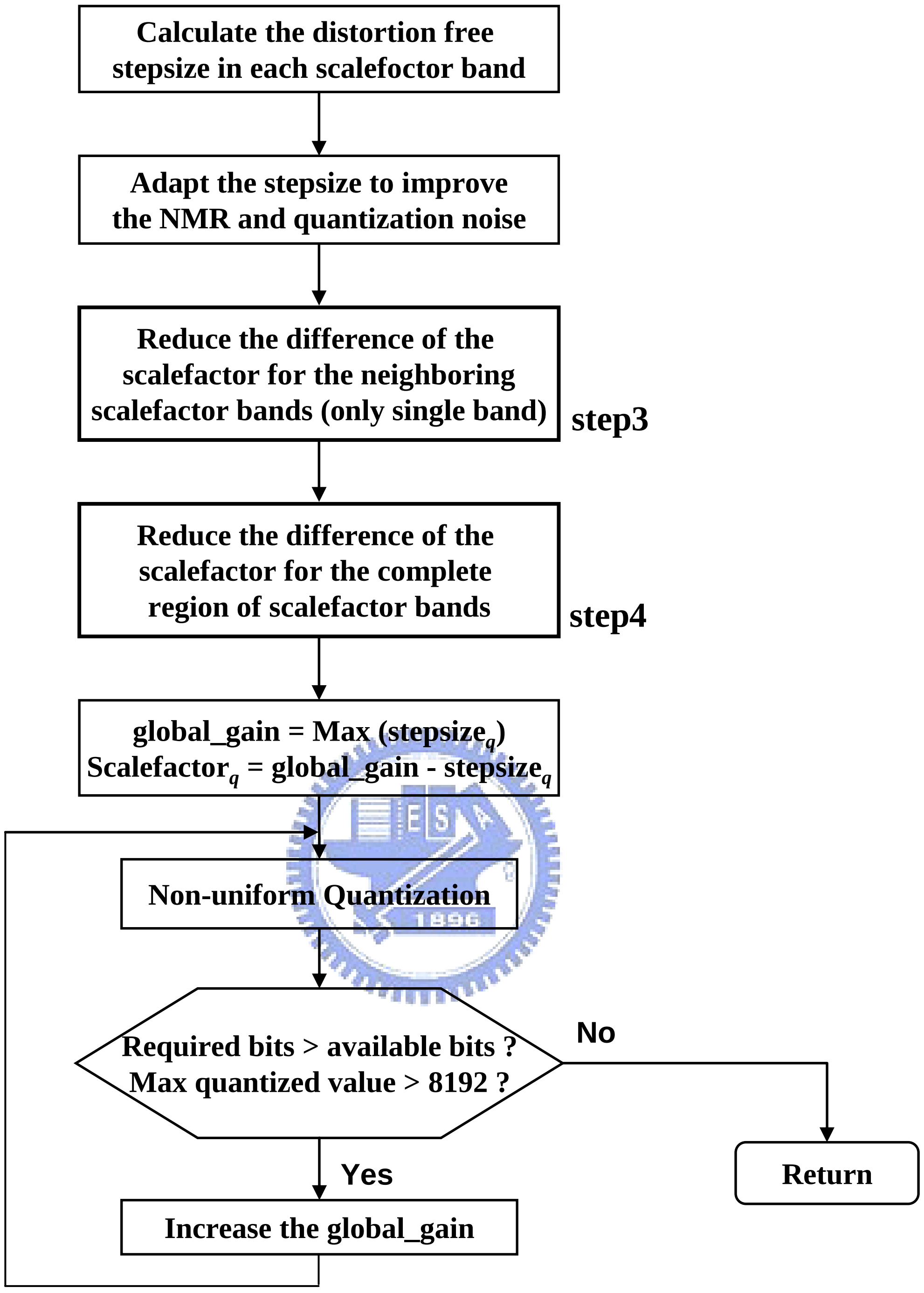

Calculate the distortion free stepsize in each scalefoctor band
Adapt the stepsize to improve the NMR and quantization noise
Reduce the difference of the scalefactor for the neighboring scalefactor bands (only single band)
step3
Reduce the difference of the scalefactor for the complete region of scalefactor bands
step4
global_gain = Max (stepsize$_q$)
Scalefactor$_q$ = global_gain - stepsize$_q$
Non-uniform Quantization
Required bits > available bits ? Max quantized value > 8192 ?
No
Return
Yes
Increase the global_gain

Figure 5.10 The flow chart of the fast quantization module

There is possibility that the masking threshold calculated by PAM may not be accurate, so the quantization module calculated based on the masking threshold is not noise-free. Figure 5.11 shows the masking threshold and signal energy in one frame. We notice that some masking thresholds are not appropriate for the quantization process, and it leads to the degradation in the audio quality.

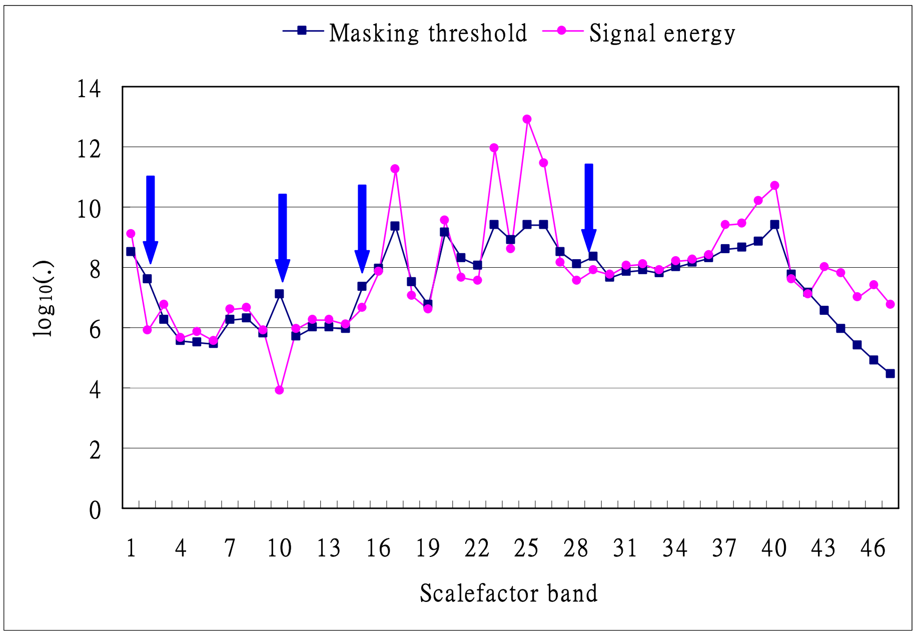


Figure 5.11 Comparison of the masking threshold and signal energy.

Therefore, step 3 and step 4 of the quantization process try to improve the audio quality and quantization noise by decreasing the stepsize. The quantized coefficients are more accurate with the decreased stepsize. Figure 5.12 shows that the stepsize calculated in step 1 (Original stepsize) and the stepsize calculated after step 3 and step 4 (Adjusted stepsize).

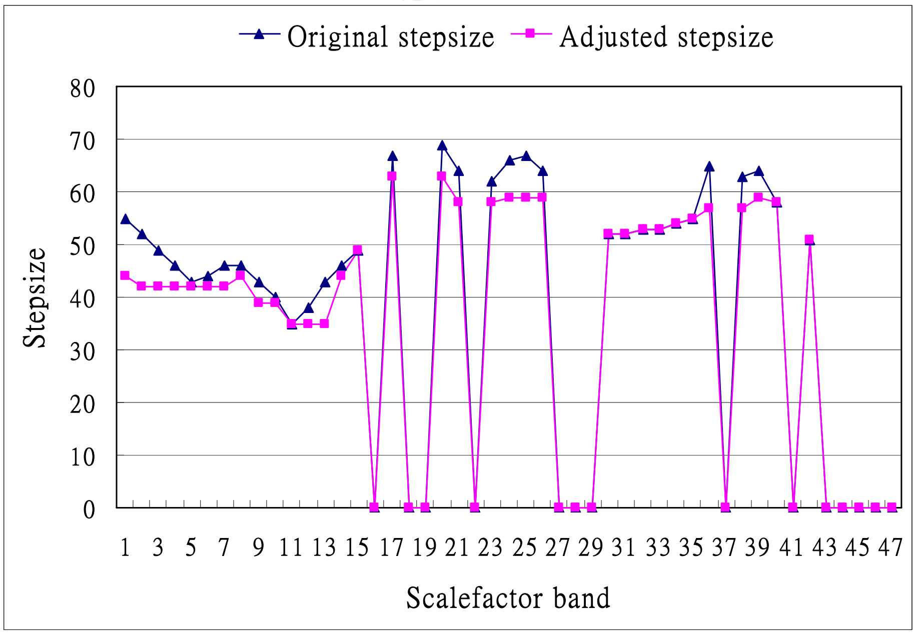


Figure 5.12 The 10th frame in a audio sequence. The original stepsize means that the stepsize calculated in step 1. Adjusted stepsize means that the stepsize after step 3 and step 4.

We notice that in the steps 3 and 4 of the quantization process, the stepsize is decreased in order to produce the better quantized coefficients. When decreasing the stepsize between two adjacent scalefactor bands, the bits used for quantized coefficients can be increased and the bits used for the side information (scalefactor) can be decreased. This implies that decreasing the bits used in side information (scalefactor) may spare extra bits used for quantized coefficients. The increased bits of the quantized coefficients would allow more accurate quantization. However, step 4 repeats the same procedure as step 3, but it spends a lot of time to decrease the difference of stepsize for the entire scalefactor bands. Therefore, in order to reduce the complexity of quantization module, we propose an accelerated method to simplify step 4.

We find that, in step 4, the stepsizes in the high frequency part of scalefactor bands are rarely changed, and the stepsizes in the low frequency part of scalefactor bands are sometimes changed. Figure 5.13 and Figure 5.14 show the results.

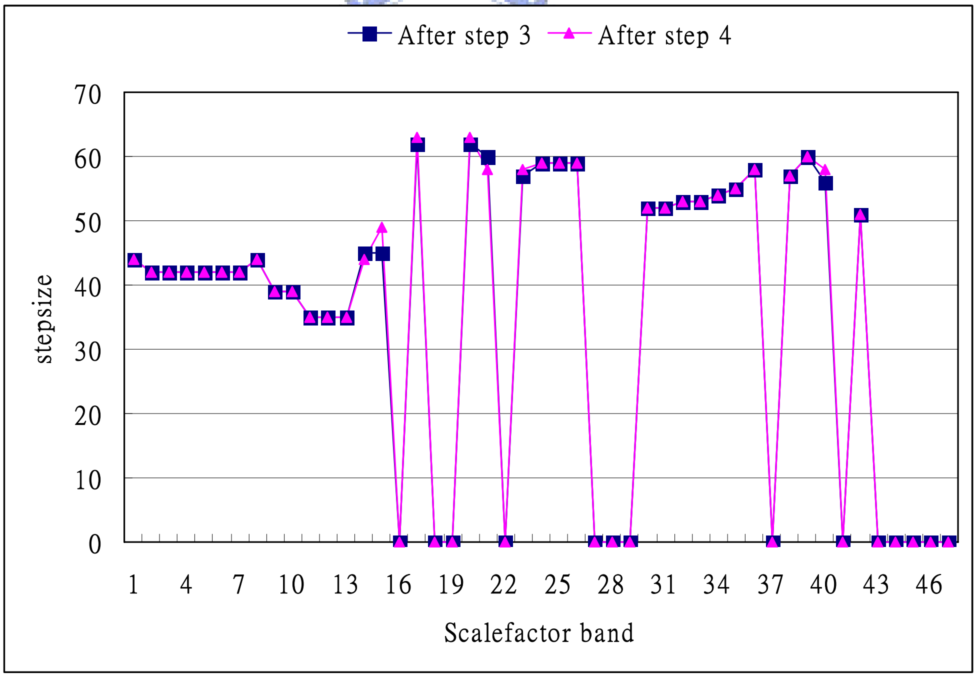


Figure 5.13 The stepsize of each scalefactor band after step 3 and step 4 in a certain frame.

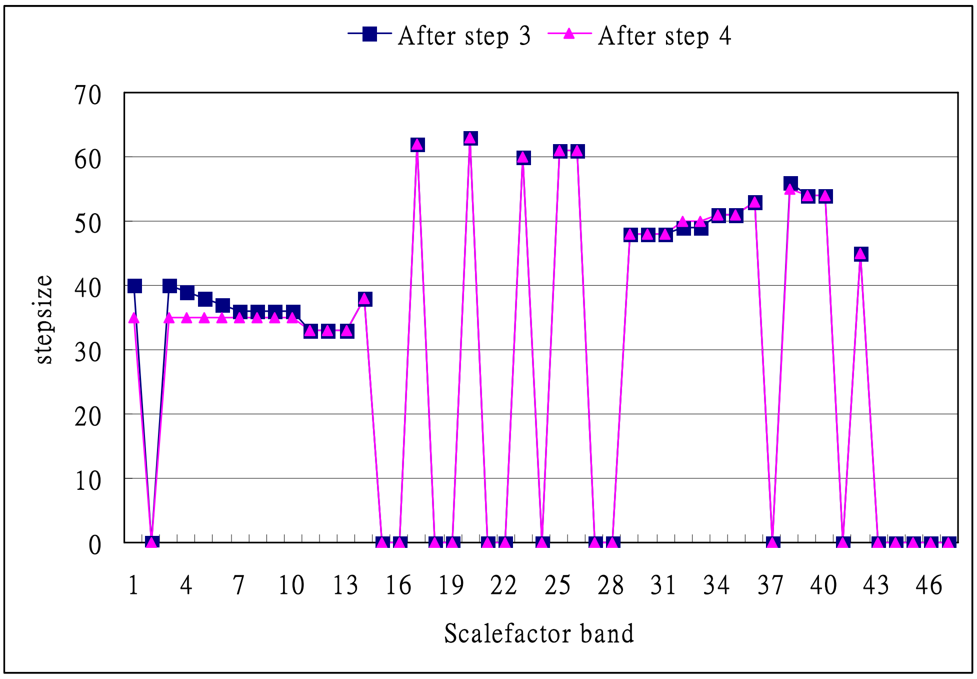


Figure 5.14 The stepsize of each scalefactor band after step 3 and step 4 in another frame.

We notice that some stepsizes are changed by step 4 of the quantization process. In Figure 5.13, there is almost no changing in stepsize in every scalefactor band. In Figure 5.14, there are a few stepsizes changed in the low frequency part of the scalefactor bands. Therefore, we do not change the stepsize in every scalefactor band. We change only the stepsizes for scalefactor band 1 to scalefactor band 33 in step 4 to accelerate the quantization module. Another reason to choose scalefactor band 33 is that the scalefactor band 33 is about 4k Hz signals. We only adjust the signals below 4k Hz which human hearing is sensitive to this range. Table 5.11 shows the reduction ratio of applying the accelerated method. We notice that the operational cycles are about 57 % reduced in the quantization process.

Table 5.9 Simulated Reduction Ratio of the Accelerated Quantization Module

Test Sequences	Original Quantization (cycles)	Accelerated Quantization (cycles)	Reduction Ratio %
Gspi35_2	6,892,276	2,751,259	60.1 %
vibr37	8,663,742	4,410,110	49.1 %
sopr44_1	10,863,258	4,709,163	56.7 %
guit58	9,912,155	4,363,515	56 %
edra70	9,590,086	4,128,428	57 %

5.8 Window Grouping

The block switching determines whether to use the long window or short window to encode the frame. If the decision is to use the short window sequence, the frame divides into eight short window sequences each with 256 samples. Then the eight short windows may be grouped and interleaved in order to reduce the side information. The coefficients associated with the contiguous short windows can be grouped to share one set of scalefactors among all scalefactor bands within the group. However, the specification 14496-3 [2] does not explicitly describe how to group the eight short window sequences. The 3GPP HE-AAC has a set of grouping rules [13]. From the block switching detection in Section 5.4 , if an attack is detected in the frame, short window sequences will be used to encode this frame. Then the eight short windows will divide into 4 groups. The number of short windows in each group depends on the position of the attack. The position of the attack is shown in Figure 5.15. The 3GPP HE-AAC grouping rules is shown in Table 5.10.

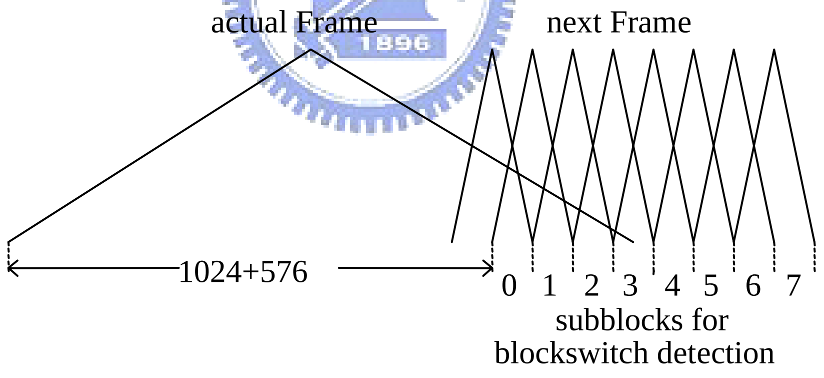


Figure 5.15 Block switching detection

Table 5.10 Grouping of Windows in an Eight Short Window Sequences

Position of Attack	Number of Windows in Group 1	Number of Windows in Group 2	Number of Windows in Group 3	Number of Windows in Group 4
0	1	3	3	1
1	1	1	3	3
2	2	1	3	2
3	3	1	3	1
4	3	1	1	3
5	3	2	1	2
6	3	3	1	1
7	3	3	1	1

However, 3GPP HE-AAC does not provide the sufficient reasons for dividing the eight short window sequences into four groups. Thus, we should verify that whether the grouping rules are appropriate or not. The idea come from another AAC codec, which known as FAAC [28]. FAAC codec has only one group in an eight short window sequence. We find that if we apply two groups or four groups, the audio quality and the Noise-to-Mask-Ratio (NMR) would be almost the same compared to the one group which shown in Figure 5.16 and Figure 5.17. We discover that the 3GPP HE-ACC has a similar performance degradation in this situation, and we find the reason for this situation.

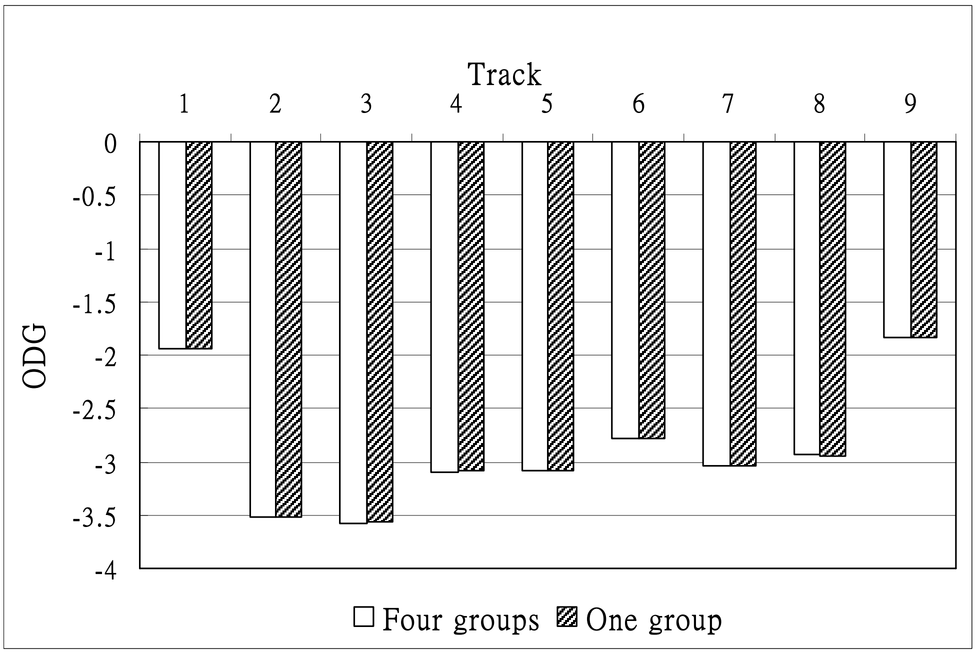


Figure 5.16 The ODG value of applying four groups and one group.

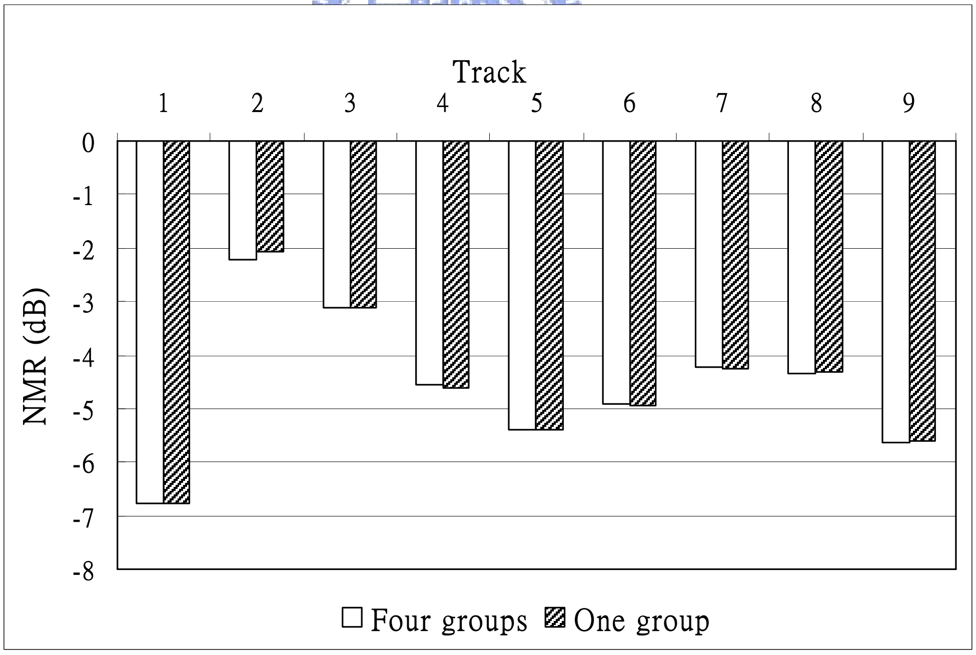


Figure 5.17 The NMR value of applying four groups and one group.

If we apply two or four groups to encode this frame, the way of allocating the bits to each group does not clearly specified in the 3GPP documents [13]. The quantization module does two loop R-D control from group 1 to group 4, and bit allocation module allocates the bits from group 1 to group 4. If the transient signals appear in the group 4, then group 4 should get more bits. However, the specification does not clearly define how to allocate bits to each groups, and the stopping conditions of the R-D control for short window does not configure properly. The other reason is that the side information is larger in the four groups than in one group. If the side information is large, the available bits for the spectrum coefficients will be less.

Therefore, we apply one group for the short window frame. The R-D control of the quantization module is accelerated because there is only one group. Table 5.11 shows the reduction ratio of using one group in the quantization module. We notice that the operational cycles are about 7 % reduced in the quantization function.

Table 5.11 Simulated Reduction Ratio of Applied the One Group in Quantization Module

Test Sequences	Original Quantization Module (cycles)	Apply One Group to Quantization Module (cycles)	Reduction Ratio %
Gspi35_2	6,892,276	6,444,278	6.5 %
vibr37	8,663,742	8,252,028	4.8 %
sopr44_1	10,863,258	10,045,721	7.5 %
guit58	9,912,155	9,042,005	8.7 %
edra70	9,635,256	9,039,708	6.2 %

5.9 Experiments and Acceleration Results

Several experiments verifying the above acceleration methods are presented in the following. The proposed HE-AAC encoder is implemented on the 32-bit fixed-point C6416T DSP processor. HE-AAC encoder runs under the AAC LC profile with SBR. The range of the bitrate is from 24k to 48k bps for stereo channel. The performance evaluation of HE-AAC includes the encoding speed, code size, and coding quality.

5.9.1 Test Sequence

We take the nine test audio sequences from EBU [28]. These audio sequences are CD-quality sampled at 48k Hz with 16 bits for stereo channel. These audio sequences and their characteristics are listed in Table 5.12. Tracks 1 to 3 are single instruments. Tracks 4 and 5 are vocal. Track 6 is human speech. Track 7 is the solo instrument. Tracks 8 and 9 are pop music.

Table 5.12 Test Audio Files and Their Characteristic

Track	Audio File	Time (sec)	Mode	Description
1	horn23_2	25	stereo	Horn. Melodious phrase
2	Gspi35_2	9	stereo	Glockenspiel. Melodious phrase
3	harp40_1	8	stereo	Harpsichord, Melodious phrase
4	sopr44_1	12	stereo	Soprano. Vocal
5	Bass47_1	24	stereo	Bass. Vocal
6	spfe49_1	19	stereo	Female speech in English
7	guit58	13	stereo	Guitar. Solo instrument
8	ABBA69	21	stereo	ABBA. Pop music
9	edra70	14	stereo	Eddie rabbitt. Pop music

5.9.2 Profile on the proposed HE-AAC

In order to verify the performance of the proposed HE-AAC encoder, we simulate with the C6416T DSP simulator to profile the cycles before and after acceleration of the HE-AAC encoder. We use the “File level” (-o3) optimization for the proposed HE-AAC encoder on DSP. The original 3GPP HE-AAC encoder uses no optimization and “File level” (-o3) optimization. Table 5.13 and Table 5.14 and show the reduction ratios of the speed-up performance.

Table 5.13 Simulated Reduction Ratios of the Proposed HE-AAC Compared to the Original HE-AAC with Bitrate at 48k bps.

Track	Reduction Ratio % of Comparing 3GPP HE-AAC (**no opt.**) with Proposed HE-AAC (**-o3**)	Reduction Ratio % of Comparing 3GPP HE-AAC (**-o3**) With Proposed HE-AAC (**-o3**)
1	82.7 %	55.6 %
2	82.6 %	56.5 %
3	80.8 %	54.9 %
4	82 %	54.1 %
5	81.8 %	56.7 %
6	82.3 %	57.3 %
7	81.7 %	56.5 %
8	81.3 %	55.2 %
9	81.1 %	53.8 %

Table 5.14 Simulated Reduction Ratios of the Proposed HE-AAC Compared to the Original HE-AAC with Bitrate at 32k bps.

Track	Reduction ratio % of comparing 3GPP HE-AAC (**no opt.**) with proposed HE-AAC (**-o3**)	Reduction ratio % of comparing 3GPP HE-AAC (**no opt.**) with proposed HE-AAC (**-o3**)
1	82.5 %	55.3 %
2	82.3 %	56.1 %
3	80.1 %	54.5 %
4	81.7 %	53.7 %
5	81.2 %	56.3 %
6	81.9 %	57 %
7	81.5 %	56.2 %
8	81.2 %	54.9 %
9	81 %	53.4 %

5.9.3 Million Instructions per Second (MIPS)

The computational complexity of the original HE-AAC encoder is 61 MIPS. After our acceleration, the computational complexity of the proposed HE-AAC encoder can be reduced to 50 MIPS.

5.9.4 Memory / Code-size Improvement

We present the code size and data size (ROM) and RAM requirement comparison between the original and the proposed HE-AAC encoder. The smaller ROM and RAM requirement are more desirable for the embedded system. The data size (ROM) of the proposed HE-AAC is about 41.7 kB. The code size (ROM) the proposed HE-AAC is about 230 kB. The RAM (static and dynamic memory) the proposed HE-AAC is about 90 kB. Table 5.15 shows the reduction ratio of the code size and RAM requirement. Therefore, we can reduce the memory usage on DSP system.

Table 5.15 Code Size and RAM Requirement with Bitrate at 48k bps.

	Original HE-AAC (byte)	Proposed HE-AAC (byte)	reduction ratio%
Code size (ROM)	377k	230k	39 %
RAM	100k	90k	10 %

5.9.5 Encoding Quality

In order to evaluate the coding quality, ITU-R Recommendation BS. 1378 [26] is adopted as the objective audio quality measurement method, which defines the Objective Difference Grade (ODG). The ODG values ideally ranges from 0 to -4, where 0 corresponds to an imperceptible difference between reference and test signal and -4 corresponds to the very annoying difference. Therefore, the ODG value that is close to zero represents the better sound quality. Table 5.16 shows the scales of the ODG.

Table 5.16 The Scales of ODG

ODG scale	Quality
0	Imperceptible
-0.1 to -1	Perceptible but not annoying
-1.1 to -2	Slightly annoying
-2.1 to -3	Annoying
-3.1 to -4	Very annoying

Apart from ODG, Noise-to-Mask-Ratio (NMR) can be used as an alternative method to measure the sound quality. NMR is the ratio of the noise generated by the encoding process to the masking threshold calculated by the PAM. Negative NMR value represents the noise is masked by masking threshold. Therefore, the smaller NMR corresponds to the better sound quality. The values of ODG and NMR are calculated by EAQUAL [27] software. We compare the sound quality of the compressed audio with the sound of the uncompressed audio by EAQUAL. The EAQUAL has been widely used to measure the compression technique due to its capability of detect perceptual difference sensible to human hearing system.

Table 5.17 and 5.18 show the results of the original 3GPP HE-AAC encoder and proposed HE-AAC encoder with bitrate at 48k and 32k bps. The decoder that we use is FAAD2 [32].

Table 5.17 The ODG and NMR of the Original and the Proposed HE-AAC Encoder of 48k bps Bitrate.

Track	Original (ODG)	Original (NMR)	Proposed (ODG)	Proposed (NMR)
1	-1.94	-6.7689	-1.95	-6.7692
2	-3.56	-3.0803	-3.52	-3.1671
3	-3.62	-3.5643	-3.63	-3.5948
4	-3.10	-4.5748	-3.15	-4.4212
5	-3.08	-5.3934	-3.10	-5.2647
6	-3.23	-5.1082	-3.32	-4.7368
7	-3.03	-4.2553	-3.13	-3.9331
8	-2.94	-4.3282	-2.98	-4.2870
9	-2.09	-5.6609	-2.15	-5.4219

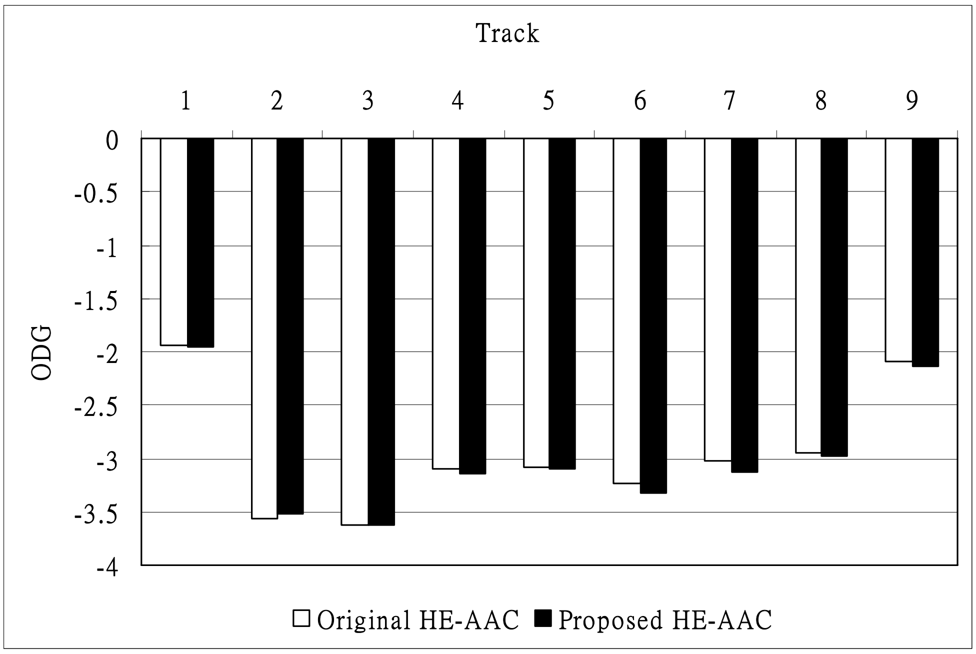


Figure 5.18 The ODG of the original and the proposed HE-AAC of 48k bps bitrate.

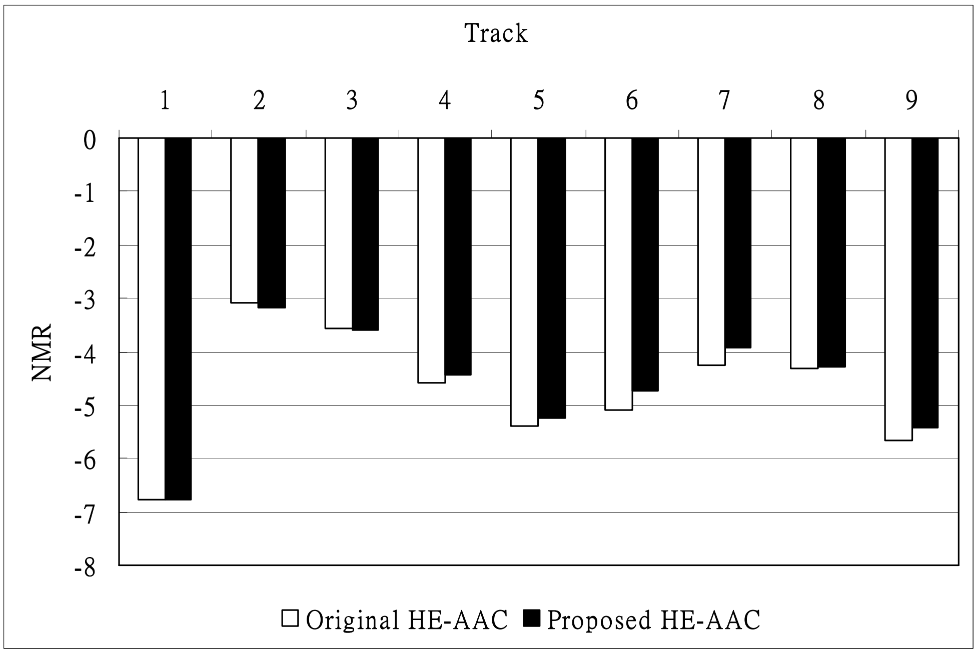


Figure 5.19 The NMR of the original and the proposed HE-AAC of 48k bps bitrate.

Table 5.18 The ODG and NMR of the Original and the Proposed HE-AAC Encoder of 32k bps Bitrate.

Track	Original (ODG)	Original (NMR)	Proposed (ODG)	Proposed (NMR)
1	-2.23	-6.3075	-2.21	-6.3371
2	-3.64	-1.6547	-3.65	-1.6242
3	-3.73	-1.0310	-3.73	-1.0287
4	-3.27	-3.4430	-3.28	-3.4202
5	-2.34	-3.4714	-2.35	-3.3868
6	-3.54	-3.1952	-3.58	-3.0127
7	-3.53	-2.4583	-3.54	-2.4315
8	-3.56	-2.8753	-3.59	-2.8321
9	-2.94	-4.1159	-2.97	-3.9063

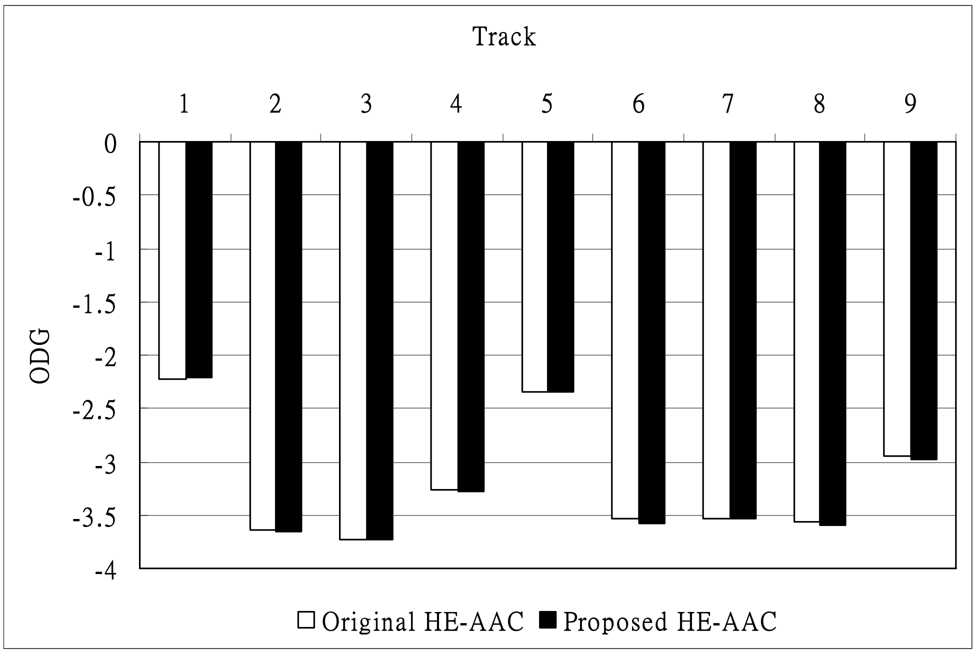


Figure 5.20 The ODG of the original and the proposed HE-AAC of 32k bps bitrate.

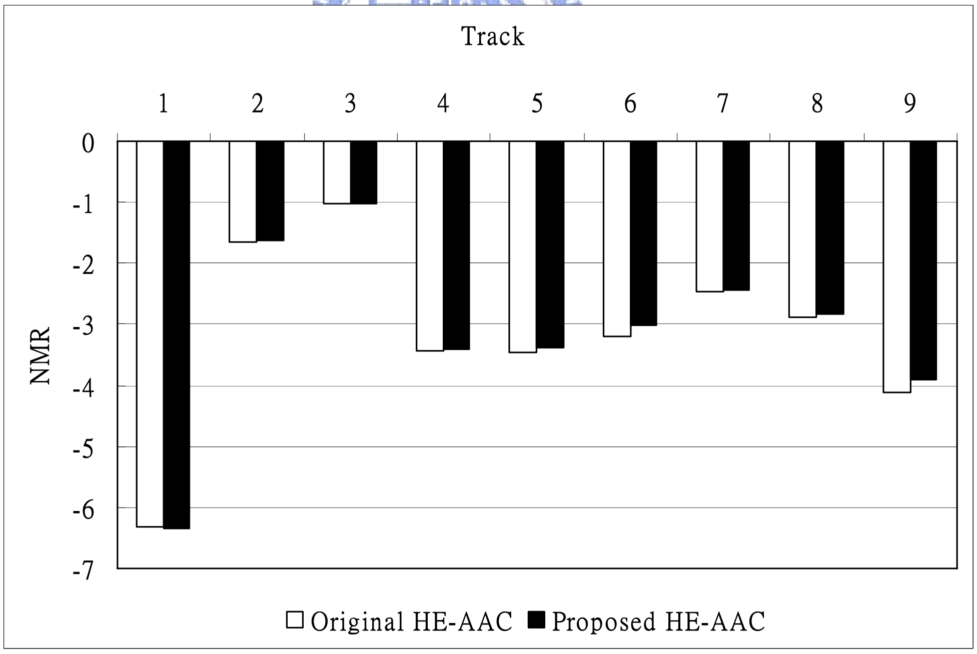


Figure 5.21 The NMR of the original and the proposed HE-AAC of 32k bps bitrate.

Chapter 6
Conclusion and Future Work

6.1 Conclusion

The main goal of this thesis is to accelerate the MPEG-4 HE-AAC encoder and to implement it on the TI C6416T DSP processor. Our proposed acceleration methods efficiently reduced the complexities of the HE-AAC encoder. These methods included transient detector acceleration, fast down-sampling filter, simplified block switching mechanism, low complexity psychoacoustic model, simplified TNS, fast quantization, and simplified window grouping module. The transient detector acceleration and fast down-sampling filter are the acceleration methods for the SBR portion. For the transient detector, we analyzed the compressed audio spectrum and the SBR frequency band tables in Section 3.3.3 . We observed that the frequency above 17k Hz is generally low power and is truncated at the end of process. Hence, in finding the transients, we simply calculated the signal energy of the frequency below 17k Hz to accelerate the transient detector. For down-sampling filter, we used the poly-phase decomposition to reduce its computations.

The simplified block switching, low-complexity psychoacoustic model, simplified TNS, fast quantization, and simplified window grouping module are the acceleration methods for the AAC portion. In the simplified block switching, we removed the high-pass filter to speed up the AAC encoder and the experimented data shows that removing the high-pass filter still maintained good audio quality. In the low complexity PAM approach, we reduced the calculation of spreading functions and spreaded energies by replacing them with a look-up table. For the simplified TNS, we used an early termination method to accelerate the TNS module. This method significantly reduced the computations of TNS. For fast quantization, we used a single loop distortion control algorithm at the outer loop to speed up the quantization. We applied the noise estimation method to derive the non-distortion stepsize of each scalefactor band. We also proposed a simplified step of checking the complete region of the scalefactor bands. For window grouping acceleration, we used only one group for the eight short windows to replace four groups.

Several experiments at different bitrates have been conducted to verify the acceleration methods. The speed-up performance, memory requirement, and audio quality were taken into consideration together. The speed improvement of the proposed HE-AAC was about 55 % over the original 3GPP HE-AAC under the same compiler optimization level. The computational complexity of the proposed HE-AAC encoder could be reduced to 50 MIPS. For the memory requirement, the code size requirement was reduced by 39% and the RAM requirement was reduced by 10% when comparing to the 3GPP HE-AAC codec. As for the objective sound quality tests, we maintained the same level of the sound quality.

6.2 Future Works

Our MPEG-4 HE-AAC (aacPlus) codec is mainly concentrated on the aacPlus version 1. The aacPlus v1 is a combination of AAC and SBR. SBR exploits the possibilities of a parameterized representation of the highband signals. However, aacPlus v2 adds a new technology to the aacPlus v1 in order to support lower bitrate coding. This new technology is called Parametric Stereo (PS). Figure 6.1 shows the aacPlus audio codec family.

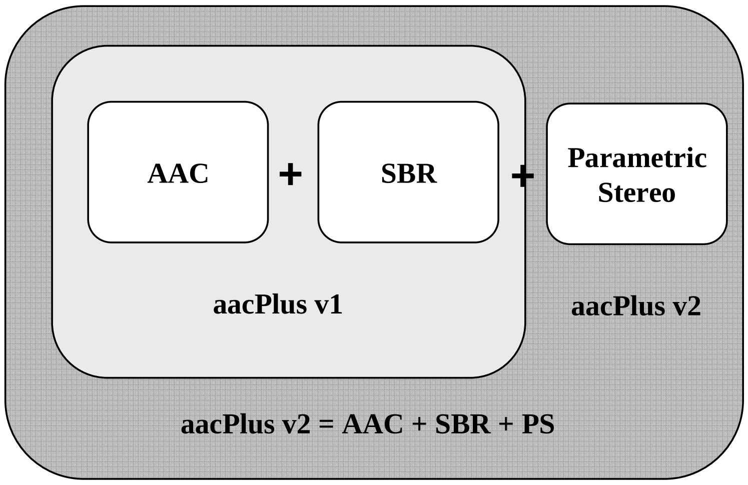


Figure 6.1 aacPlus audio codec family

PS module increases the coding efficiency by exploiting a parametric representation of the stereo image of a given input signal. Then, in order to provide lower bitrate coding and to maintain a high audio quality, aacPlus v2 can be chosen as the audio codec. But the parametric stereo is also a time consuming module, and should be speeded up. Therefore, accelerating the aacPlus v2 and implementing it on the DSP system can be a useful and challenging task. Our acceleration methods for AAC and SBR can be a part of techniques used for accelerating the aacPlus v2.

References

[1] ISO/IEC JTC1/SC29/WG11, "Text of ISO/IEC 13818-7:2005 (MPEG-2 AAC 4th edition)", ISO/IEC JTC1/SC29/WG11 N7126, April 2005.

[2] ISO/IEC JTC1/SC29/WG11, "Draft ISO/IEC 14496-3:2001/Amd.2:2004 (Audio 3rd Edition)", ISO/IEC JTC1/SC29/WG11 N7027, Jan. 2005.

[3] T. Painter and A. Spanias, "Perceptual Coding of Digital Audio", *Proc. the IEEE*, Vol. 88, Issue 4, pp. 451-515, Apr. 2000.

[4] M. Dietz and et al., "Spectral Band Replication, a novel approach in audio coding", *112th Audio Engineering Society (AES) Convention*, Munich, May 2002.

[5] M. Wolters and et al., "A closer look into MPEG-4 High Efficiency AAC", *115th Audio Engineering Society (AES) Convention*, New York, October 2003.

[6] P. Ekstrand, "Bandwidth Extension of Audio Signals by Spectral Band Replication", *IEEE Benelux Workshop on Model based Processing and Coding of Audio (MPCA-2002)*, Leuven, Belgium, Nov 15, 2002.

[7] M. Dietz and M. Wolters, "Enhancing Perceptual Audio Coding through Spectral Band Replication", *Coding Technologies*, Nuremberg, Germany.

[8] Digital Radio Mondiale (DRM). Available: http://www.drm.org/ .

[9] Third Generation Partnership Project (3GPP). Available: http://www.3gpp.org/ .

[10] Digital Video Broadcasting (DVB). Available: http://www.dvb.org/ .

[11] Coding Technologies. Available: http://www.codingtechnologies.com/ .

[12] 3GPP Telecommunication Standard 26.401: "General audio codec audio processing functions; Enhanced aacPlus general audio codec; General description", Mar. 2005.

[13] 3GPP Telecommunication Standard 26.403: "General audio codec audio processing

functions; Enhanced aacPlus general audio codec; Encoder specification; Advanced Audio Coding (AAC) part", Sep. 2004.

[14] 3GPP Telecommunication Standard 26.404: "General audio codec audio processing functions; Enhanced aacPlus general audio codec; Encoder specification; Spectral Band Replication (SBR) part", Sep. 2004.

[15] T.-H. Tsai, and et al., "Design of a low power psycho-acoustic model co-processor for MPEG-2/4 AAC LC stereo encoder", *Proc. of the IEEE Int. Symp. Circuits and Systems*, Vol. 2, May, 2003, pp. 552-555.

[16] S.-W. Huang, and et al., "A low complexity design of psychoacousitc model for MPEG-2/4 advanced audio coding", *IEEE Trans. Consumer Elec.*, Nov., 2004.

[17] C.-Y. Lee and et al., "A fast audio bit allocation technique based on a linear R-D model", *IEEE Trans. Consumer Elec.*, Vol. 48, Issue 3, pp. 662-670, Aug. 2002.

[18] C.-M. Liu and et al., "A fast bit alloaction method for MPEG layer III", *Proc. of the ICCE*, pp. 22-23, 1999.

[19] C.-T. Chen, "A new bit allocation for MPEG layer III," Master thesis, Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, ROC, 1998.

[20] Y.-L. Lin, "MPEG-2/4 low complexity AAC encoder optimization and implementation on a strongARM platform," M.S. thesis, Department of Electrical and Control Engineering , National Chiao Tung University, Hsinchu, Taiwan, ROC, 2005.

[21] Texas Instruments, "TMS320C6000 Programmer's Guide", Literature number SPRU198F, Feb. 2001.

[22] Texas Instruments, "TMS320C6000 CPU and Instruction Set Reference Guide", Literature number SPRU189F, Jan. 2000.

[23] Texas Instruments, "TMS320C64x Technical Overview", Literature number SPRU395B, Jan. 2001.

[24] Sundance DSP System, “SMT 395 User Manual”, May 2001.

[25] Texas Instruments, “TMS320C6414T, TMS320C6415T, TMS320C6416T FIXED POINT DIGITAL SIGNAL PROCESSORS”, Literature number SPRU228H, NOV. 2003.

[26] ITU-R Recommendation BS.1387, “Method for objective measurements of perceived audio quality”, 2001.

[27] EAQUAL. [Online]. Available: http://www.mp3-tech.org/programmer/sources/eaqual.tgz

[28] European Broadcasting Union (EBU), Sound Quality Assessment Material: Recordings for Subjective Tests, Brussels, Belgium, Apr. 1988.

[29] M. Bosi and Richard E. Goldberg, *Introduction to digital audio coding and standards*. Kluwer Academic Publisher Press, 2003.

[30] K. Sayood, *Introduction to data compression*. Morgan Kaufmann Publisher, 2000.

[31] FAAC. Freeware Advanced Audio Coder. v1.24 source code [Online]. Available: http://www.audiocoding.com/modules/mydownloads/

[32] FAAD2. Freeware Advanced Audio Decoder source code. [Online]. Available: http://www.audiocoding.com/modules/mydownloads/

[33] MPEG-4 AAC and High Efficiency AAC Reference Software ISO/IEC 14496-5 / FPDAM6: 2004. ISO/IEC JTC1/SC29/WG11 N6248, May 2004.

[34] H.-Y. Huang, “Design and Implementation of AAC Audio Coder,” M.S. thesis, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, ROC, 2004.

自　　傳

黃育彰, 西元 1981 年生於高雄市。 西元 2004 畢業於台灣新竹國立交通大學電機與控制工程學系，之後進入交通大學電子研究所攻讀碩士學位，於 2006 年取得碩士學位。研究方向為數位訊號處理、音訊壓縮。

Yu-Chang Huang was born in KaoHsiung in 1981. He received the BS degree in Electrical and Control Engineering from National Chiao Tung University (NCTU), HsinChu, Taiwan in 2004. He pursue master degree in Electronics Engineering at National Chiao Tung University. His research interests are digital signal processing and audio compression.