國 立 交 通 大 學

電機與控制工程研究所

碩 士 論 文

MPEG-1 LAYER 3 音訊解碼器於DSP晶片之即時軟體實現

Real-Time Implementation of MPEG-1 Layer 3 Audio Decoder on a DSP Chip

研 究 生：賴鴻志

指導教授：胡竹生 博士

中華民國 九十 年 六 月

MPEG-1 LAYER 3 音訊解碼器於DSP晶片之即時軟體實現

REAL-TIME IMPLEMENTATION OF MPEG-1 LAYER 3 AUDIO DECODER ON A DSP CHIP

研 究 生：賴 鴻 志　Student: Hung-Chih Lai

指導教授：胡 竹 生　Advisor: Dr. Jwu-Shen Hu

國立交通大學
電機與控制工程研究所
碩士論文

A Thesis
Submitted to Department of Electrical and Control Engineering
College of Electrical Engineering and Computer Science
National Chiao-Tung University
in Partial Fulfillment of the Requirements
for the Degree of Master
in
Electrical and Control Engineering
June 2001
Hsinchu, Taiwan, Republic of China

中華民國　九十　年　六　月

國立交通大學

研究所碩士班

論文口試委員會審定書

本　　校 電機與控制工程 研究所 賴鴻志 君

所提論文 MPEG-1 LAYER 3 音訊解碼器於 DSP 晶片之即時軟體實現
Real-Time Implementation of MPEG-1 Layer 3 Audio Decoder on a DSP Chip

合於碩士論文資格水準、業經本委員會評審認可。

口試委員：

指導教授：

系主任：　　教授

中華民國　九十 年 六 月 十三 日

MPEG-1 LAYER 3 音訊解碼器於DSP晶片之即時軟體實現

研究生：賴鴻志　　指導教授：胡 竹 生　博士

國立交通大學電機與控制工程研究所

摘　　要

本論文主要針對 MPEG-1 Layer 3 音訊編碼標準作研究並在定點 DSP 晶片上實作一即時解碼器。本論文分成兩大部分，第一部分敘述 MPEG-1 Layer 3 音訊編碼標準，包括壓縮與解壓縮。第二部分簡介軟硬體平台並實作出一可即時播放出音樂的即時解碼器。實作的重點包括組合語言的撰寫、定點數的運算、高效率的運算法則、多功及多執行序的管理與結果比較。此解碼器程式記憶體共使用 7.1k 字元(word)，資料記憶體共使用 17.2k 字元(word)。若以此定點晶片最快速度 100 MHz 執行，則此解碼器的解碼速度為 34.16 MIPS，約佔此晶片 34%的運算能力。

REAL-TIME IMPLEMENTATION OF MPEG-1 LAYER 3 AUDIO DECODER ON A DSP CHIP

Student: Hung-Chih Lai Advisor: Dr. Jwu-Shen Hu

Institute of Electrical and Control Engineering
National Chiao-Tung University

Abstract

In this thesis, an investigation is done for MPEG-1 Layer 3 audio coding standard. A real-time implementation on a fixed-point DSP chip is also proposed. This thesis is twofold: one is to introduce the MPEG-1 Layer 3 audio coding standard, including encoder and decoder. The other is to describe software and hardware development environment and implement a real-time decoder. The keys of implementation are hand-coded in assembly language, fixed-point operation, an efficient algorithm, multi-task and multi-thread management and verification. The decoder uses 7.1 kwords of program memory and 17.2 kwords of data memory, respectively. This decoder is 34.16 MIPS and uses about 34% computation power of this DSP chip if it run at its maximum speed, 100MHz.

誌　謝

感謝我的指導老師胡竹生博士對我的指導，讓我在碩士兩年內能夠獲獎、能到 TI 增廣見聞、能完成一本著作、完成論文並順利畢業。在定點 DSP 領域的專業知識都是從老師身上所學的。另外要感謝鄭木火老師以及林源倍老師撥冗指導與寶貴的建議，使得本論文的內容更加完備。

感謝實驗室的所有伙伴，余祥華學長、澎哥、胖胖、許誌尤，詹玉麒、廖建龍等學長，讓我在研一時感受到實驗室的溫暖，對學弟的照顧。學長們的專業能力更是讓我佩服，也才有機會和學長們一起得到第一屆 TIC100 的首獎與 23 萬的獎金。謝謝實驗室的同學：酷酷的小陶子學長、常開車載我們的瓊宏，不太說話的俊德、和我同居兩年的阿邦以及一起為了論文，為了讓 DSP 達到 real-time 而奮鬥的凱。還有實驗室的學弟妹們：劉維瀚、劉欣慈、蘇宗敏、林家銘、鄭价呈，幫我買便當、簽到、泡咖啡、當值日生。

感謝德州儀器 FAE 部門的朋友：讓我學到很多、會請我吃東西、會帶我到客戶那逛，很看重我的 Jeffrey，還有 Peter, Terence, YT, Paul, Johnny 等，在 TI 讓我學到很多，我的論文也是在那得到靈感的。此外，還有研華及 IPC 的朋友們。

謝謝陪伴我六年的手語社，手語社是我課業的避風港兼休息室，每當課業遇到瓶頸或是有所突破時，第一個想到的就是到社窩走一走。除了課業外，社團幾乎佔了所有的時間。還要感謝系上許許多多的同學，坤在 mp3 方面的幫忙還有會找我一起打羽球的梁耀文老師、廖德誠老師、鄭木火老師，計概老師蔡中庸老師。

僅以此論文獻給我摯愛的雙親賴秋顯先生和朱桂蕖女士，為我挑起生活的重擔，使我在求學之路能夠順利，最重要的是您們對我的愛，是我一輩子不會忘記的。還有大哥常偉、弟弟信仁在家裡對家庭的照顧。最後，謝謝我的女朋友周貞伶，常常要我帶她出去玩，也時時督促我寫論文，陪我一起度過求學生涯中最困難、最忙碌卻也是最精彩的部分。

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Audio Signal Compression

During the passed ten years, digital audio has essentially replaced analog audio because the digital audio has many advantages compared to the analog. Digital audio provides better preservation, cheaper distribution, and invokes various audio processing easily. The most common format of all the digital audio is the Pulse Code Modulation (PCM). PCM samples the sound at a fixed rate with fixed bits for the audio signal [1]. Recently, new digital audio applications have been used for network, communication, broadcasting, multimedia, and computer systems which face constraints such as channel bandwidth, limited storage capacity and low cost. The audio of CD-quality equivalent using a standard sampling frequency 44.1 kHz, 16 bits quantization, two channel stereo requries 1,411,200 bits per second. This means to be able to play an audio PCM format with CD-quality over a network, we need more than 1.4 Mbit/s in bandwidth. Storing one song of 4-minutes duration needs over 40 Mbyte disk storage. Therefore, how to transmit audio signal on Internet with small bitrate or store in hardware with less

volume is an important issue. As a result, audio compression technology becomes more and more precious.

In comparison with the digital video compression and speech compression, the digital audio is relatively complex. The human ear has a sensitivity over a dynamic range exceeding 100 dB. In contrast, the vision ability of human visual system are higher than the resolution of general television or displayer. Compared to speech coding, there are two disadvantage of audio coding. One is that no source model of the audio signal is known as the speech coding. The other is the quality of the reproduced audio signal should be much higher than speech coding.

Due to the urgency of audio compression and complexity it required, several methods have been purposed to solve this problem. Audio coding methods can separate into two categories: transform coding and subband coding. Transform coding algorithms use unitary transforms for the time-to-frequency analysis. These algorithms typically achieve high resolution spectral estimates with a good compromise of adequate temporal resolution, like MSC (Multiple adaptive Spectral audio Coding, Thompson Consumer Electronics) [2], OCF (Optimum Coding in the Frequency domain, Brandenburg in 1987) [3], PXFM/hybrid (Perceptual transform coder, Johnston in 1988) [4], CNET (Mahieux in 1989) [5]. Combining elements of above algorithms, ASPEC (Adaptive Spectral Perceptual Entropy Coding) [6] was included in the ISO/IEC MPEG-1 audio coding standards. Instead of transform coding, subband coding relies upon frequency-domain representations of the signal obtained from banks of bandpass filters. The MUSICAM (Masking pattern adapted Universal Subband Integrated Coding And Multiplexing) [7] which was also included in ISO/IEC MPEG-1 audio coding

stadnard is derived from MASCAM (Masking pattern Adapted Subband Coding And Multiplexing) which was purposed by IRT [8].

1.2 Digital Signal Processor

The general purpose Digital Signal Processor (DSP) is developed for implementation of a wide variety of algorithms. Algorithms well suited for DSP implementation are characterized by multiply-accumulate operations and linear data access. Most algorithms require a fast, convenient framework for getting large sequences of data, manipulating them, and restoring them. Farther, many signal processing algorithms are organizied by multiply-accumulate operations such as filtering or convolution. Therefore, DSP had been applied in many field like control [9], consumer [10], military [11], image [12], telecommunications [13] and audio [14].

So far the performance of high level programming language, for example C or C++, can not keep up assembly language. DSP provides many instructions for implemention of using assembly language. In addition, DSP supports circular and bit-reversed addressing. Circular addressing enables the user to set up a group of memory locations that may be accessed one after the other without any extra test to determine when the last memory location has been reached. Pointers to the memory locations automatically wrap around to the beginning of the set once they reach the end. Bit-reversed addressing caters to the needs of certain signal processing techniques, notably the decimation-in-time Fast Fourier Transformation or Discrete Cosine Transformation, effectively streamlining a computationally-intensive algorithm [15].

There are two types of DSP according to their operation mode and architecture. A floating-point DSP is a processor capable of handling floating-point arithmetic where real operands are represented using exponents. It performs higher performance but requires higher power consumption and costs. A fixed-point DSP is a processor that does arithmetic operations using integer arithmetic with no exponents. It uses the scaling property to replace the exponent part and has to manipulate the location of decimal point. A fixed-point DSP has powe-efficient performance and low cost.

1.3 Motivation

The International Standard Organization and the International Electrotechnical Commission (ISO/IEC) adopted the MPEG-1 algorithm which was developed by the Motion Picture Experts Group (MPEG) in 1992. The MPEG/Audio is one part of a multiple part standard that addressed the compression of video part (11172-2), the compression of audio part (11172-3) [16], and synchronization of the audio, video, and related data streams, system part (11172-1). The audio part (MPEG/Audio), which is the first standardize algorithm in audio compression field, had be applied into many way [17], including

- Internet streaming (Microsoft Media player, Apple Quick time)
- Digital audio broadcasting (Eureka-147 DAB, ARIB, DRM)
- Sound for digital television (DVB, Video CD, HDTV)
- Portable audio devices (mpman, mplayer3, VAIO, Rio, and many more)

The MPEG/Audio offers three levels of compression, each with increasing complexity and better sound quality. The MPEG/Audio Layer 3 (as known as MP3) is the most complex scheme and provides best sound quality of the three layers. In MPEG-1 standard, there are many filtering and matrix operations that are well suited for DSP's multiply-accumulate characteristic. Since MPEG/Audio Layer 3 is the most complex layer that provides the best sound quality and DSP is well suited for its most operations, we intend to realize a MPEG/Audio decoder on DSP chip. In this thesis focus will be on the MPEG/Audio Layer 3 of the MPEG-1 standard only. Principles and functionality of MPEG/Audio Layer 3 will be introduced in this thesis and real-time implementation of decoder with mixed C and assembly language on a single DSP chip will be presented.

1.4 Preface

This thesis contains five chapters. Chapter 1 is in the premise. Chapter 2 introduces the MPEG-1 Layer 3 standard, including principles and functionality. In Chapter 3, the hardware and software environment where the decoder is developed are introduced. Chapter 4 presents the implementation and performance verification. This thesis finishes with conclusion and future works in Chapter 5.

Chapter 2

MPEG/Audio Layer 3 Coding

In this chapter, we describe the basic principles and algorithms in the MPEG/Audio Layer 3 coding standard. The most important reason why MPEG/Audio Layer 3 can compress digital audio signals effectively without perceptual loss is to use the “quantization” and “entropy coding” techniques. Quantization removes the auditory irrelevant parts of the audio signal without losing the sound quality by exploiting the perceptual properties of the human auditory system. Removal of such irrelevant parts results in inaudible distortion. Entropy coding is a lossless coding method that encodes the quantized data to minimize the entropy of the quantized value of the audio signal thereby achieving the goal of compression without any quality loss. The two techniques are also wildly adopted in other compression standard, like image (JPEG) and video (H.261) compression.

Section 2.1 will introduce the MPEG/Audio Layer 3 encoding standard and its algorithm. Section 2.2 will explain the decoding process.

2.1 MPEG/Audio Layer 3 Encoding Algorithm

In this section the MPEG/Audio Layer 3 encoder will be described with its functionality. The description of the encoding process is based on the block diagram in Figure 2.1. The input audio signal which comes from a single channel PCM signal is passed through a polyphase filter bank. This filter bank divides the input signal into 32 equally-space frequency subbands. After this process, the samples in each subband are still in the time domain. A Modified Discrete Cosine Transform (MDCT) is then used to map the samples in each subband to frequency domain. In the meantime, input signal after FFT transformation passes through a psychoacoustic model that determines the ratio of the signal energy to the masking threshold for each subband. The distortion control block uses the signal-to-mask ratios (SMR) from the psychoacoustic model to decide how to assign the total nomber of code bits available for the quantization of the subband signals to minimize the audibility of the quantization noise. The quantized subband samples are coded with the lossless Huffman coding to decrease the entropy of samples. Finally, the end block takes the Huffman coded subband samples and side information into a packed bitstream according to the MPEG/Audio standard.

In the following subsections, we will describe the operation and the functionality in detail for each block in the block diagram.

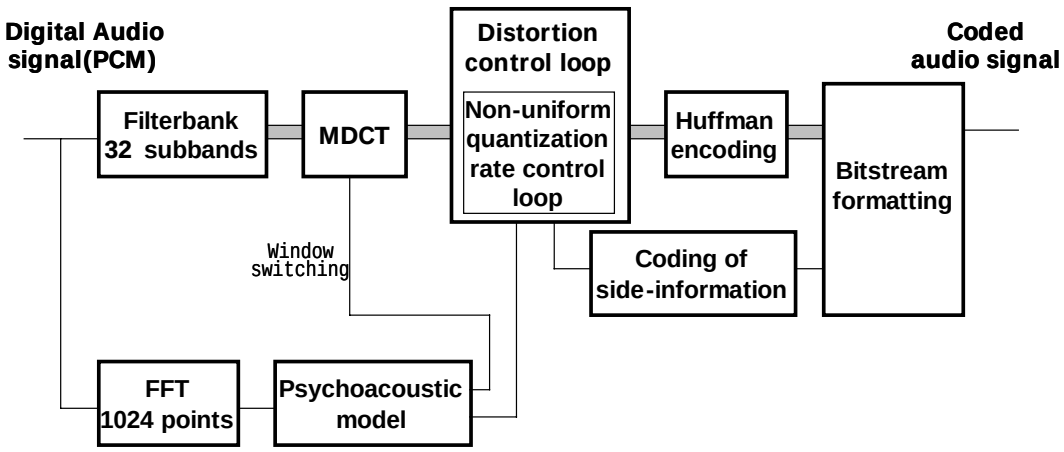

Figure 2.1 MPEG/Audio Layer 3 encoder block diagram [17].

2.1.1 Analysis Polyphase Filter Bank

The first step in the encoding process is the filtering of the audio signal through a filter bank. The analysis polyphase filter bank divides the audio signal into 32 equal-width frequency subbands and decimates the subband samples by a factor 32 with good time resolution and reasonable frequency resolution. Decimation results in an aggregate number of subband samples that equals the source signal but also introduces some aliasing [18].

In one frame a sequence of 1152 PCM audio samples are filtered so each subband contains 36 subband samples. The following equation derives the filter bank outputs:

$$St[i] = \sum_{k=0}^{63} \sum_{j=0}^{7} M[i][k] * (C[k + 64j] * x[k + 64j]) \tag{2.1}$$

where:
i is the subband index and ranges from 0 to 31,
St[i] is the filter output sample for subband i at time t, where t is an integer multiple of 32 audio sample intervals,

C[n] is one of 512 coefficients of the analysis window defined in the standard,
x[n] is an audio input sample read from a 512 sample buffer, and
$M[i][k] = \cos[\frac{(2*i+1)*(k-16)*\pi}{64}]$ are the analysis matrix coefficients.

Manipulate Equation (2.1) into a intelligible filter convolution Equation (2.2) for more convenient to analysis.

$$St[i] = \sum_{n=0}^{511} x[t-n] * Hi[n] \tag{2.2}$$

where:
$x[\tau]$ is an audio sample at time τ,
$Hi[n] = h[n] * \cos[\frac{(2*i+1)*(n-16)*\pi}{64}]$ with
h[n]= -C[n], if the integer part of (n/64) is odd,
= C[n] otherwise, for n=0 to 511.

The coefficients of h[n] are the prototype low-pass filter for the polyphase filter bank, as Figure 2.2 shown. The modulation of the prototype filter (h[n]) with a cosine term (M[i][k]) results in filter shifting. Clearly, Hi[n] are the filter banks that shift the low-pass response to the appropriate frequency band, so these are called “polyphase” filter bank. These filters have center frequencies at odd multiples of π/(64T) and each has a bandwidth of π/(32T) where T is the audio sampling period. For example, if sampling period T is 31.25 ms (32 kHz sampling frequency), the frequency response of the polyphase filters has center freqeuncy 250 Hz and bandwidth 500 Hz while 2π presents the sampling freqeuncy, as Figure 2.3 shown.

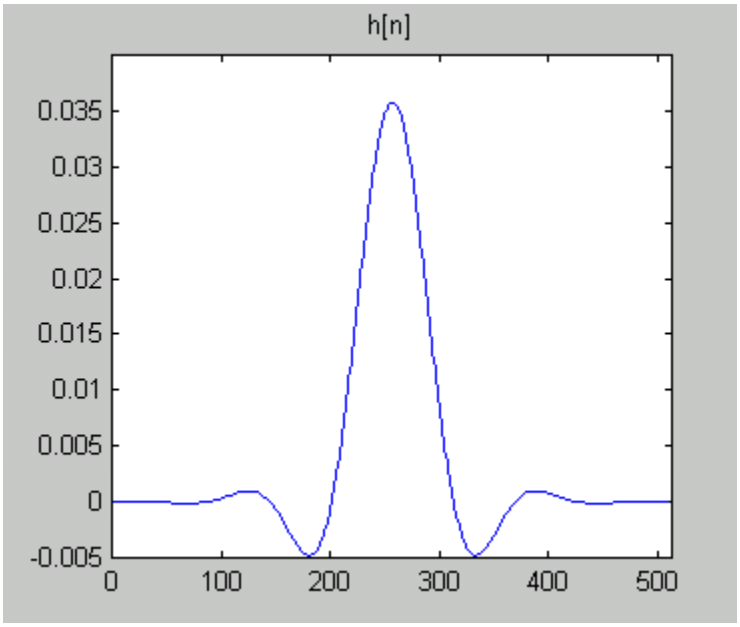

Figure 2.2 H[n], the prototype low-pass filter for the polyphase filter bank.

In Figure 2.3, the overlap of adjacent polyphase filters is inimicable for audio compression, because alias will be introduced by this overlap and decimation [19]. Signal frequency near nominal subband edges will generate output in two adjacent polyphase filter. Figure 2.4 shows how a pure sinusoid tone, which has frequency near subband edge, appears at the output of two polyphase filters. This disadvantage will be cancelled by using a series of butterfly computations later and appropriate design of analysis/synthesis filter bank in the encoding/decoding part [20].

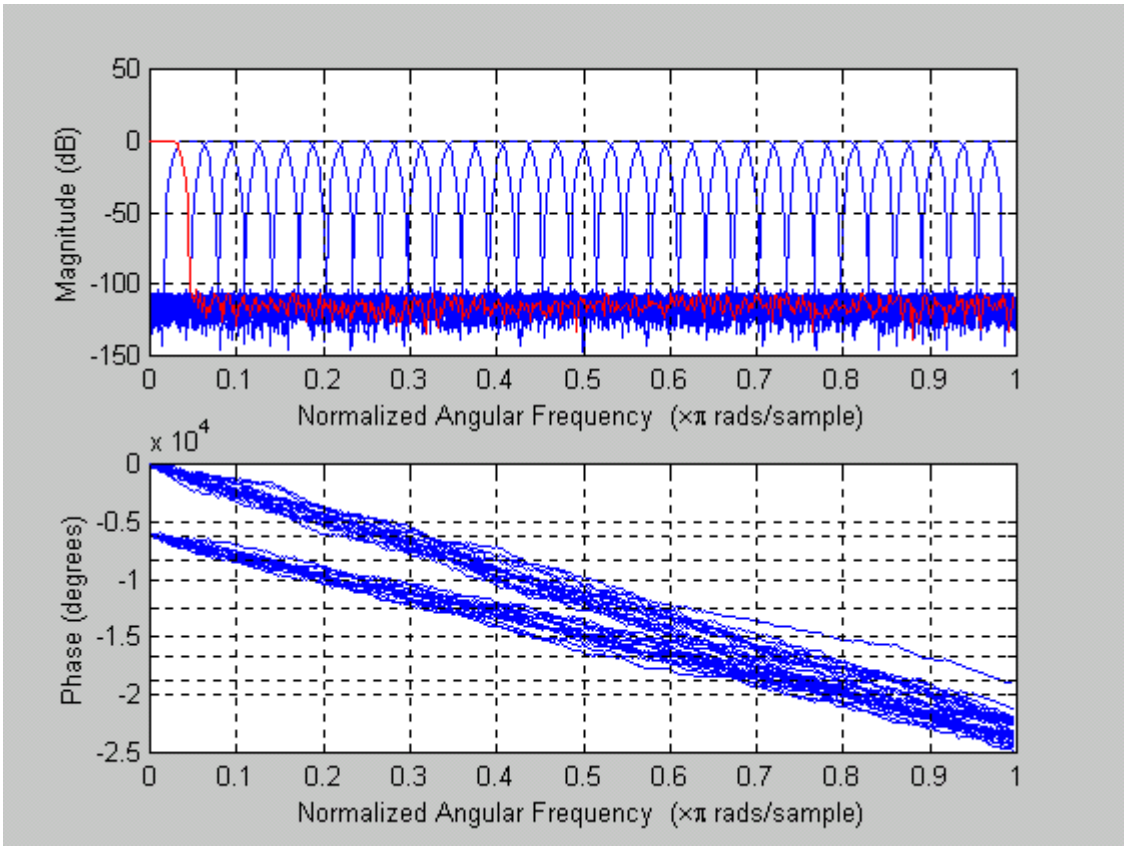

Figure 2.3 Frequency response of polyphase filter bank.

The samples of the output in each subband are still in the time domain, and will be processed through a MDCT block which transfers the samples from the time domain to the frequency domain. Figure 2.5 illustrates the analysis polyphase filter bank and its detail procedure.

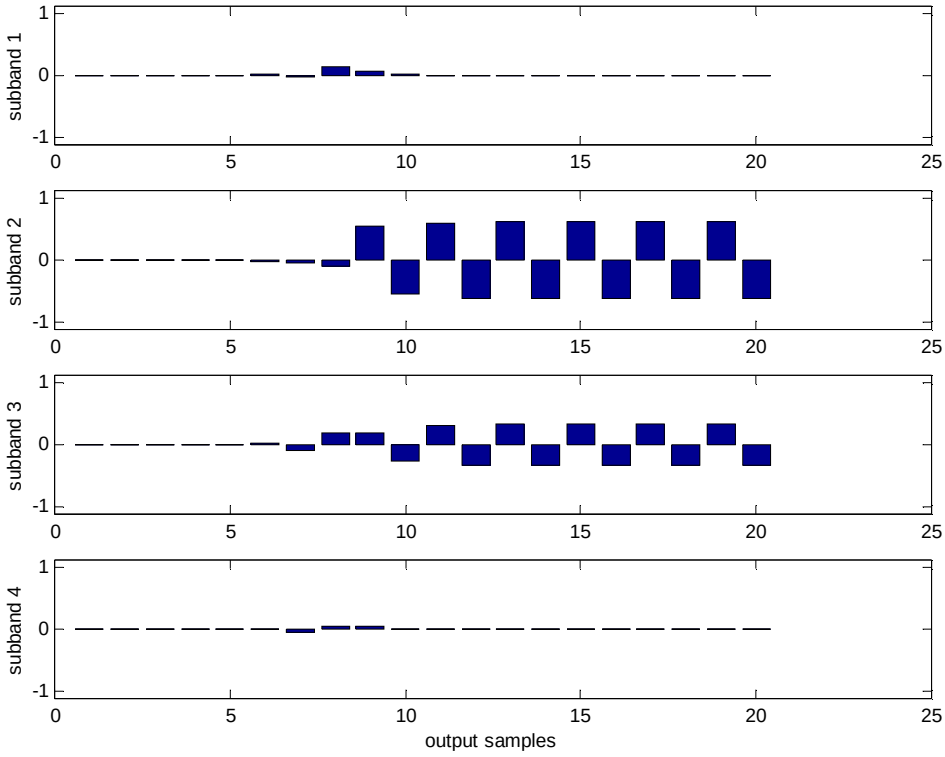

Figure 2.4 Pure sinusoid input can produce non-zero output for two subbands [19].

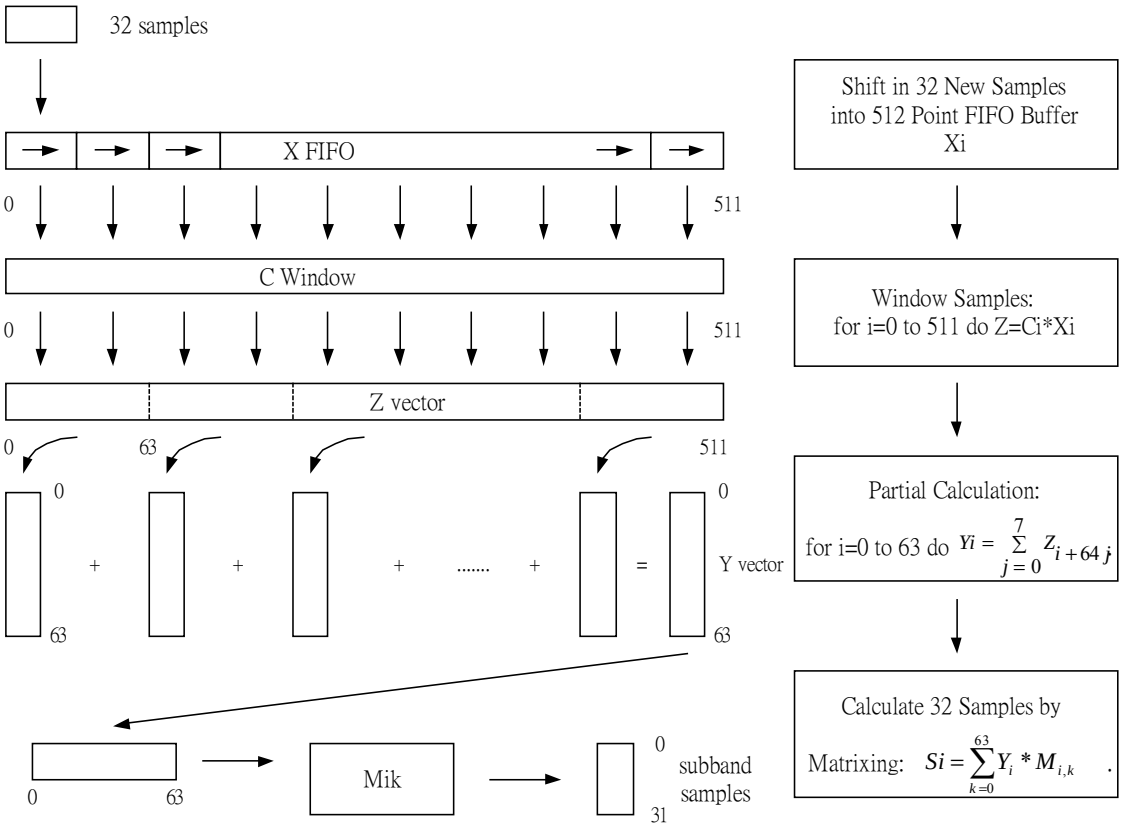

Figure 2.5 Diagram and procedure of analysis polyphase filter bank [16].

2.1.2 MDCT and Alias Reduction

◆ Modified Discrete Cosine Transformation

In this process the 32 subbands are mapped into a Modified Discrete Cosine Transform (MDCT) [21] representation. Performing this transformation will enhance the frequency resolution per subband. Equation (2.3) shows the formula for MDCT transformation.

$$X_i = \sum_{k=0}^{n-1} z_k \cos\left(\frac{\pi}{2n}(2k+1+\frac{n}{2})(2i+1)\right) \quad \text{, for i} = 0 \sim \frac{n}{2}\text{-1} \tag{2.3}$$

Prior to computing the MDCT four window functions are applied to the subband samples. MPEG/Audio Layer 3 specifies two different MDCT block lengths: a long block of 18 samples or a short block of 6. The windowing use either long window or short window depending on the dynamics within each subband. If the subband samples in a given subband show a stationary behavior, the regular window, long window (Type 0), is used. If the subband samples contain transients, a short window (Type 2) is applied to subdivide the subband outputs in frequency in order to enhance the time resolution. The switching mechanism helps to prevent the appearance of pre-echo phenomenon which will be introduced in next subsection. The other two windows used to handle the transitions from long-to-short or short-to-long are called start window (Type 1) and stop window (Type 3). Note that the short block length is one third of a long block. In short block mode, three short blocks replace a long block so that the number of MDCT samples for a frame of audio samples is unchanged regradless of the block size selection. For a given frame of audio samples, the MDCT can all have same block length (long or short) or have a mixed-block mode. In the mixed block mode the MDCT uses long window for the two lower frequency subbands and short window for the 30 upper subbands. This mode provides better frequency resolution for the lower frequencies without sacrificing time resolution for the higher frequencies.

The window functions are given as following and shown in Figure 2.5.

a) block_type=0 (long window)

$$z_i = x_i \sin(\frac{\pi}{36}(i+\frac{1}{2})) \qquad \text{, for i = 0 ~ 35} \tag{2.3}$$

b) block_type=1 (start window)

$$z_i = \begin{cases} x_i \sin(\frac{\pi}{36}(i+\frac{1}{2})) & 0 \sim 17 \\ x_i & 18 \sim 23 \\ x_i \sin(\frac{\pi}{12}(i-18+\frac{1}{2})) & 24 \sim 29 \\ 0 & 30 \sim 35 \end{cases} \quad \text{, for i =} \tag{2.4}$$

c) block_type=3 (stop window)

$$z_i = \begin{cases} 0 & 0 \sim 5 \\ x_i \sin(\frac{\pi}{12}(i-18+\frac{1}{2})) & 6 \sim 11 \\ x_i & 12 \sim 17 \\ x_i \sin(\frac{\pi}{36}(i+\frac{1}{2})) & 18 \sim 35 \end{cases} \quad \text{, for i =} \tag{2.5}$$

d) block_type=2 (short window)

$$z_i = x_i \sin(\frac{\pi}{12}(i+\frac{1}{2})) \quad \text{, for i = 0 ~ 11} \tag{2.6}$$

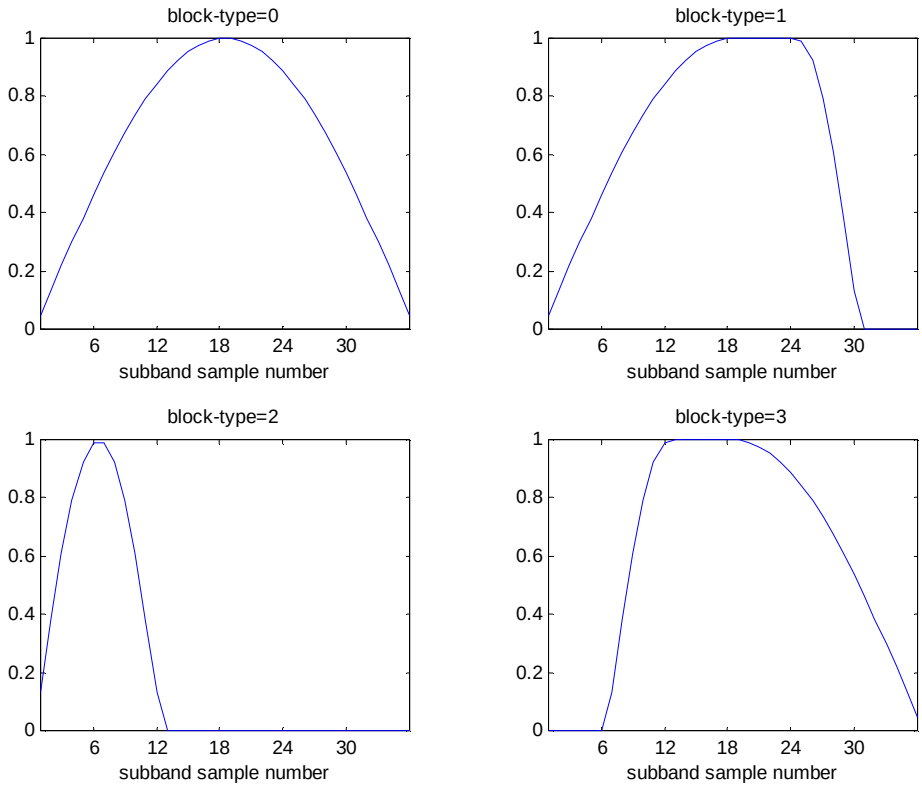

Figure 2.6 Illustration of the four applicable window types.

◆ Alias Reduction

Before passing the frequency lines a reduction of the aliasing introduced in the analysis polyphase filter bank is removed. The aliasing is removed at this early stage in order to reduce the amount of information for transmission. The reduction is obtained by means of a series of butterfly computations, see Figure 2.7. The cs_i and ca_i constants are tabulated in standard [16]. The butterfly operations with appropriate weighting cancel the alias caused by the overlap of two adjacent overlapped subbands.

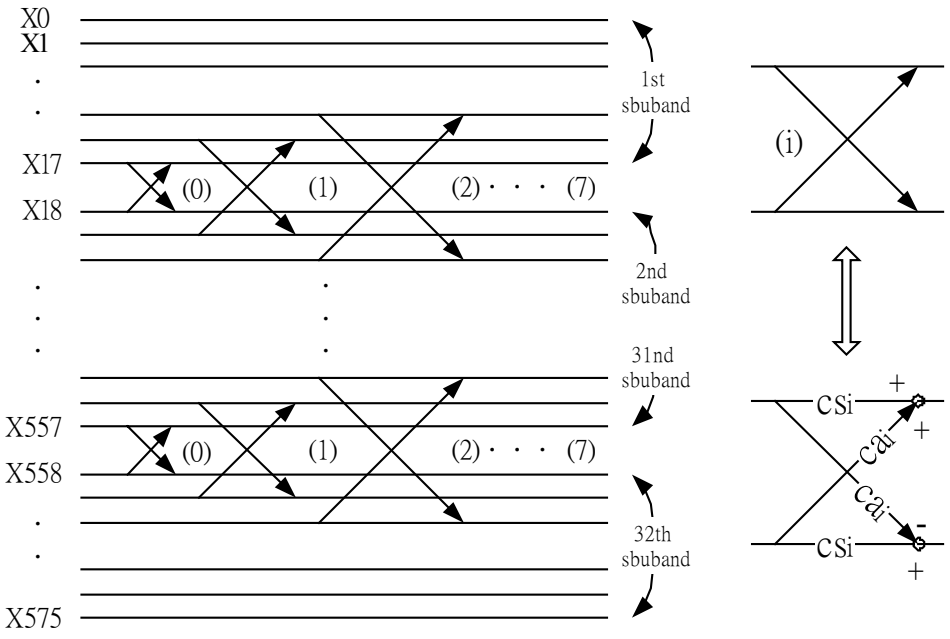

Figure 2.7 Illustration of alias reduction butterflies.

2.1.3 Psychoacoustic Model

It is apparent that while we can hear a very silent sound like a needle falling, and easily a very loud noise like an airplane taking off, it is impossible to discern the falling needle if we hear the airplane at the same time. This phenomenon shows that hearing system adapts dynamic variations in the sound, and some tone we will not hear.

The psychoacoustic model is a pattern that simulates the human sound perceptional system. The model is used in the encoder only to decide which parts of the audio signal are acoustically irrelevant and which parts are not, and removing the inaudible parts. It takes advantage of the inability of human auditory system to hear quantization noise under conditions of auditory masking. This masking is a perceptual property of the human auditory system that occurs when the presence of strong audio signal makes a

temporal or spectral neighborhood of weaker audio signals imperceptible. The results of the psychoacoustic model are utilized in the MDCT block and in the nonuniform quantization block.

Auditory masking consists of three masking principles, which being described below:

◆ Absolute Threshold of Hearing

The absolute threshold of hearing is characterized by the minimum amount of energy needed in a pure tone such that it can be detected by a listener in a quiet environment. If we measure the energy of a number of tone frequencies, the relation curve can be plotted on a graph like Figure 2.8 [22]. Since the listener has no a prior knowledge regarding actual playback levels, the energy values, sound pressure level (SPL) are expressed in terms of decibels (dB), with the value of 0 dB assigned to the weakest energy in +/- 1 bit that can be heard. In this figure, tones in the neighborhood of 3,000 Hz require least intensity to be heard. As a result, their threshold is expressed as 0 dB and all other values are expressed relatvie to this value. The absolute threshold of hearing is also called the threshold in quiet. This absolute threshold of hearing varies with frequency and covers a dynamic range of more than 60 dB, as shown in this figure.

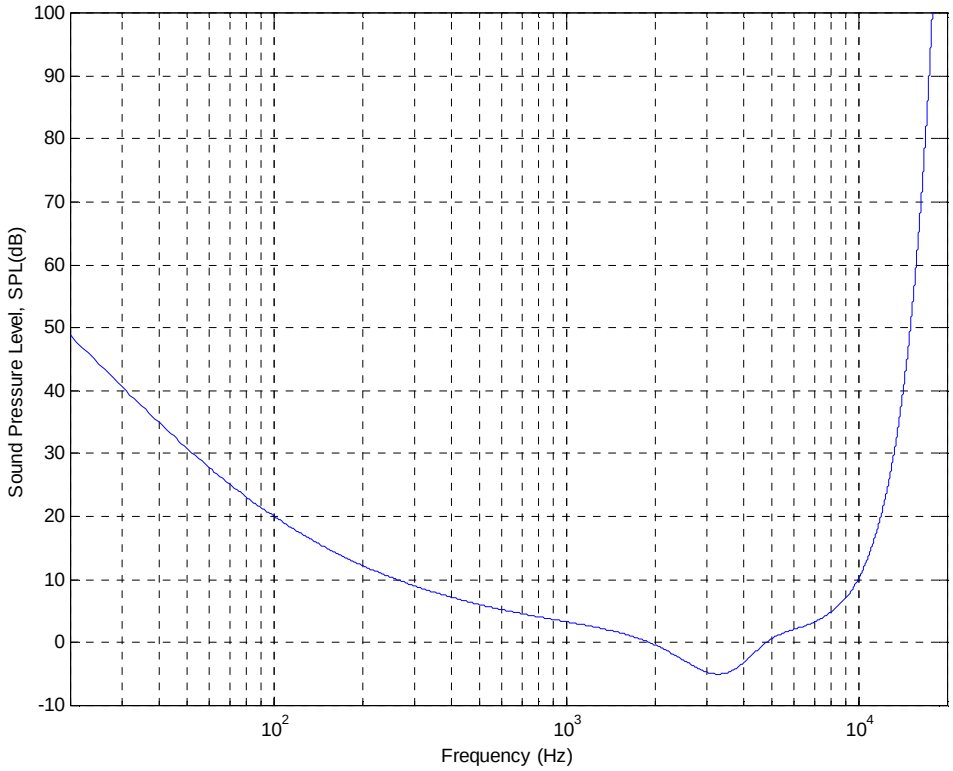

Figure 2.8 The absolute threshold of hearing.

◆ Frequency Masking

Frequency masking, also called simultaneous masking, is a frequency domain phenomenon where a low-level signal (the maskee) can be made inaudible by a simultaneously occurring stronger signal (the masker) as long as masker and maskee are close enough to each other in frequency. A frequency masking threshold can be measured below which any signal will not be audible. The masking threshold depends on the sound pressure level and the frequency of the masker. Take an example of the masking threshold for the SPL = 80 dB narrowband masker in Figure 2.9: the masker is the signal S_0, and any signal's energy under the border of this masking threshold will be

masked by the presence of S_0. The weaker signals S_1 and S_2 are completely inaudible. This is because their individual sound pressure levels are totally below the masking threshold. The signal S_L is only partially masked and the perceivable portion of the signal is above the masking curve. Thus, it is possible to increase the quantization noise in the subband containing the signal S_L up the level AB, which means that fewer bits are needed to represent the signal in this subband.

Without any masker, a signal is also inaudible if its sound pressure level is belw the absolute threshold. The psychoacoustic model supplies the nonuniform quantization block with information about how to quantize the frequency lines. The quantization of the frequency lines is adapted to the limitations of the human ears perception.

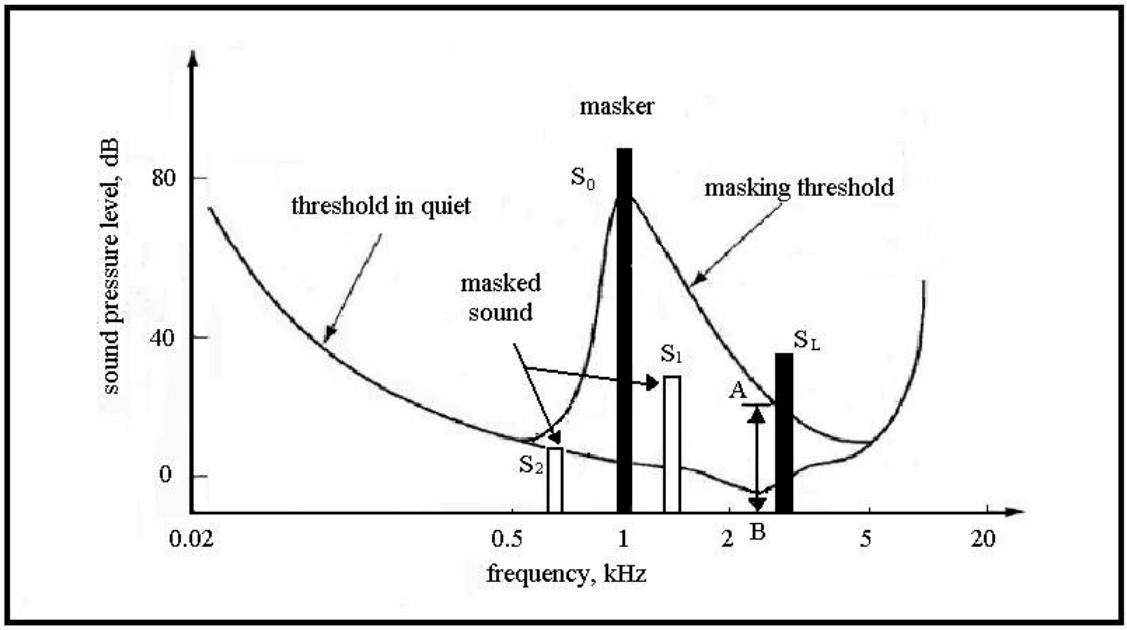

Figure 2.9 Frequency masking threshold and threshold in quiet [23].

◆ Temporal Masking

In addition to simultaneous masking in frequency domain, the temporal masking, also called nonsimultaneous masking, plays an important role in human auditory perception in time domain. It may occur when two sounds appear within a small interval of time. The stronger sound may mask the weaker one, even if the maskee precedes the masker.

Two temporal masking effects occur before and after a strong sound. If a sound is masked after a louder sound it's called post-masking, and if it's masked ahead in time it's called pre-masking, as Figure 2.10 shown. Signal in the dark areas will be masked. Note that in Figure 2.10, post-masking uses a different time lasted longer than pre-masking. Post-masking continues more than 160 ms after the masker while pre-masking only acts 20 ms befor the masker.

Pre-masking can help to mask the appearance of pre-echoes. Consider the case where a silent period is followed by a percussive sound, such an transient sound cause large instantaneous quantization errors. These pre-echoes can become distinctly audible, especially at low bit rates. The effect of pre-echoes can be mitigated by the time domain effect of pre-masking if the time spread is of short duration. Take an example of 44.1 kHz sampling rate, the most common used, 1152 samples stand for about 26.1 ms. The duration of pre-masking effect is about 20 ms. Quantization errors of the 1152 samples spread in time over the pre-masking can mask and become audible. If we use smaller transformation block, the quantization errors can be limited in a smaller time duration. Using short MDCT block transformation, which is 3 times shorter than long block, the

quantization errors will spread in 8.7 ms. Obviously, the duration is less than the time of pre-masking appearance and pre-masking effect will mask the quantization errors.

Since the post-masking effect extends over 160 ms, even in long window case, the quantization errors of a transient sound will not be heard. Both pre-masking and post-masking are being exploited in the MPEG/Audio Layer 3 encoding algorithm.

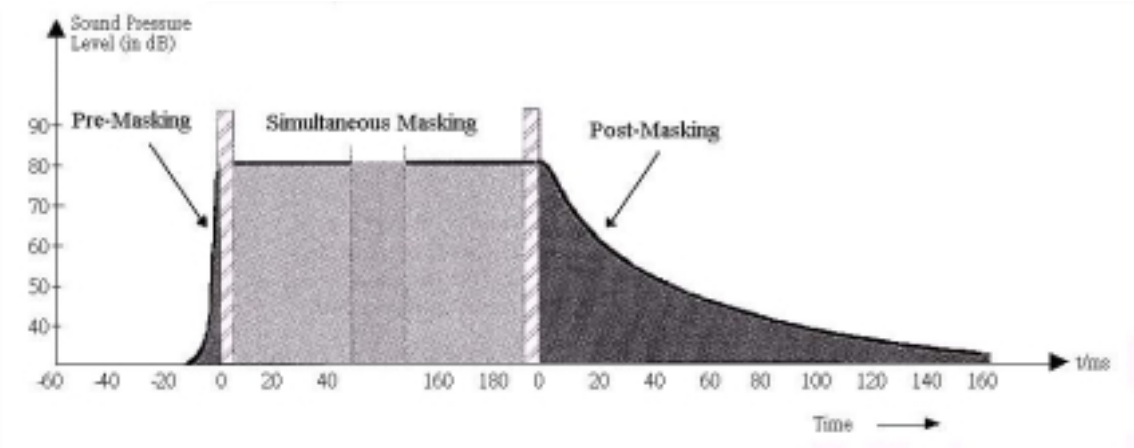

Figure 2.10 Temporal masking threshold [23].

2.1.4 Nonuniform Quantization

The nonuniform quantization block which received the frequency line from the MDCT block and window switching, masking informations from the psychoacoustic model, performs the important key techniques “quantization” and “Huffman coding”. This block outputs the coded data satisfied human auditory system and their correlative side information.

The nonuniform quantization loop is the most time consuming part of MPEG/Audio Layer 3 encoding algorithm. It depends on the variation of audio signal

and does not have a fixed execution time. The more party-colored the signal is, the more encoding time it needs. The description of the Layer 3 loop module is subdivided into three levels. The top level is called “loops frame program”. The loops frame program calls a subroutine named “outer iteration loop” which calls the subroutine “inner iteration loop”. For each level a corresponding flowchart is shown below.

The loop module, as Figure 2.11 shown, quantizes an input data vector of spectral lines in an iterative process according to several demands. The inner loop quantizes the input data and increases the quantizer step size until the output data can be coded with the available amount of bit. After completion of the inner loop, an outer loop checks the distortion of each scalefactor band and, if the allowed distortion is exceeded, amplifies the scalefactor and calls the inner loop again.

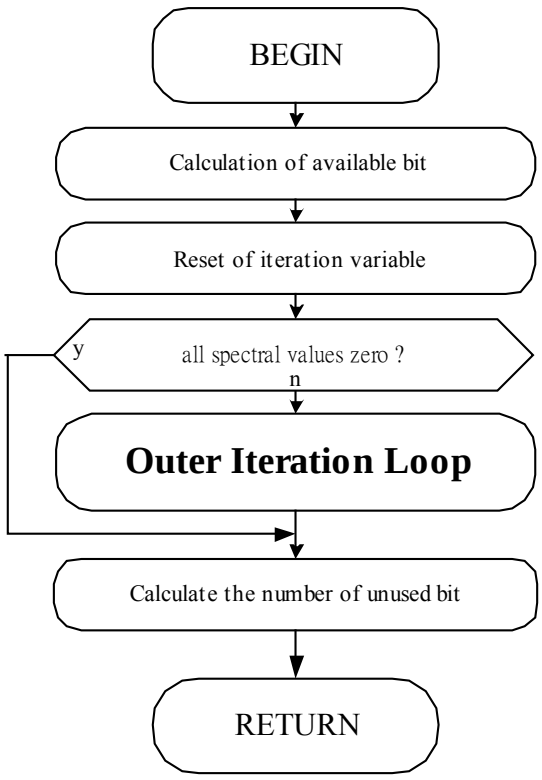

Figure 2.11 MPEG/Audio Layer 3 loops frame program [16].

◆ Outer Iteration Loop (distortion control loop)

The outer iteration loop controls the quantization noise which is produced by the quantization of the frequency domain lines within the inner iteration loop. The coloration of the noise is done by multiplication of the lines within scalefactor bands with the actual scalefactors befor doing the quantization. If the quantization noise is found to exceed the masking threshold, the scalefactor for this band is adjusted to reduce the quantization noise. The outer loop is executed until the actual noise is below the masking threshold for every scalefactor band. Figure 2.12 shows the flowchart of outer loop.

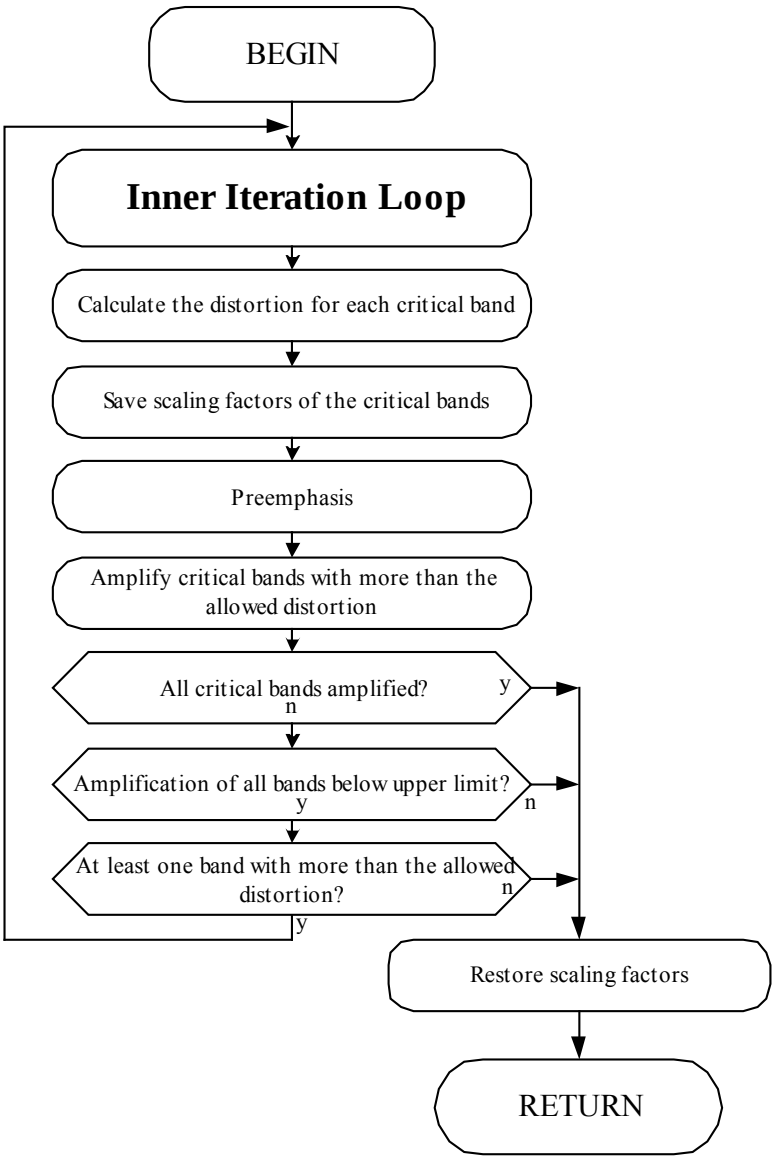

Figure 2.12 MPEG/Audio Layer 3 outer iteration loops [16].

◆ Inner Iteration Loop(rate control loop)

The inner iteration loop does the actual quantization of the frequency domain data and prepares the formatting operation. The Huffman code tables assign shorter code words to smaller quantized values. If the number of total bits of resulting from the Huffman coding operatoin exceeds the number of bits available to code one frame, this

can be corrected by adjusting the global gain to result in a larger quantization step size, leading to smaller quantized value. This operation is repeated with different quantization step sizes until the resulting number of bits demand for Huffman coding is small enough. Figure 2.13 shows the detail flowchart of the inner loop.

Except scaling, quantization and Huffman coding operation, the Huffman table selection, subdivision of the big_value range of subregions and the selection of the quantizer step which will be introduced in the next subsection also take place here.

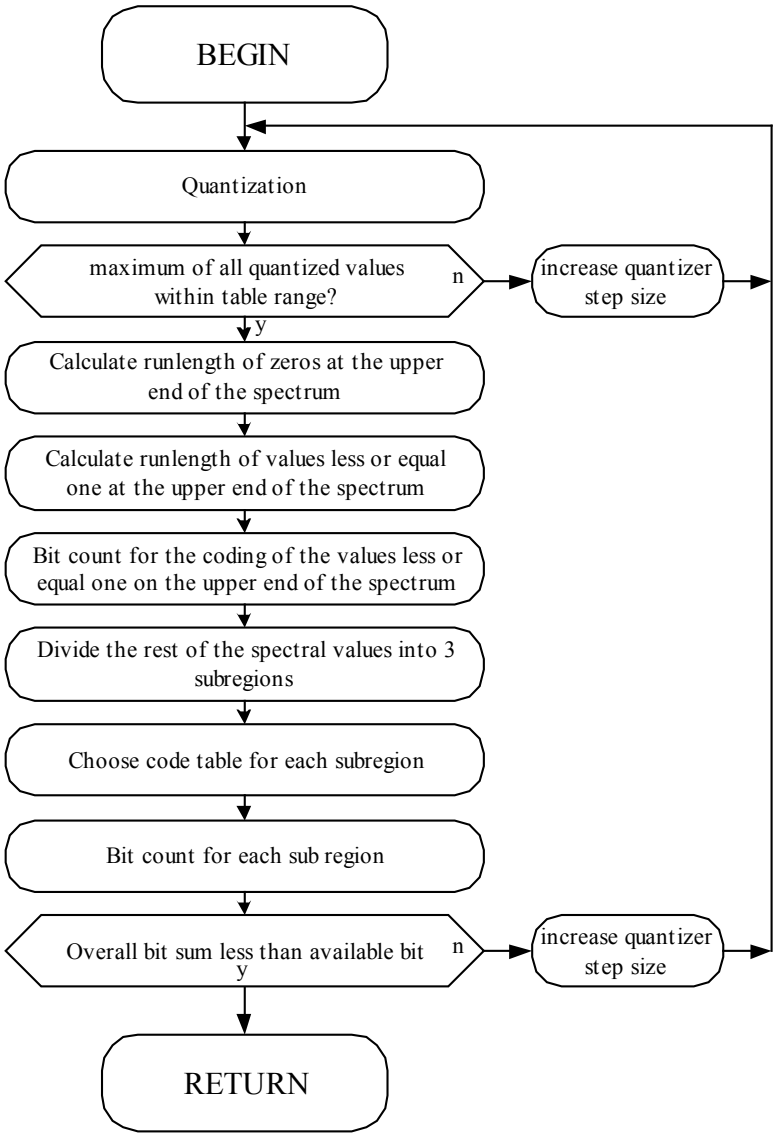

Figure 2.13 MPEG/Audio Layer 3 inner iteration loops [16].

2.1.5 Huffman Encoding

In this block an entropy coding of the quantized frequency lines is performed using the Huffman coding algorithm based on 32 static Huffman tables. The Huffman coding

provides lossless compression and thereby reduces the amount of data to be transmitted without the quality loss.

The most popular technique for removing coding redundancy is Huffman coding. In Huffman coding the entropy is based on a statistic distribution of the group of data values. From the data statistics a substitution table covering all data values is established. In this table, values with a high probability of being present in the data are associated with short code words and data rarely present are associated with longer code words. Thereby, the Huffman coding is a variable-length coding (VLC).

MPEG/Audio Layer 3 delimits the frequency lines into three sections and adopts an ESCAPE value in one of the three section in the coding process for two reason:

1.The order is by increasing frequency except for the short MDCT block mode. For short block there are three sets of window values in a subband so the ordering is by frequency, then by window, then by scalefactor. Ordering is advantageous because large values tend to be at the lower frequencies and long runs of zero or near-zero values tend to be at the higher frequencies.

2.When a large number of symbols is to be coded, the construction of the optimal binary Huffman code table is a nontrivial task.

With the benefit for the first reason, the encoder delimits the ordered frequency lines into three distinct regions. This enables the encoder to code each region with a different set of Huffman tables specifically tuned for the statistics of that region. Three region are called “rzero”, “count1_region” and “big_value region”.

Starting at the higher frequency, the encoder identifies the continuous run of all-zero values as one region, “rzero”. This region does not have to be coded because its

size can be deduced from size of the other two regions. However, it must contain an even number of zeroes because the other regions code their values in even numbered groupings.

A second region, “count1_region”, comprises of a continuous run of values consisting only of -1, 0, or 1. Two Huffman tables for this region code 4 values at a time so the number of values in this region must be a multiple of 4.

Finally, a third region covers all the remaining values, called “big_values region”. The 30 Huffman tables for this region code the values in pairs. This region is further subdivided into three subregions that each has its own specific Huffman table. In the “big_values region”, due to the disadvantage of the second reason in page 28, an “ESCAPE” value is introduced in order to improve the coding efficiency before coding the frequency lines. In this region, values exceeding 15 are repreesented with the number 15 and the remainder is the ESCAPE value. Depending on the size of the ESCAPE value a number of bits, called linbits, is assigned to represent the ESCAPE value, see the following Equation (2.7).

ESCAPE value = quantized value -15, if quantized value > 15 (2.7)

linbits = word length(ESCAPE value)

Figure 2.14 shows the relation of three Huffman coded region and scalefactor. Table 1 lists the characteristic of the 32 Huffman tables of MPEG/Audio Layer 3 standard.

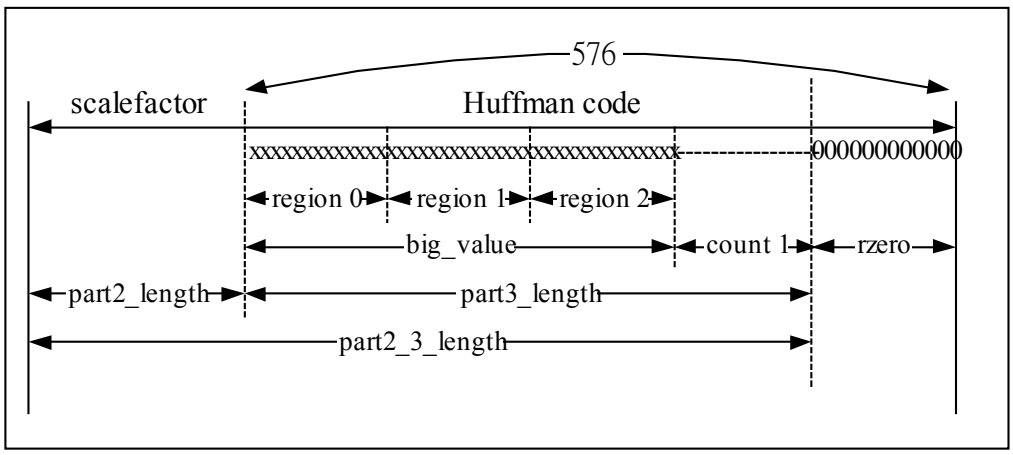

Figure 2.14 Main data organization.

Table 1 Characteristic of 32 Huffman tables.

Table index	Max. value	Table index	Max. value	linbits	Max. value *	Region used
A	1	B	1	No		count_1
0	0	16	15	1	16	
1	1	17	15	2	19	
2	2	18	15	3	23	
3	2	19	15	4	31	
4	not used	20	15	6	79	
5	3	21	15	8	271	
6	3	22	15	10	1039	
7	5	23	15	13	8207	Big value
8	5	24	15	4	31	
9	5	25	15	5	47	
10	7	26	15	6	79	
11	7	27	15	7	143	
12	7	28	15	8	271	
13	15	29	15	9	527	
14	not used	30	15	11	2016	
15	15	31	15	13	8207	

* means the addition of maximum vaule and ESCAPE value

2.1.6 Bitstream Formatting

The last block of encoding process is to produce a MPEG/Audio Layer 3 compliant bitstream. The Huffman coded frequency lines, the side information and a frame header are assembled to form the bitstream. The bitstream is partitioned into frames each represents 1152 audio samples. The header describes which bit rate and sampling frequency that is being used for the encoded audio. The side information tells what block type, Huffman tables, subband gain and subband factors are being selected.

◆ Bit Reservoir

In this block, an enhancement method called “bit reservoir” is used to fit the encoder’s time-varying demand on code bits. The encoder can donate bits to a reservoir when it needs less than the average number of bits to code a frame. Next, when the encoder needs more than the average number of bits to code a frame, it can borrow bits from the reservoir mechanism. The encoder can only borrow bits donated from past frames; it cannot borrow from future frames. The MPEG/Audio Layer 3 bitstream uses a 9-bit pointer, called main_data_begin, in each frame's side information to point out the location of starting byte of audio data for that frame. An example of how the main data can be distributed is illustrated in Figure 2.15.

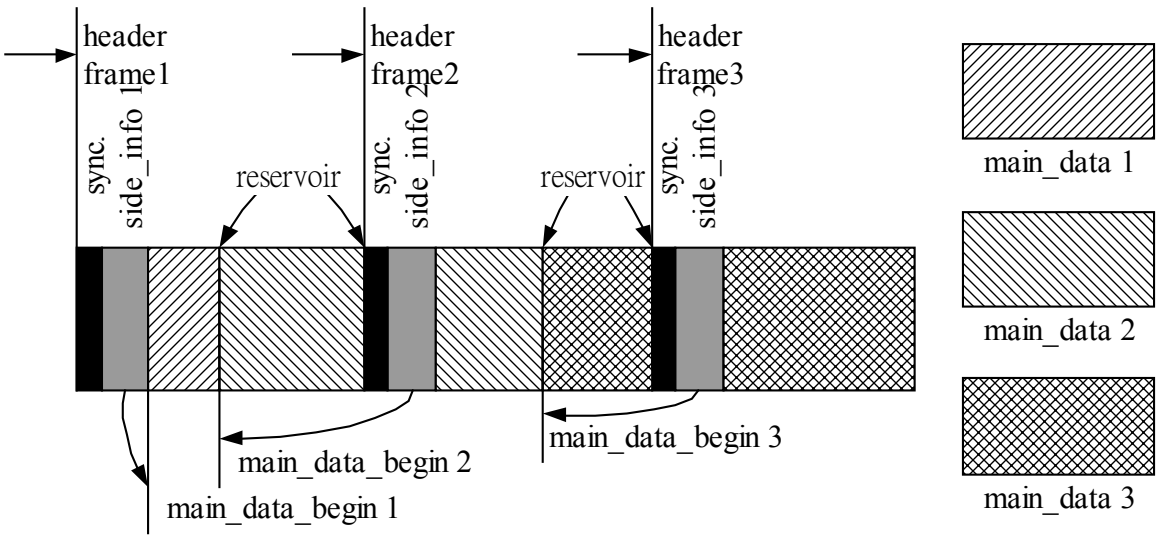

Figure 2.15 An example of bit reservior [16].

2.2 MPEG/Audio Layer 3 Decoding Algorithm

In this section the MPEG/Audio Layer 3 decoder will be described with its functionality. The decoding process is based on the block diagram in Figure 2.16. The decoder has three main parts: “Decoding of Bitstream”, “Inverse Quantization”, and “Frequency to Time mapping”.

The input coded bitstream is passed through the first parts to synchronize and extract the quantized frequency line and other information of each frame. The inverse quantization part dequantizes the frequency line according to the output of previous part. Finally, the last part is a set of reverse operations of the MDCT and analysis polyphase filter bank in the encoder. Its output is the audio signal in PCM format.

In the following subsections, we will describe the opeation and the functionality for each block in Figure 2.16.

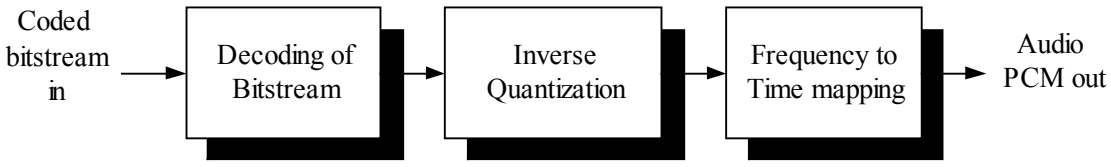

Figure 2.16 MPEG/Audio Layer 3 decoder block diagram.

2.2.1 Decoding of Bitstream

This decoding part effects to synchronize and extract the quantized frequency lines and other information of each frame. Of course, it needs to synchronize where a frame begins and where the data resides. The block diagram of this part is shown in Figure 2.17 and will be discussed as following.

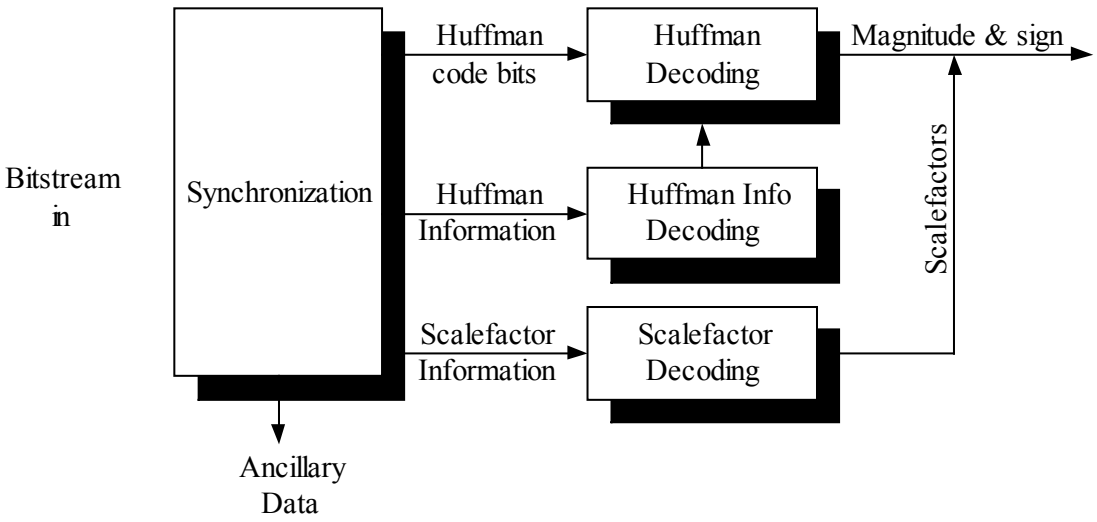

Figure 2.17 Decoding of bitstream block diagram [24].

◆ Synchronization

The purpose of this block is to receive the incoming bitstream, identify the contents of the bitstream and pass the information onto the succeeding blocks in the decoder.

a) The Format of the Bitstream:

The contents of a MPEG/Audio bitstream is organized into frames, each contains information to reconstruct the audio PCM samples. A frame consists of four parts: header, side information, main data, and ancillary data.

b) Header:

The header part of the frame contains a synchronization word and system information. To be able to detect the beginning of a new frame, each frame starts with a 12 bit synchronization word. The rest of the header describes the type of frame, that is which layer is used in the frame, which bitrate is used for transmission, the sampling frequency of original digital audio signal, whether the audio signal is single channel or dual channel, and other additional informations. Figure 2.18 shows the MPEG/Audio Layer 3 header format.

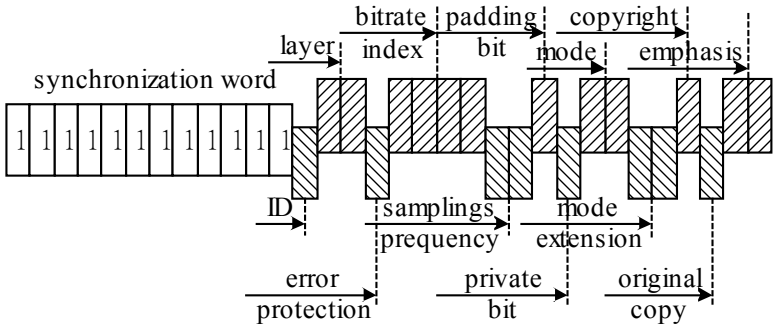

Figure 2.18 MPEG/Audio Layer 3 header format [24].

c) Side Information:

The side information sectoin in the frame contains the necessary information to decode the main data. The side information contains information concerning which Huffman tables to use during the Huffman decoding process, and information of scalefactors. The side information section also contains information about where the main data begins due to the "bit reservoir" technique described in previous section 2.1.6.

d) Main Data:

The main data section contains the coded scalefactor value and the Huffman coded data. See Figure 2.14 in section 2.1.5.

e) Ancillary Data:

It is possible to include an ancillary data section in each frame. The format of this ancillary data is user defined and can be used for, e.g. the title, the artist and the album of the song. The ancillary data is placed between the end of the main data bits in one frame and the start of the main data bits in the next frame.

◆ Huffman Decoding

In this block the decoding of the Huffman code bits is performed. Since the Huffman coding is a VLC method, a single code word in the middle of the Huffman code bits cannot be identified without starting to decode from a point in the Huffman code bits known to be the start of a code word [24].

◆ Huffman Info Decoding

The Huffman Info Decoding block serves to setup all the parameters necessary for the Huffman decoding block to perform a correct Huffman decoding. The first task to

perform is to collect data in the side information which describs the characteristics of the Huffman code bits. This includes where to find the Huffman code bits in the bitstream, to decide which Huffman tables are used in each region and whether ESCAPE values are present in the Huffman code bits. Moreover, this block must make sure that all frequency lines are generated regardless of how many frequency lines are described in the Huffman code bits. When fewer than 576 frequency lines appear, the Huffman Info Decoding block must perform a zero padding to fill the lack of data.

◆ Scalefactor Decoding

The purpose of the scalefactor decoding block is to decode the coded scalefactors, i.e. the first part of the main data. Input to this block is scalefactor information and coded scalefactors. The output of the block is the decoded scalefactors, to be used in the next inverse quantization block.

2.2.2 Inverse Quantization

The purpose of this block is to reestablish a perceptually identical data of the frequency lines generated by the MDCT block in the encoder. The descaling is based on the scaled quantized frequency lines reconstructed from the Huffman decoding block and the scalefactor reconstructed in scalefactor decoding block. The descaling calculation of the frequency lines is shown in Equation (2.8).

$$xr[i] = sign(is[i]) \bullet abs(is[i])^{\frac{4}{3}} \bullet \frac{2^{\frac{1}{4}(global\_gain-210-8sbg[i])}}{2^{scalefac\_multiplier*(sf[i]+preflag*pt[i])}} \tag{2.8}$$

where:
is[i] is the frequency line reconstructed by Huffman decoder,

global_gain, sbg[i], scalefac_multiplier, sf[i],preflag,pt[i] are from scalefactor decoding.

2.2.3 Frequency to Time Mapping

This decoding part performs to reproduce the audio signal from the dequantized frequency line. This part contains several sub-blocks as Figure 2.19 shown and will be described in the following.

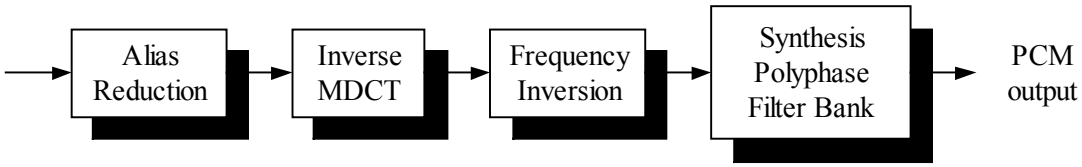

Figure 2.19 Frequency to time mapping [24].

◆ Alias Reduction

In the MDCT block within the encoder it was described that an alias reductoin was applied. In order to obtain a correct reconstruction of the analysis polyphase filter bank in the algorithms to come back, the aliasing artifacts must be added to the decoding process again.

◆ Inverse MDCT

The frequency lines from the alias reduction block are processing through IMDCT block. The analytical expression of the IMDCT is shown in Equation (2.9)

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos\left(\frac{\pi}{2n}(2i+1+\frac{n}{2})(2k+1)\right) \quad \text{, for i = 0 ~ n-1} \tag{2.9}$$

where:

X_k is the frequency line,
n is 12 for show window, and 36 for long window.

◆ Frequency Inversion

In order to compensate the decimation used in the analysis polyphase filter bank, every odd time sample of every odd subband is multiplied with -1.

◆ Synthesis Polyphase Filter Bank

Each time 32 samples, from each of the 32 subbands, are applied to the synthesis polyphase filter bank and 32 consecutive audio samples are calculated. Figure 2.20 illustrates the algorithm and procedure of synthesis polyphase filter bank.

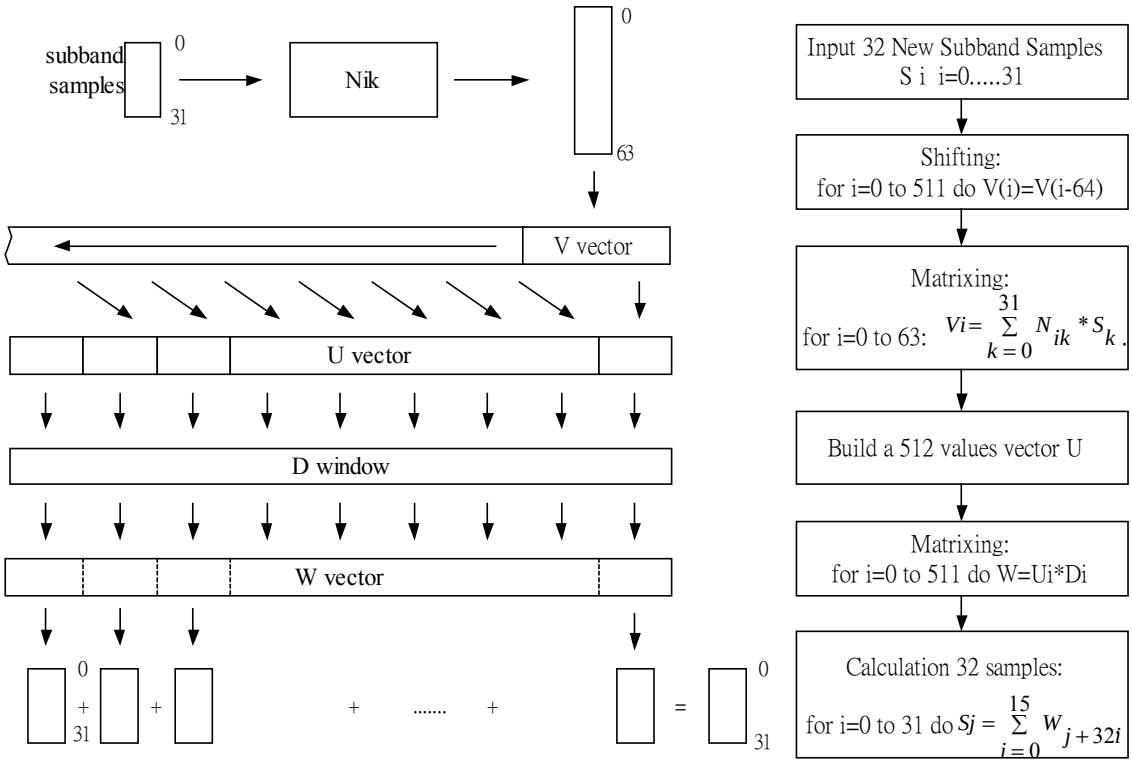

Figure 2.20 Diagram and procedure of synthesis polyphase filter bank.

Chapter 3

Environment of Hardware and Software

In this chapter, we describe the hardware and software environment briefly. The hardware is concerned with the development of programs while the software influences the development speed and enhancement.

3.1 Hardware Environment

The hardware used in this thesis is Texas Instruments™ TMS320C5402 DSK (DSP Starter Kit). DSK5402 provides a low-cost, standalone C54x development platform that enables users to evaluate and develop applications for the C54x DSP. It also provides the DSK schematics that is useful for someone who wants to design the DSP embedded system. DSK5402 contains many components and interfaces including:

- 100 MHz VC5402 DSP
- 64 K SRAM and 256 K FLASH
- 2 AIC (Analog Interface Circuit)
- DAA (Data Access Arrangement) ,the telephone interface
- Microphone and speaker

- RS-232 UART interface
- Parallel and JTAG interface for debug

Figure 3.1 shows the hardware block diagram of DSK5402. For more information about DSK5402, please refer to [25].

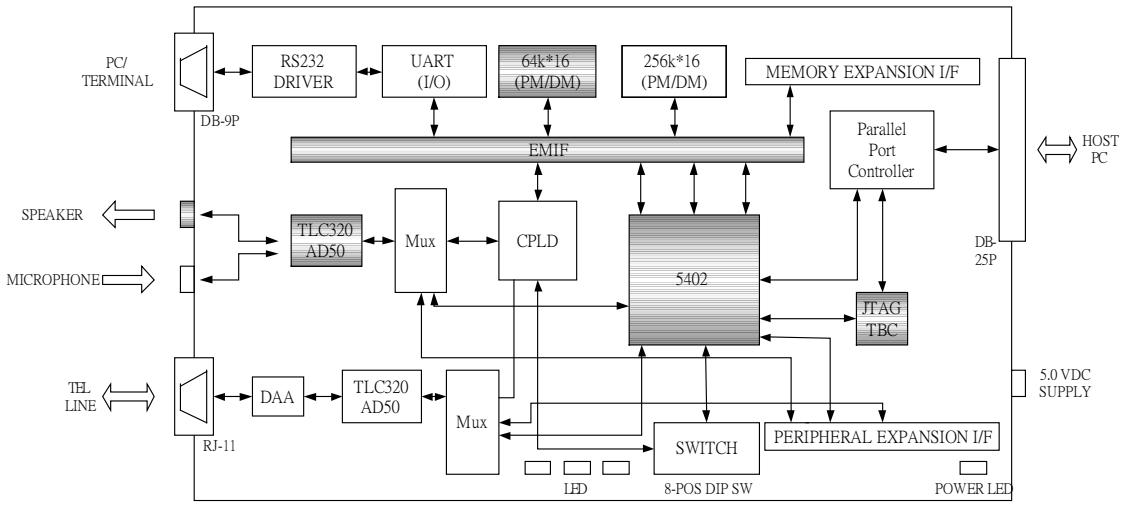

Figure 3.1 DSP Starter Kit's functional block diagram [25].

The kernel of DSK is Texas Instruments™ TMS320VC5402 DSP chip [26]. This chip is a 16 bits, fixed-point DSP with specific hardware logic, on-chip memory, on-chip peripherals, and a highly specialized instruction set. All operations are executed on this chip, includeing the bitstream access, conditional control, vector, matrix and filter operations. Figure 3.2 shows the architecture of C54x DSP.

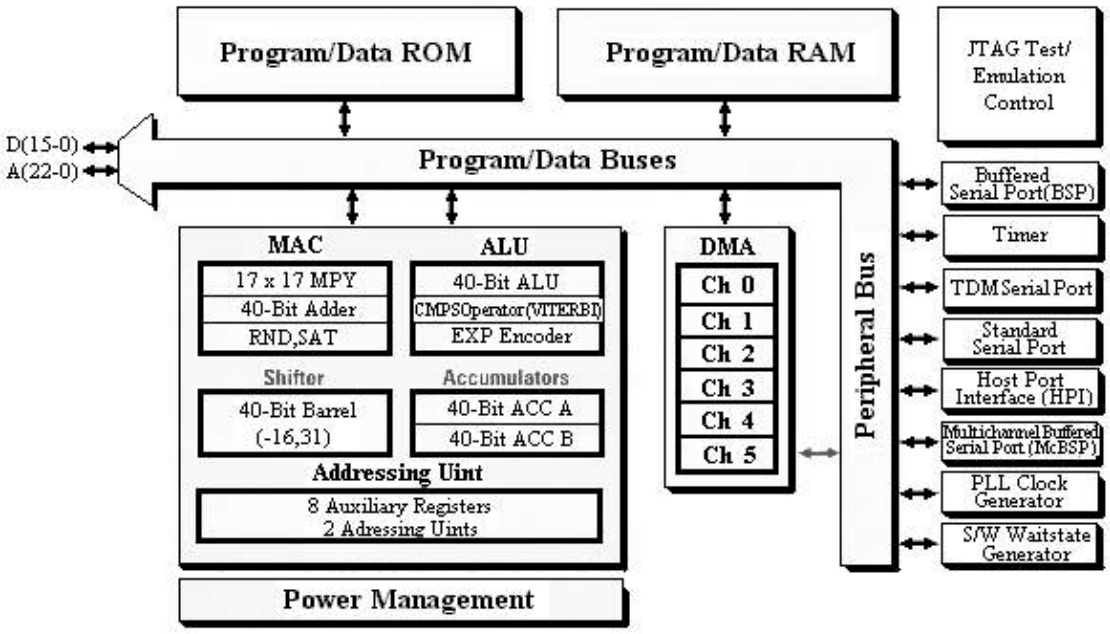

Figure 3.2 Architecture of TMS320C54x DSP [26].

In general, C54x DSP have a total memory space of 192K 16-bit words. This space is divided into three specific memory segments: 64K words of program, 64K words of data, and 64K words of I/O. The parallel structure of the C54x architecture and the dual-access capability of the on-chip DRAM allow the C54x four concurrent memory operations. There are several advantages of operating from on-chip memory:

- Higher performance because no wait states are required.
- Lower cost than external memory.
- Lower power than external memory.

The main advantage of operating from off-chip memory is the ability to access a large memory space. Since this hardware is a DSK board, its program and data use the same 64K memory without separating program and data memory. Figure 3.3 shows the memory maps for the C5402 DSP and DSK's external memory.

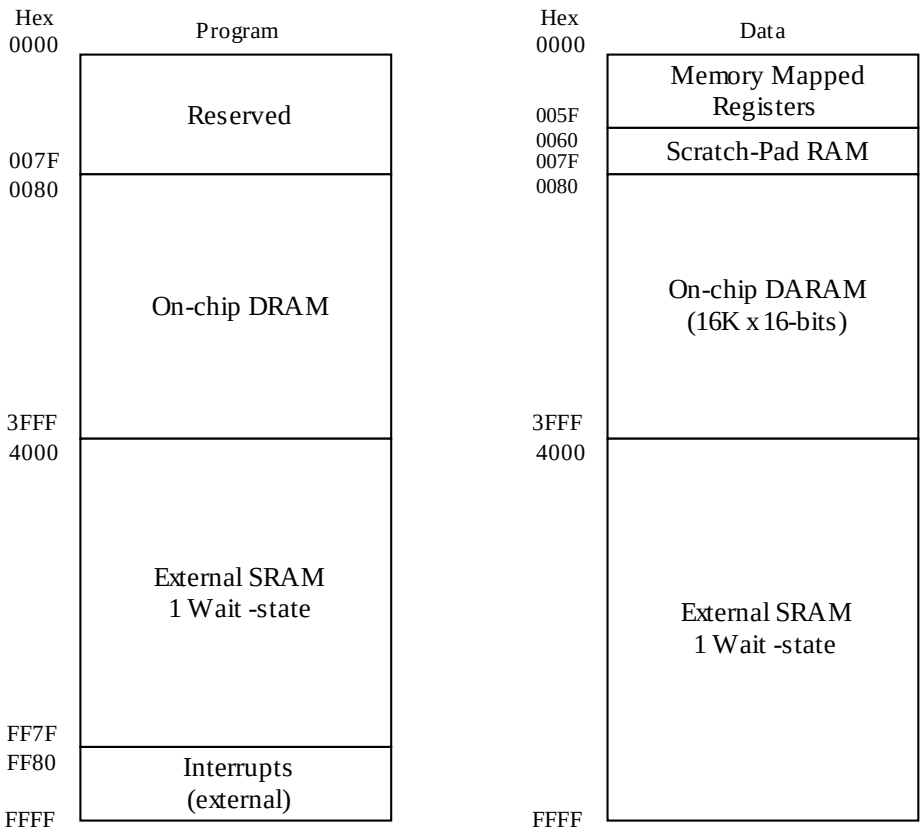

Figure 3.3 Memory maps of C5402 and external memory [26].

3.2 Software Environment

The DSP software offers Integrated Development Environment (IDE) tools, called Code Composer Studio® which user can develop quickly and debug easily in programming stage. Figure 3.4 shows the IDE software environment. This software development support enables user to develop DSP applications that can be loaded and executed on the C54x DSK. DSP applications can easily be developed with the use of high-level DSP board control and on-board peripheral interface functions. This DSP

support package allows users to quickly develop DSP applications for evaluation of the DSK or the application.

The DSP source code debugging support consists of a debugger driver compatible with TI's Code Composer debugger. DSP source code debugging support enables us to load, execute, and test DSP applications in their native C or assembly language source code formats. The debugging environment gives visibility into the operation of DSP applications by:

- Supporting execution control with single-stepping and breakpoint capabilities.
- Enabling code to be profiled for performance monitoring.

For more information of source code debugging, please refer to [27][28].

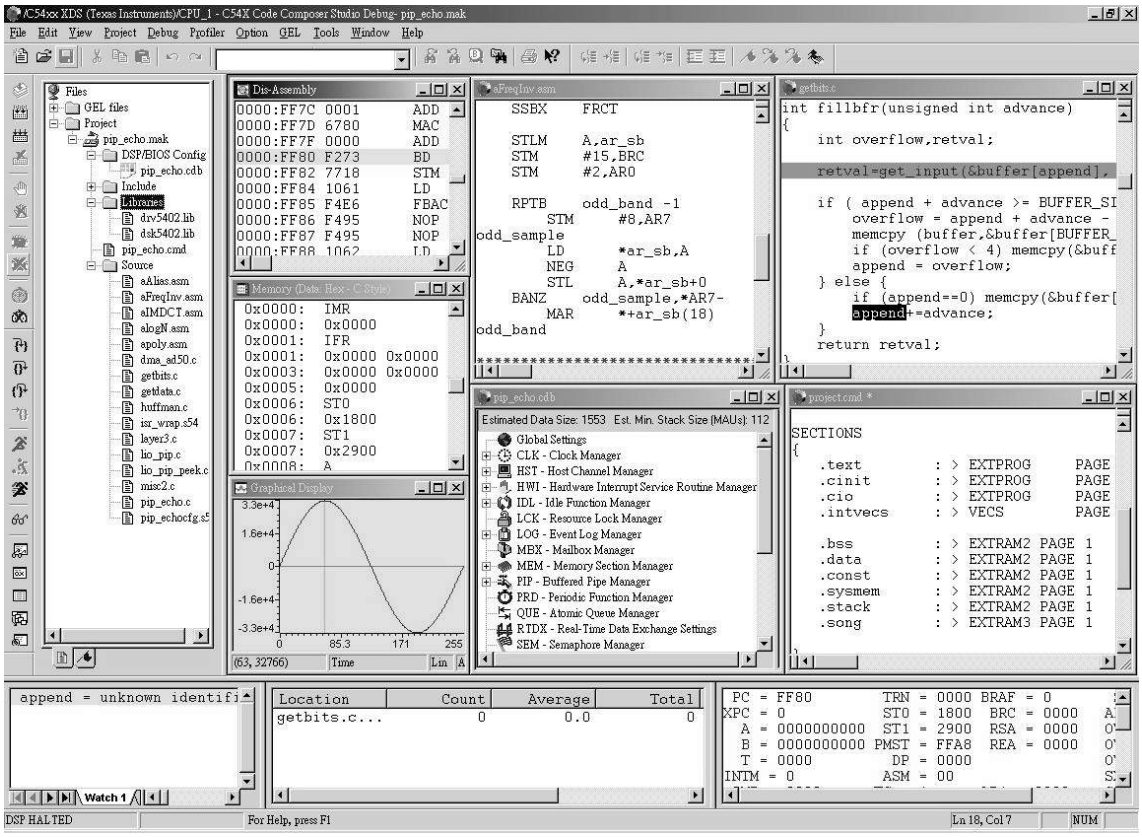

Figure 3.4 Software IDE environment.

Figure 3.5 shows a software development flow used in this thesis. Through three stages: C complier, Assembler and Linker, it generates the executable COFF (Common Object File Format) file.

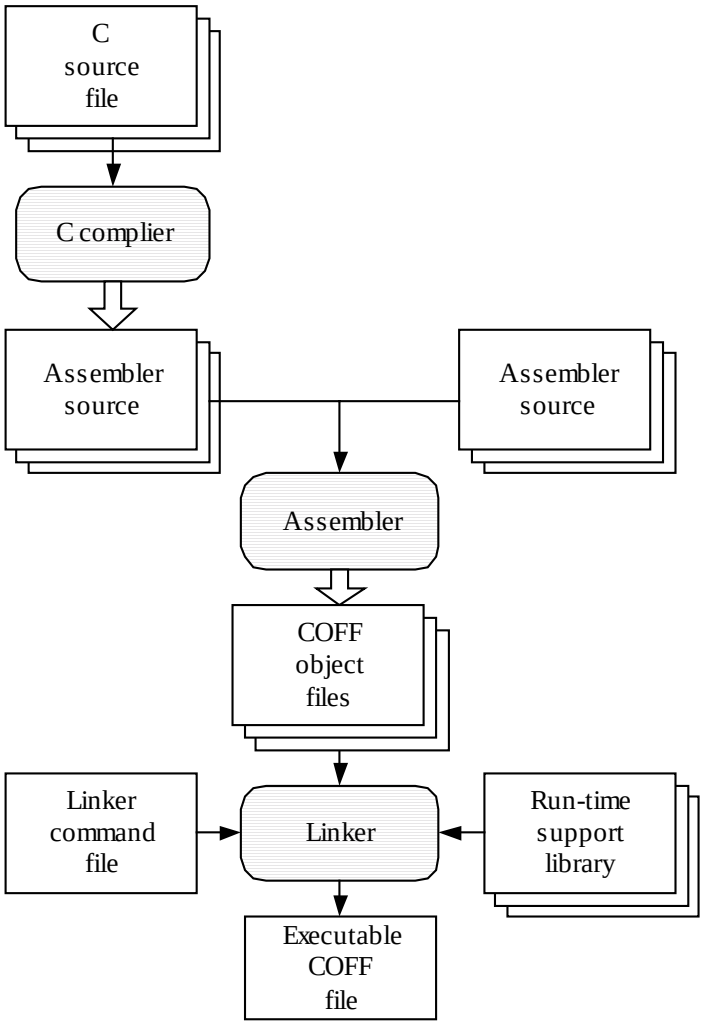

Figure 3.5 Software development flow [27].

Chapter 4

Implementation and Verification

MPEG/Audio Layer 3 decoder had been accomplished based on Win32 or Unix-like OS several years ago. But there is a few decoder based on single chip like DSPs or FPGAs. For example, there are shareware WinAmp® and Winplay® for Win32 system and freeware FreeAmp® for Linux. But we can't find any decoder based on DSPs except from commerce, especially for fix-point DSPs. Since the PC or workstation has enough large memory and fast execution speed, the programmer need to concentrate on the algorithm only. Referencing the standard C code can implement such decoder straightly and easily. In other way, implementation on DSP chips not only has to using the not-user-friendly assembly language but also has to coding in consideration of memory and speed.

Section 4.1 describe the different of implementation between PC and DSP, including handling the fixed-point operation, an efficient algorithm of IMDCT and synthesis polyphase filter bank, and multi-task, multi-thread management. Section 4.2 exhibits the performance of this fixed-point DSP based decoder.

4.1 Implementation

4.1.1 Fixed-Point Operation

Although this fixed-point DSP can support floating-point operations as well, it calls for more instructions cycles and memories. Accommodate to the architecture of this fixed-point DSP chip, all mathematics are handled by fixed-point operation in the decoder. Fixed-point operations have to consider the dynamic range of all mathematics, scaling operation and overflow situation. This subsection will discuss those consideration block by block in detail.

◆ Synchronization and Side Information Decoding

The two blocks just synchronize each frame and get the information which is in the integer range. We do not discuss their fixed-point operation since synchronization and side information decoding contain only logical operation and integer mathematics. There are no any scaling operation and overflow situation.

◆ Scalefactor and Huffman Decoding

Scalefactor decoding extracts the scalefactor used for the inverse quantization. Its dynamic range is 0 to 15 which expressed by one to four bits.

Refer to Table 1, Huffman decoding operation decodes the encoded quantized value and the maximum value is 8207. The dynamic range of Huffman decoding value influences the inverse quantization block. These two block use integer operations without any overflow condition.

◆ Inverse Quantization

Except normal mathematics of addition and multiplication, the decoder algorithm also contains logarithmic and exponential operations, e.g. the inverse quantization. The inverse quantization equatoin, Equation (2.8), can be manipulate into a familiar equation:

$$Xr[i] = sign(is[i]) * is^{4/3} * 2^{\exp} \tag{4.1}$$

where –88.5 ≦ exp ≦ 11.5, 0 ≦ is[i] ≦ 8207

Since the characteristic of audio signal and encoder, exp and is[i] will manipulate in good relative value. The dynamic range of Xr will be |Xr| < 1. we can realize this equation on fixed-point operation by using logarithm and exponentiation. First, take the logarithmic operation on Equation (4.1) and derive Equation (4.2).

$$Y[i] = \ln(|Xr[i]|) = \frac{4}{3}\ln(is[i]) + \exp * \ln(2) \tag{4.2}$$

Now the problem is how to calculate ln(is) and compute the final result Xr[i] from Y[i]. The logarithm of Taylor expansion can be written as Equation (4.3).

$$\ln(1-x) = -1 - \frac{x^2}{2} - \frac{x^3}{3} - ..... - \frac{x^n}{n} \tag{4.3}$$

where -1 ≦ x < 1.

Since is[i] is an integer, it should be scaled to an appropriate range as Equation (4.4) and (4.5) before using the Taylor's equation.

$$is = C(1-x) \tag{4.4}$$

where $C = 2^N$.

$$\ln(is[i]) = \ln(C) + \ln(1-x) = N*\ln(2) + \ln(1-x) \quad (4.5)$$

Experimental results show that for n=11 in Equation (4.3), a good audio decompression can be achieved [29]. The maximum value of is[i] is 8207, so the dynamic range of ln(is) will be $|\ln(is)| < 16$, and Q11 format is sufficient for this operation.

After getting Y from the logarithmic operation, we calculate Xr[i] from Y[i] by performing exponential operations. The exponentiation can be written as Taylor expansion as Equation (4.6).

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!} \quad (4.6)$$

where $-\infty < x < \infty$

The Y[i] given by logarithmic calculatoin is in Q11 format. Therefore, the MSB5 bits are integer and LSB11 bits are fractional part. It is convenient to rewrite the equation as

$$Xr[i] = e^{Y[i]} = e^{lk} * e^x \quad (4.7)$$

Now look into the fraction part, if x is smaller than 0.5, the Taylor equation converges quickly. If it is larger than 0.5, it converges very slowly. Therefore, if x is larger than 0.5, Equation (4.7) should be rewritten as

$$Xr[i] = e^{Y[i]} = e^{lk} * e^x = e^{lk+1} * e^{x-1} \quad (4.8)$$

After expansion, we can implement the above equations easily by using the “poly” instruction which is supported by C54x DSP. This instruction is useful for polynominal evaluation to implement computations that take one cycle per monomial to execute.

Finally, we get the Xr[i] in Q15 format which means the dynamic range of Xr[i] is |Xr[i]| <1.

Due to the mixed integer and fraction numbers in original algorithm, we usually have to scale the number in appropriate scaling for fixed-point operation. The fixed-point operation is a common acquaintance, so we will not discuss the detail scaling operations between integer and fraction mode in this thesis. For more information about fixed-point operation, please refer to [26] [30]. In this block, we totally use the Q0 (integer), Q15 (fraction), and Q11(mixed integer-fraction) formats.

◆ IMDCT

The IMDCT Equation (2.9) can be viewed as a convolution of two vectors. Manipulate the IMDCT operation into a familiar Equation (4.9) for long window case.

$$x_i = \sum_{k=0}^{17} XkYki \text{ , for i = 0 ~ 35} \tag{4.9}$$

where $Yki = \cos(\frac{\pi}{72}(2i+19)(2k+1))$

The dynamic range of x_i can derive from Equation (4.10)

$$x_i = \left|\sum_{k=0}^{17} XkYki\right| \le \sum_{k=0}^{17} |Xk||Yki| \le \sum_{k}^{17} |Yki| * \max|Xk| \tag{4.10}$$

Since the dynamic range of Xk and Yki are 0< |Xk|, |Yki| < 1 , the maximum value of x_i is $\sum_{k}^{17} |Yki|$. From the standard, the maximum value of x_i is smaller than 12. If the worst case occures, x_i will overflow and Q15 format will be fail. We assume that in general

case, digital audio will not cause the overflow since the general digital audio does not have tremendous energy. We will discuss the overflow condition later.

◆ Synthesis Polyphase Filter Bank

In this block, only three equation below may make the overflow happen.

$$V[i] = \sum_{k=0}^{31} Nik * Sk \tag{4.11}$$

$$W[i] = U[i]D[i] \tag{4.12}$$

$$S[j] = \sum_{i=0}^{15} W[j + 32i] \tag{4.13}$$

In Equation (4.11), the analysis step is also similar to IMDCT's analysis. Nik is cosine term with dynamic range between –1 and 1. If the worst case occures, V[i] will overflow and Q15 format will be fail.

In Equatoin (4.12), the coefficient of D[i] range from –1.449 to +1.4449. Scaling by 2 changes its dynamic range into pure fractional. The multiplication of two fractional number is still in the fractional range, hence Q15 format handle this operation well with scaling by 2.

In Equation (4.13), the output S[j] has maximum value 32767 and minimum value –32768 for 16 bits processor. This decoder uses the saturation mechanism to prevent the overflow during the decoder operation because overflow is seldom happened except decoding the digital signal with very large energy. If checking overflow in each operation it may occure, decoder will waste much instruction cycle since overflow is seldom happened.

◆ Overflow

Figure 4.1 shows the test pattern that will cause overflow operation due to its large energy. This kind of large energy signals is seldom used in MPEG/Audio coding because it is a sound that grates on the ear and is not comfortable for hearing. That's the reason why most decoders only check the overflow at the output and just perform saturation if it is overflow.

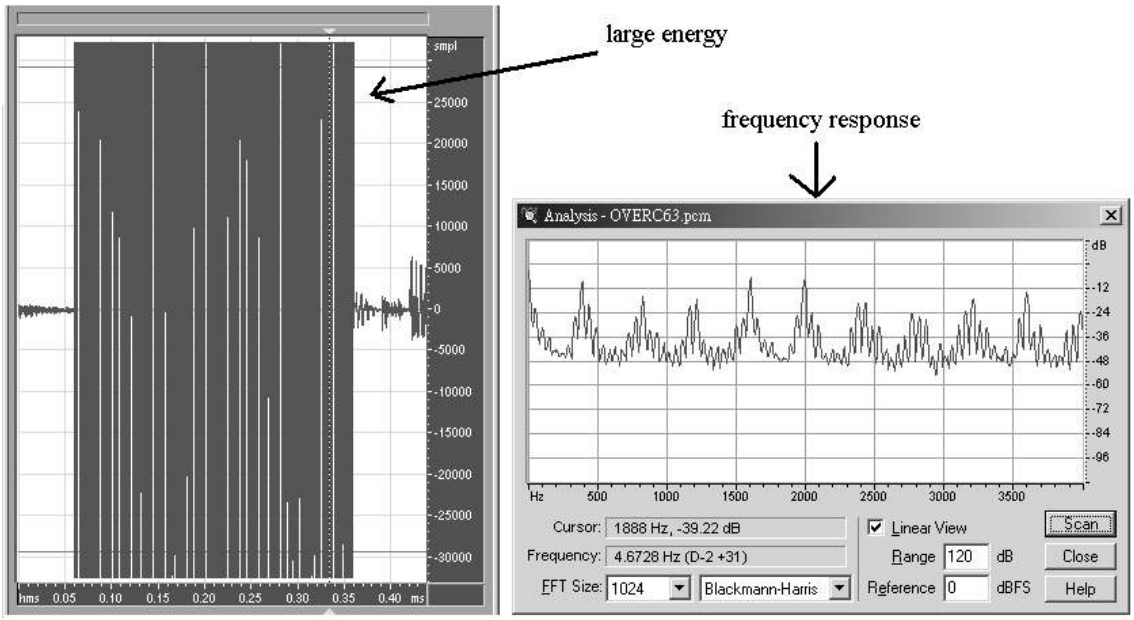

Figure 4.1 Test pattern with large energy and its frequency response.

End of this subsection, we list some statistics of floating-point and fixed-point operation. Table 2 compares the floating-point C code with the fixed-point assembly code. As listed in Table 2, fixed-point assembly code provides very great improvement in speed. Table 3 shows the comparison of memory size. As shown in Table 3, memory size are greatly reduced.

Table 2 Fixed-point improvement of instructions per frame

	Floating-point C code	Fixed-point Assembly code	Improvement
Descaling	235418.14	95415.11	59.47%
AntiAlias	350568.59	7901.33	97.75%
FreqInv	35060.00	2128.00	93.93%
IMDCT	2044628.70	111715.95	94.54%
Synthesis	4706888.00	505220	89.27%

Table 3 Fixed-point improvement of program memory (word)

	Floating-point C code	Fixed-point Assembly code	Improvement
Descaling	827	112	86.46%
AntiAlias	571	39	93.17%
FreqInv	42	25	40.48%
IMDCT	3141	144	95.42%
Synthesis	163	126	22.70%

4.1.2 An Efficient Algorithm Implementation

Although there are many efficient implementations proposed for MDCT/IMDCT and analysis/synthesis filter bank computation, these implementations need to unroll the algorithm and use the memory size to interchange the time [31][32]. This thesis uses another efficient implementation which is a good compromise between memory and time. This method also fits the architecture of C54x DSP.

The IMDCT is defined as Equation (4.14) and has the even anti-symmetric property as Equation (4.15) [33].

$$x(i) = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos(\frac{\pi}{2n}(2i+1+\frac{n}{2})(2k+1)) \quad \text{, for i = 0 ~ n-1} \tag{4.14}$$

$$\begin{cases} x(\frac{n}{2}-i-1) = -x(i) \\ x(n-i-1) = x(\frac{n}{2}+i) \end{cases} \text{, for } \quad 0 \le i < \frac{n}{4} \tag{4.15}$$

This is to say, IMDCT can be realized by calculating x(i) where $0 \le i < \frac{n}{4}$ and $\frac{n}{2} \le i < \frac{3n}{4}$ only. Additionally, we can reverse the sign of the particular window coefficients which are performed after IMDCT. Figure 4.1 illustrates the efficient operation for n=12 (short window).

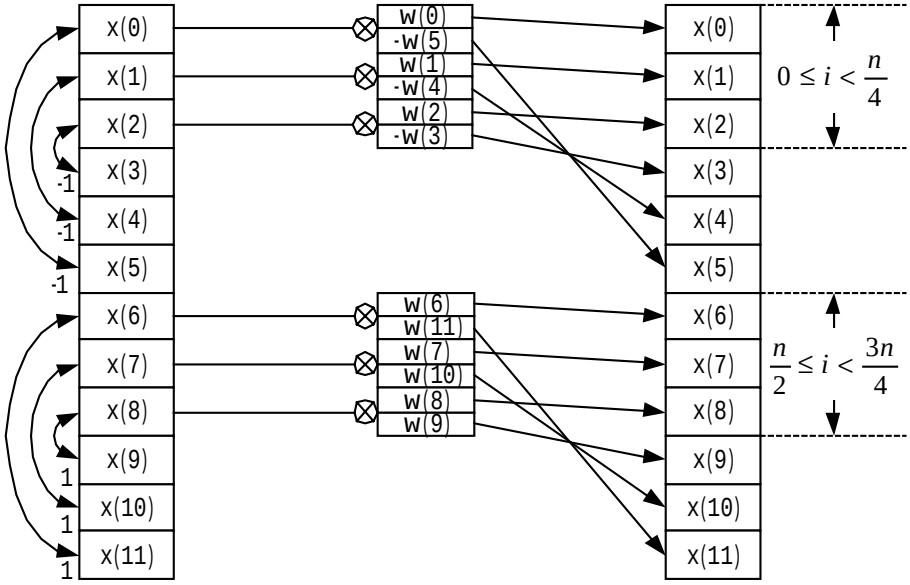

Figure 4.2 Efficient IMDCT and windowing operation [33].

In addition, there are some properties of cosine terms in (4.14) between short and long window as Equation (4.16) shown.

$$\cos_S\left(\frac{\pi}{2m}\left(2j+1+\frac{m}{2}\right)(2k+1)\right) = \cos_L\left(\frac{\pi}{2n}\left(2i+1+\frac{n}{2}\right)(2k+1)\right) \tag{4.16}$$

for j = 0 ~ 11 i = 3j + 1

where:
$\cos_S(x)$ are the coefficients for short window,
$\cos_L(x)$ are the coefficients for long window.

Obviously, the coefficients for short window overlap the coefficients for long window as Figure 4.3 shown. Inserting the coefficients of short window into long window with specified index addressing also reduces the memory sizes

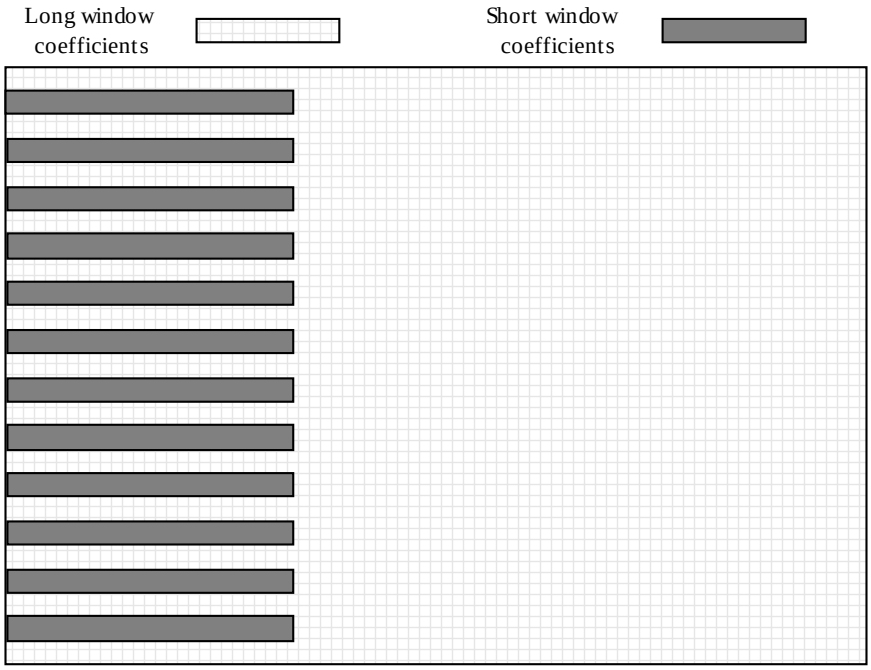

Figure 4.3 The overlap of coefficients for short and long windows.

Besides IMDCT, the synthesis filter bank implementation can also be efficient in two parts. First part is realized in operations that are similar to the efficient IMDCT. Second part is to skip some operations but still generate the correct outputs. In synthesis filter bank, matrix operations as Equation (4.17) shown can be seen as an IMDCT operation.

$$V(i) = \sum_{k=0}^{31} N_{ik} S_k \tag{4.17}$$

where:

$$N_{ik} = \cos((16 + i)(2k + 1)\frac{\pi}{64}) \quad \text{, for i=0~63, k=0~31} \tag{4.18}$$

The coefficients of Nik in Equation (4.17) also have the even anti-symmetric property so V(i) can be expressed by Equation (4.19).

$$\begin{cases} V(\frac{n}{2}-i) = -V(i) & 0 \le i < \frac{n}{4} \\ V(i) = 0 & i = 16 \\ V(n-i-1) = V(\frac{n}{2}+i+1) \text{ , for } & 0 \le i < \frac{n}{4}-1 \\ V(i) = \sum_{k=0}^{31} S_k & i = 48 \end{cases} \tag{4.19}$$

This means that matrix operations can be realized by calculating V(i) where $0 \le i < \frac{n}{4}$ and $\frac{n}{2}+1 \le i \le \frac{3n}{4}$ only. Therefore, this part can save many instruction cycles.

In the second part, we skip the operation of building a 512 values U vector in synthesis filter bank operation in Figure 2.20 and replace Equation (4.20) by Equation (4.21). It implies that we can decrease instuction cycles and memories by skipping building U vector and using another index method.

$$S_i = \sum_{j=0}^{15} U(i+32j)D(i+32j) \text{ , for } 0 \le i \le 31 \tag{4.20}$$

where U is from V vector by Equation (4.22)

$$S_i = \sum_{j=0}^{15} V(32i+j)D(32i+j) \text{ , for } 0 \le i \le 31 \tag{4.21}$$

$$\begin{cases} U(64i+j) = V(128i+j) \\ U(64i+32+j) = V(128i+96+j) \end{cases} \text{ , for } 0 \le i < 8 \text{ , } 0 \le j < 32 \tag{4.22}$$

Table 4 and Table 5 show the improvement of time and memory reduction in IMDCT and synthesis filter bank by using the efficient algorithm.

Table 4 Efficient algorithm improvement of instructions per frame

	Original algorithm	Efficient algorithm	Improvement
IMDCT	111715.95	71803.64	35.73%
Synthesis	505220	356936	29.35%

Table 5 Efficient algorithm improvement of memory (word)

	Original algorithm	Efficient algorithm	Improvement
IMDCT	763	468	38.7%
Synthesis	2652	1143	56.9%

Figure 4.4 and Figure 4.5 show the percentage of execution instructions of IMDCT and synthesis function relative to others without and with the efficient algorithm, respectively. We can see that IMDCT and synthesis functoin reduce many instruction cycles.

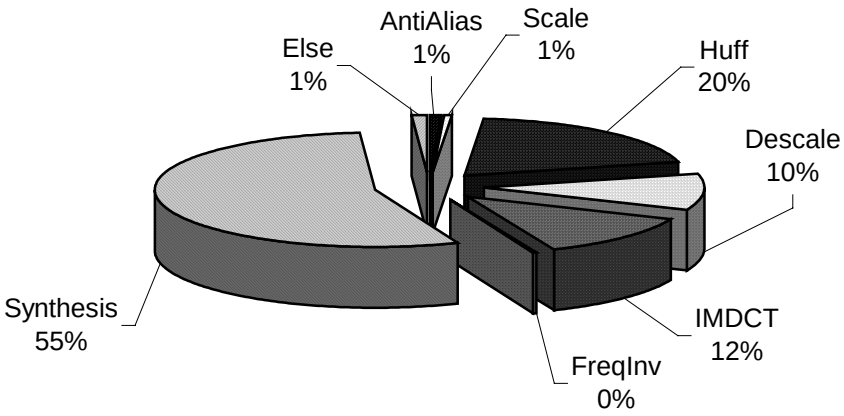

Figure 4.4 Execution instructions without the efficient algorithm.

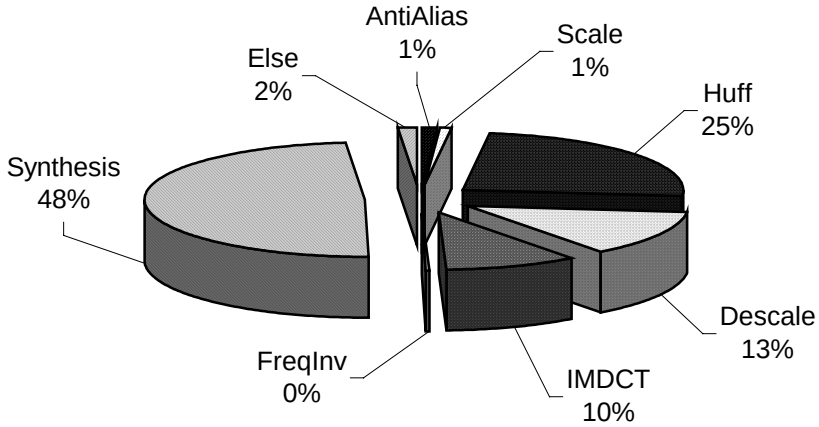

Figure 4.5 Execution instructions with the efficient algorithm.

4.1.2 Multi-Task, Multi-Thread Management

The software required in typical embedded microprocessor systems is comprised of two general components, the application software and the system software. Application software is what programmers usually implement first, e.g. audio coding or speech coding algorithms. Under this layer is system software which is responsible for managing the system resources for application software. System resources include the hardware devices on the target platform and the microprocessor. Take the MPEG/Audio Layer 3 decoder as an example, the decoding algorithm is application software and reading the input stream and playing the output PCM samples are the system software, as Figure 4.5 shown.

In order to operate correctly in real-time, both of application software and system software have their own task cycles and must meet their deadlines respectively. For example, application software has to decode one frame in a limited period while system software has to read-in bitstream according to the bit-rate of bitstream and send-out each audio sample in the original sampling period. To realize the mechanism, we implement the real-time system by multi-task with multiple threads.

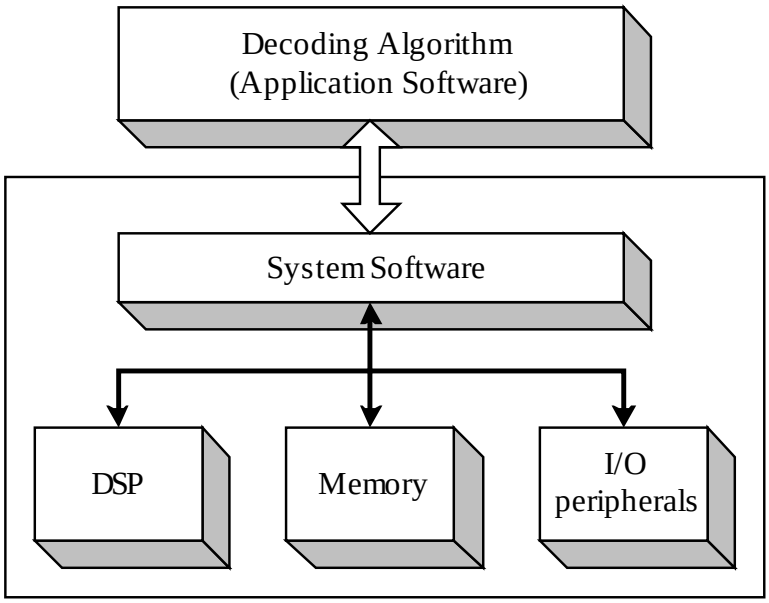

Figure 4.6 Embedded system software components [34].

In simple systems, the system software consists of basic hardware initialization, peripheral access functions and hardware interrupt service routines. Systems that are more complex require real-time scheduling of the DSP to ensure correct operation. Furthermore, as applications require concurrent access to hardware resources such as the DSP, memory, or I/O, the need for an efficient resource manager and scheduler becomes paramount. Managing these resources is precisely the benefit of using DSP/BIOS II. DSP/BIOS II provides system services to manage the DSP system hardware components and to provide applications with services that manage the DSP utilization.

DSP/BIOS II® is a kernel that provides run-time services which developers use to build TI DSP applications and manage application resources. DSP/BIOS II effectively extends the DSP instruction set with real-time, run-time kernel services that form the

underlying architecture, or infrastructure, of real-time DSP applications. DSP/BIOS provides many features for program development [35]:

- A program can dynamically create and delete objects that are used in special situations.
- The threading model provides thread types for a variety of situations. Hardware interrupts, software interrupts, tasks, idle functions and periodic functions are all supported.
- Structures to support communication and synchronization between threads are provided. These include semaphores, mail boxes and resource locks.
- Two I/O modules are supported for maximum flexibility and power. Pipes are used to support simple cases in which one thread writes to the pipe and another reads from the pipe. Streams are used for more complex I/O and to support device drivers.
- The DSP/BIOS plug-ins allow real-time monitoring of program behavior.

In many DSP applications, the data flow from input to output is often a continuous flow of data blocks or buffers. The DSP/BIOS II data pipes (PIP) and data streams (SIO) modules are well suited to manage streaming data. Streaming data applications require management of the flow of data buffers throughout the application. DSP/BIOS II data pipes and data streams are kernel objects optimally designed to perform these tasks. Both module transfer buffers within the pipe or stream by copying pointers rather than by copying data between buffers. In general, the pipe module supports low-level communication, while the stream module supports high-level, device-independent I/O.

Except PIP and SIO modules, DSP/BIOS also provide other modules including HWI, SWI, TSK, MBX and DEV which will be used with PIP or SIO. HWI (hardware interrupt module) is executed after a hardware interrupt triggered in order to perform a critical task that is subject to a deadline. HWIs are the threads with the highest priority in a DSP/BIOS application. SWI (software interrupt module) is triggered by calling SWI functions from the program. Software interrupts provide additional priority levels between hardware interrupts and the background tasks. TSK (multitasking module) dynamically schedules and preempts tasks based on the task's priority level and the task's current execution state. Lower level threads can be suspended during execution until necessary resources are available. MBX (mailbox module) is used for inter-task communication and synchronization. It can pass messages from one task to another. MBX can also be combined into SWI module to synchronize a software interrupt. DEV (device driver module) is software module that manages a class of devices. For more information about these modules, please refer to [34][35].

In the following, we will realize the implementation of PIP and SIO both and compare the difference between PIP and SIO.

◆ PIP Implementation

To transfer data between the ISR and the application, we use PIP module first. One data pipe transfers data from the ISR to the application, the other transfers full data to the ISR for output.

In Figure 4.7, the decode() function attached to the DSP/BIOS II SWI thread (echoSwi) performs the audio processing. The audio processing thread activates only when both a full block of data and an empty block of data are available. To synchronize

these events, we use the echoSwi's mailbox. The initial value of the SWI mailbox is set to 3, which sets the first 2 bits in the SWI mailbox to 1. When both of these bits become 0, the SWI thread activates to perform the process of decoder.

Both data pipes signal the software interrupt using SWI_andn() calls to their assigned bits in the SWI mailbox to synchronize the process. The input data pipe will signal the echoSwi when the ISR has a block and it is available for processing by calling SWI_andn(2) to clear bit 1 in the SWI mailbox. Likewise, the output data pipe will signal the audioSWI when an empty block of data is available for the application to fill by calling SWI_andn(1) to clear bit 0 in the SWI mailbox.

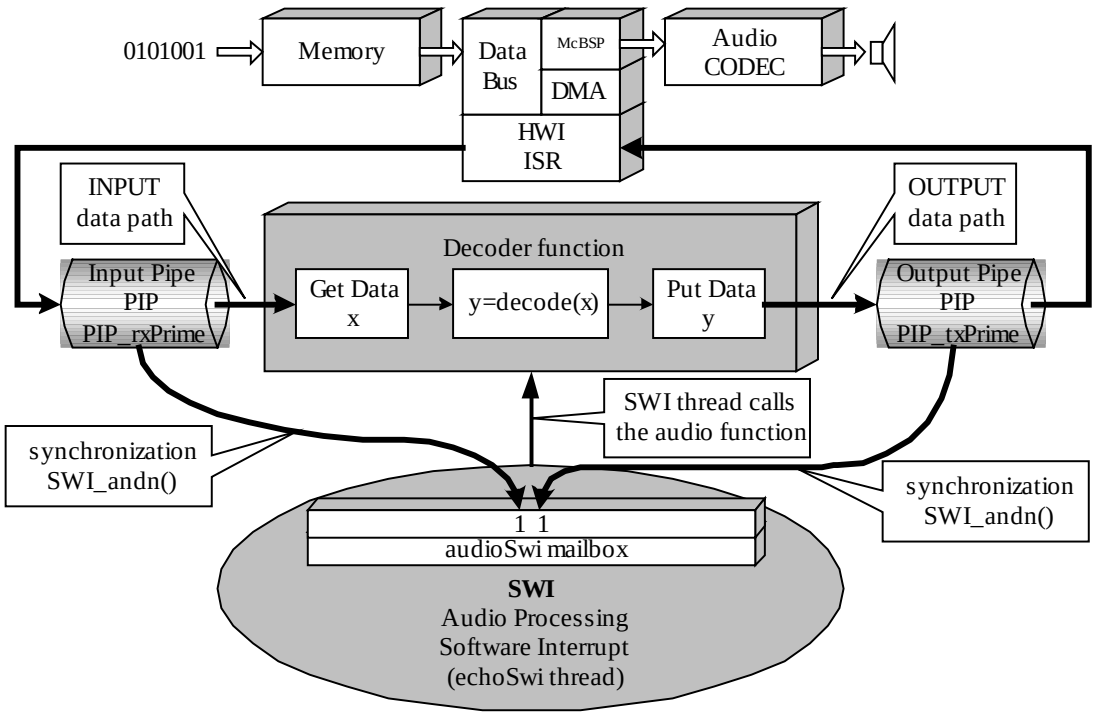

Figure 4.7 Architecture of multi-task using PIP module [34].

In Figure 4.8, execution graph, we can see that echoSwi has higher priority and preempts the other threads. After the echoSwi finishs its thread, other threads restitute to execute. Once echoSwi is ready, preemption occures again.

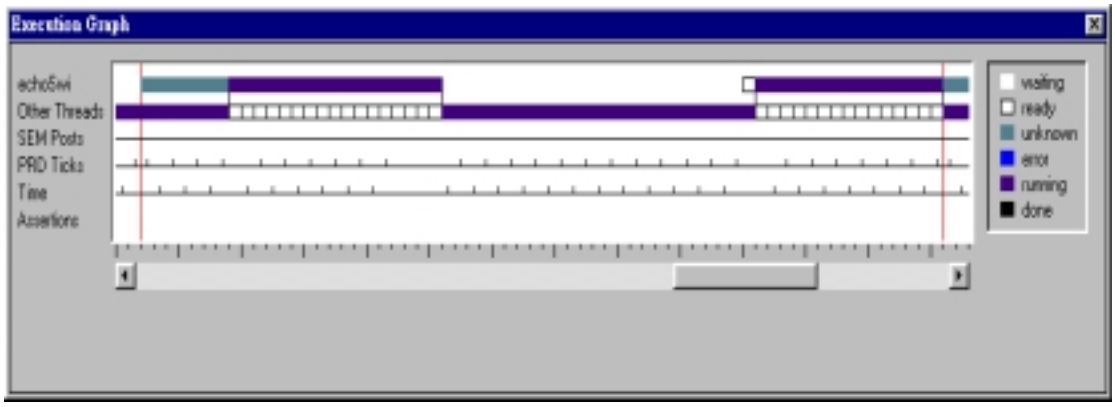

Figure 4.8 Execution graph of PIP implementation.

◆ SIO Implementation

In addition to PIP implementation, we use SIO module which is the other mechanism to transfer data between the ISR and application. The SIO provides a high-level device independent I/O mechanism for use with TSK threads. SIO goes beyond PIP by offering the ability to create new SIO objects at run-time. To provide this ability, SIO has its own device driver model, DEV. A small set of device specific functions, such as open, close, and buffer management, is implemented and accessed by a SIO object through a function table.

In this implementation, application software is in the echoTsk() task, as Figure 4.8 shown. This task creates one input stream and one output stream at run-time. Task will be suspended when streams generate software interrupt since SWI has higher priority

than TSK. Input stream will generate software interrupt when sends data to decoder() while output stream will generate software interrupt when sends out data from decoder() function.

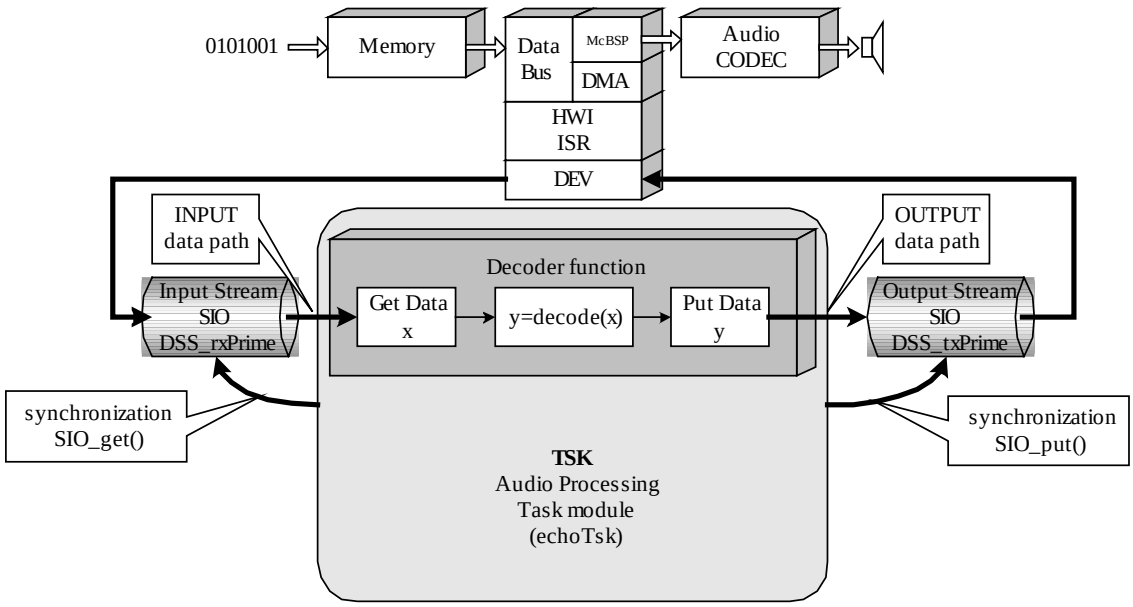

Figure 4.9 Architecture of multi-task using SIO module.

In Figure 4.9, we can see that KNL_swi has higher priority than echoTsk and preempts it when KNL_swi is ready. TSK_idle will run when echoTsk is finished and CPU has more execution power.

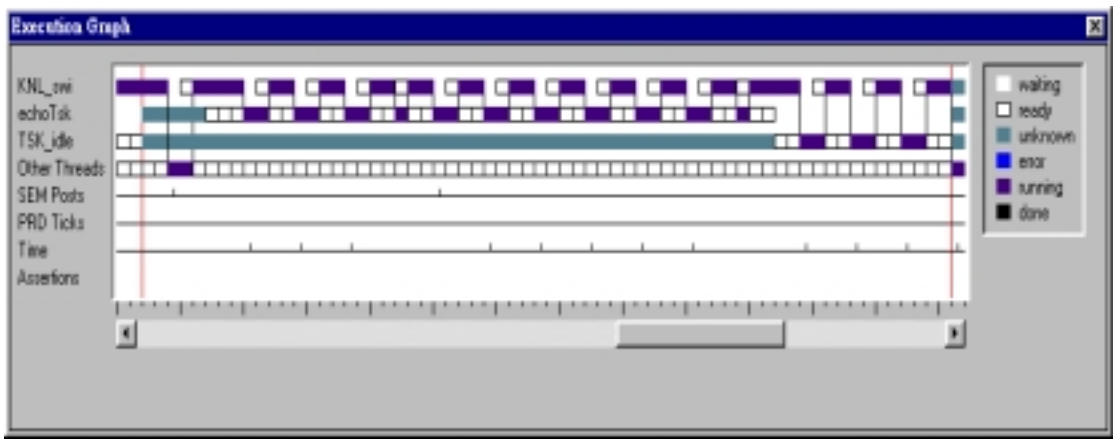

Figure 4.10 Execution graph of SIO implementation.

Consequently, DSP/BIOS II offers two basic constructs for handling data I/O in a real time system: data streams and data pipes. They differ in their approach to solving the problem, but they both provide known, solid, deterministic methods of handling data I/O. Table 6 compares these two modules in more detail. Figure 4.11 and 4.12 shows the CPU load graph of PIP and SIO mechanism respectively. Note that CPU load of PIP has lighter load than SIO's.

Table 6 Difference between PIP and SIO. [35]

Pipes	Streams
Programmer must create own driver structure	Provides a more structured approach to device-driver creation
PIP functions are non-blocking	SIO are blocking functions and will wait until a buffer is available
Uses less memory and is generally faster	More flexible; generally simpler to use and slower.
Pipes must created statically before run –time	Streams may be created either at run-time or statically.
Easy interface with SWI and TSK	Good level of hardware abstraction
Ability to have multiple buffers	Synchronization mechanism with TSK
Designed to be used by only one SWI/ TSK	Highly flexible
Can only work on one frame at a time	Can service multiple TSK
Frame sizes are fixed	Can prototype with a different SIO/ DEV

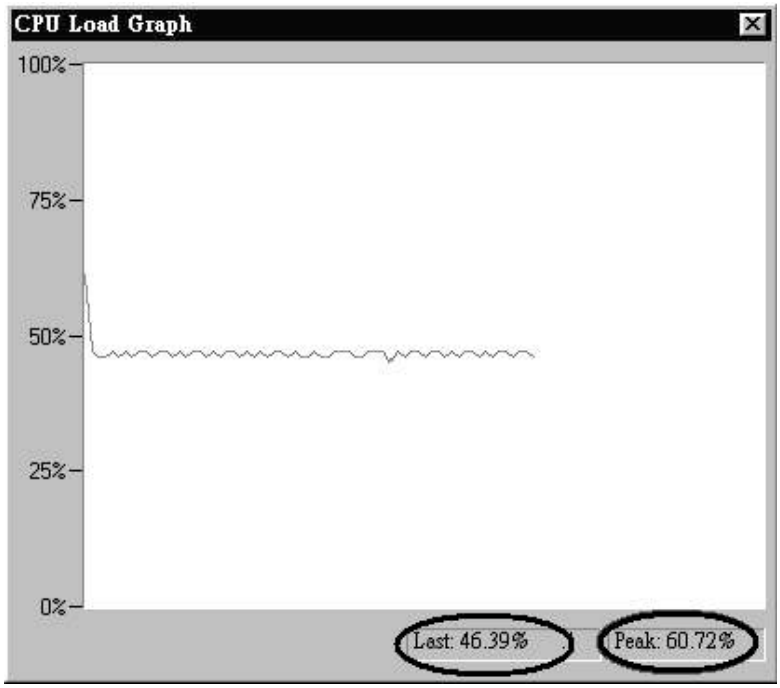

Figure 4.11 CPU load graph of PIP implementation.

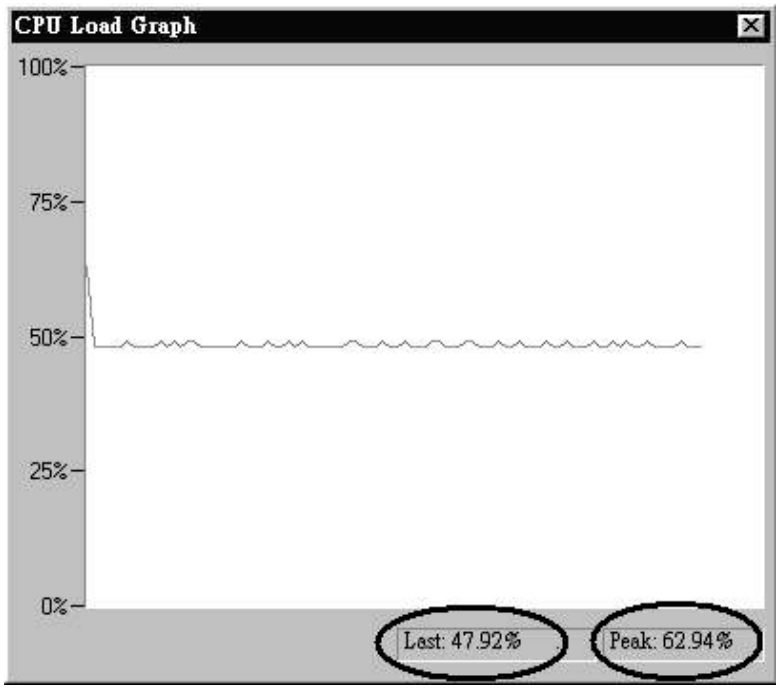

Figure 4.12 CPU load graph of SIO implementation.

4.2 Verification

The performance of this decoder is shown in Table 7. For different sampling rate and bit-rate which means the different compression ratio, the performance varies in small range. The SNR is defined by Equation 4.23 and Error Bit is derived by Equation 4.24.

$$SNR[dB] = 10 \times \log_{10} \frac{\sum PCM_{fixed}[i]^2}{\sum (PCM_{float}[i] - PCM_{fixed}[i])^2} \tag{4.23}$$

$$Error\_bit = \frac{1}{N} \sum \log_2 (\left| PCM_{fixed}[i] - PCM_{float}[i] \right|) \tag{4.24}$$

where we define $\log_2(0)=0$. N = 1152.

The MIPS (Million Instruction Per Second) in Table 7 is the required MIPS to decode a particular bitstream if the DSP is running at 100 MIPS.

Table 7　SNR, Error_bit ,MIPS of various compression ratio.

sampling rate (kHz)	bit rate (kbps)	Compression Ratio	SNR (dB) (v.s floating)	Error Bit	MIPS
32	64	8.0	49.92	3.7	22.83
	128	4.0	48.38	3.8	28.49
	160	3.2	48.62	4.2	30.40
44.1	64	11	48.57	3.2	27.94
	128	5.5	48.96	3.3	34.16
	160	4.4	48.74	3.4	37.02
48	64	12.0	50.67	3.1	29.57
	128	6.0	49.15	3.2	35.80
	160	4.8	49.02	3.4	38.66

In Figure 4.13, the output PCM waveform generated by fixed-point decoder is almost the same with the output of floating-point decoder. The errors of fixed-point and floating-point are very small.

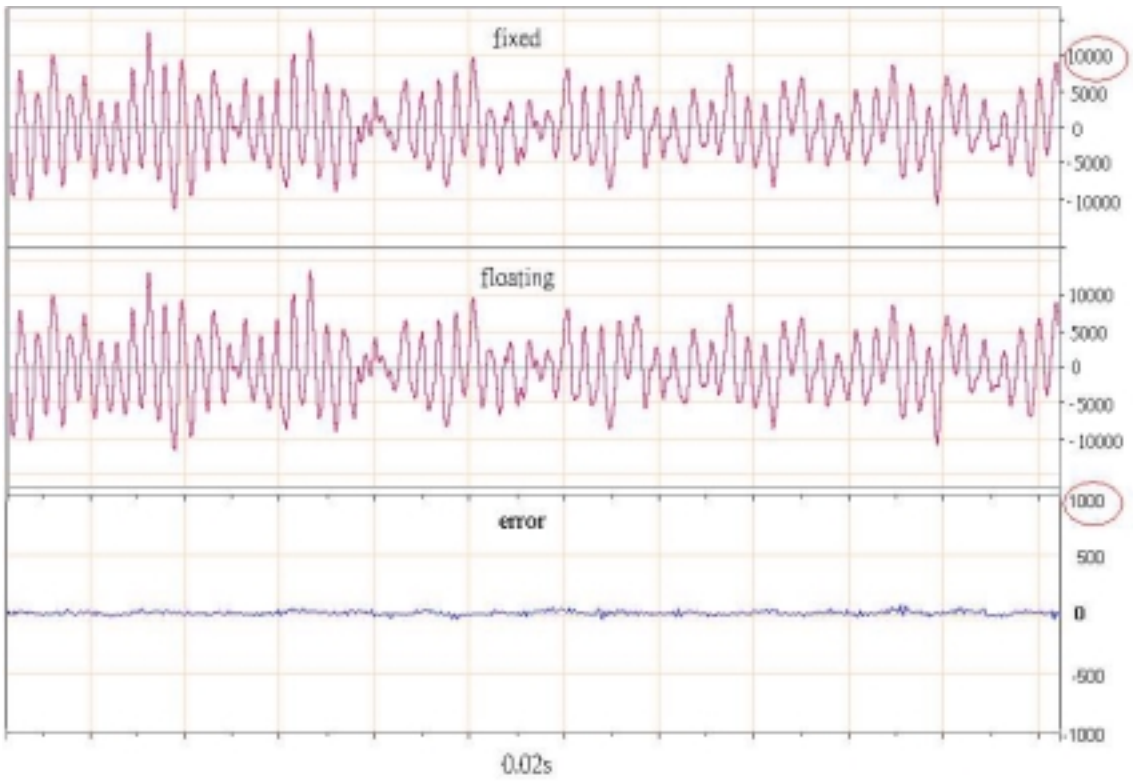

Figure 4.13 Waveform comparison between floating and fixed-point decoder.

Finally, we conclude with the performance of the most popular format in this chapter. Decoding the most popular format – 44.1 kHz sampling rate, 128 Kbps bitrate, compression ratio 5.5:1, gives SNR 48.96 dB, error_bit 3.3 bits and 34.16 MIPS. The MIPS corresponds to 34.16% of the maximum computation capacity of TMS320C5402 DSP. The program memory and data memory used in the decoder are about 7.1 kwords and 17.2 kwords, respectively.

Chapter 5

Conclusion and Future Works

5.1 Conclusion

This thesis describes the MPEG/Audio Layer 3 coding algorithm and presents a real-time decoder implementation on a fixed-point DSP chip. The MPEG/Audio Layer 3 provides high compression ratio of audio signal with good sound quality. The DSP is software programmable and offers IDE (Integrated Development Environment) tools which we can develop and debug the algorithm quickly and easily in the programming stage.

In implementation portion, using the fixed-point operation increases the execution speed and reduces the memory size. Coding with assembly language also achieves these goals. In addition, we adopt several optimum methods and an efficient algorithm to improve performance. Finally, we realize real-time decoder with the PIP and SIO module in multi-task and multi-thread framework. The implemented decoder uses 7.1 kwords of program memory and 17.2 kwords of data memory. It consumes about 34% computation power of C54x DSP chip. This decoder also provides the SNR more than

45dB while comparing to floating-point decoder and sacrifices about 4-bits resolution for fixed-point operation.

5.2 Future Works

With the rapid upgrowth of audio compression knowledge, many other audio compression formats are developed and realized. Dolby AC-3 [36], MPEG-2 Advanced Audio Coding (AAC) [37], Microsoft® Windows Media™ Audio (WMA) [38] are all the good audio coding technology compare with MPEG/Audio Layer 3 and booming in the world of audio coding.

Appling the software programmable characteristic of DSP, we can develop a multi-format decoder by “switching the decoder”. When accessing particular format, switching to corresponding decoder makes us decode the file without adding another hardware like the ASIC do.

References

[1] Audio Engineering Handbook, K. Blair Benson. Benson, McGraw-Hill Book Company, 1988 ISBN 0-07-004777-4

[2] E.F. Schroder and W. Voessing, "High Quality Digital Audio Encoding with 3.0 Bits/Sample using Adaptive Transform Coding," in *Proc. 80th Conv. Aud. Eng. Soc.*, preprint #2321, Mar. 1986.

[3] K. Brandenburg, "OCF- A New Coding Algorithm for High Quality Sound Signals," in *Proc.ICASSP-87,* pp.5.1.1-5.1.4, May 1987.

[4] J. Johnston, "Transform Coding of Audio Signals Using Perceptual Noise Criteria," *IEEE J. Sel. Areas in Comm*., pp. 314-323, Feb. 1988.

[5] Y. Mahieux, et al., "Transform Coding of Audio Signals Using Correlation Between Successive Transform Blocks," in *Proc. Int. Conf. Acous., Speech, and Sig. Process.* (ICASSP-89), pp. 2021-2024, May 1989.

[6] K. Brandenburg, et al., "ASPEC: Adaptive Spectral Entropy Coding of High Quality Music Signals," in Proc. 90th *Conv. Aud. Eng. Soc*., preprint #3011, Feb. 1991.

[7] Y.F. Dehery, et al., "A MUSICAM Source Codec for Digital Audio Broadcasting and Storage," in Proc. ICASSP-91, pp.3605-3608, May 1991.

[8] G. Theile, et al., "Low-Bit Rate Coding of High Quality Audio Signals," in *Proc. 82nd Conv. Aud. Eng. Soc*., preprint #2432, Mar. 1987.

[9] Meshkat, S., and I. Ahmed, "Using DSPs in AC Induction Motor Drives," *Control Engineering*, Feb. 1988.

[10] Reimer, J.B., and G.A. Frantz, "Customization of a DSP Integrated Circuit for a Customer Product," *Transactions on Consumer Electronics,* USA, Aug. 1988.

[11] Papamichalis, P., and J. Reimer, "Implementation of the Data Encryption Standard Using the TMS32010," *Digital Signal Processing Applications*, 1986.

[12] Reimer, J., and A. Lovrich, "Graphics with the TMS32020," *WESCON/85 Conference Record*, USA, 1985.

[13] Casale, S., R. Russo, and G. Bellina, “Optimal Architectural Solution Using DSP Processors for the Implementation of and ADPCM Transcoder,” *Proceedings of GLOBECOM* ’89, pages 1267-1273, Nov. 1989.

[14] Reimer, J., G. Benbassat, and W. Bonneau Jr., “Application Processors: Making PC Multimedia Happen”, Silicon Valley PC Design Conference, Jul. 1991.

[15] Charles D. Murphy and K. Anandakumar, “Real-Time MPEG-1 Audio Coding and Decoding on a DSP Chip”, IEEE, *Trans. on Consumer Electronics*, Vol.43, No.1, Feb. 1997.

[16] ISO/IEC JTC1/SC29/WG11 MPEG, IS11172-3 “Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5Mbit/s, Part 3: Audio” 1992.

[17] K. Brandenburg and H. Popp, “An introduction to MPEG Layer-3”, *Fraunhofer Institut fur Integrierte Schaltungen(IIS)*, EBU TECHNICAL REVIEW, Jun. 2000.

[18] P.P. Vaidyanathan. “Quadrature mirror filter banks, m-band extensions and perfect-reconstruction techniques.” *IEEE ASSP Maganize*, Jul. 1987.

[19] Davis Pan, “A Tutorial on MPEG/Audio Compression”, *IEEE Multimedia*, Vol. 2, No. 2, Summer 1995.

[20] J. H. Rothweiler, “Polyphase Quadrature Filters – a New Subband Coding Technique,” Proc of the Int. Conf. IEEE ASSP, 27.2, pp1280-1283, Boston 1983.

[21] J. Princen and A. Bradley, “Analysis/Synthesis Filterbank Design Based on Time Domain Aliasing Cancellation,” *IEEE Trans. On Acoust. Speech, and Signal Process.* Vol. ASSP-34, pp.1153-1161, 1986.

[22] Terhardt, E., “Calculating Virtual Pitch,” Hearing Research, pp. 155-182, 1, 1979.

[23] E. Ambikairajah, A. G. Davis and W. T. K. Wong, “Auditory masking and MPEG-1 audio compression”, Electronics and communication engineering journal, Aug. 1997.

[24] K Salomonsen, S Søgaard, E P Larsen, “Design and Implementation of an MPEG/Audio Layer III Bitstream Processor” 1997.

[25] TMS320C5402 DSK Help (SPRH075A) Texas Instruments.

[26] TMS320C54x DSP Reference Set, *Volume 1: CPU and Peripherals* (SPRU131) Texas Instruments, 1999.

[27] Code Composer Studio User’s Guide (SPRU328) Texas Instruments, 2000.

[28] TMS320C54x Code Composer Studio Tutorial (SPRU327A) Texas Instruments, 1999.

[29] Jason Jiang, “General Guide to Implement Logarithmic and Exponential Operations on a Fixed-Point DSP”, *TI Application Report*, Dec. 1999.

[30] 胡竹生, 賴鴻志, 張勝凱, ”TMS320C54xx DSP 晶片原理與應用”, 全華, 2000.

[31] B. G. Lee, “ A new Algorithm to Compute the Discrete Cosine Transform.” IEEE Trans. *Acous., Speech, and Signal. Processing.*, vol.ASSP-32, pp. 1243-1245, Dec. 1984.

[32] Vladimir B. and K. R. Rao, “ An Efficient Implementation of the Forward and Inverse MDCT in MPEG Audio Coding”, *IEEE Signal processing letters*, vol. 8, No. 2, Feb. 2001.

[33] Tadashi Sakamot, et al., “A Fast MPEG-AUDIO Layer III Algorithm for A 32-Bit MCU .” in IEEE Trans. *Consumer Electronics.*, vol. 45, No.3, Aug. 1999.

[34] Andy The’ and David W. Dart, “How to Get Started With DSP/BIOS II.” *Application Report* (SPRA697), Texas Instruments. Oct. 2000.

[35] TMS320C54x DSP/BIOS User’s Guide, Texas Instruments, 2000.

[36] C. Todd, et. Al., “AC-3: Flexible Perceptual Coding for Audio Transmission and Storage,” in *Proc. 96th Conv. Aud. Eng. Soc.*, preprint #3796, Feb. 1994.

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

[38] http://msdn.microsoft.com/workshop/imedia/windowsmedia/Tools/MSAudio.asp

[39] TMS320C54x Assembly Language Tools User’s Guide,(SPRU102) Texas Instruments, 1999.

[40] Noll, P. “MPEG digital audio coding”,IEEE Signal Processing Magazine , Volume: 14 Issue: 5 , Page(s): 59 –81, Sep. 1997

[41] Shlien, S. “Guide to MPEG-1 audio standard” Broadcasting, IEEE Transactions on , Volume: 40 Issue: 4 , Page(s): 206 –218, Dec. 1994

[42] T. Sakamoto, M. Taruki, and T. Hase, “A FAST MPEG-AUDIO LAYER III ALGORITHM FOR A 32-BIT MCU”, IEEE *Trans. on Consumer Electronics,* Vol.45, No.3, Aug. 1999.

[43] Davis Pan, “A Tutorial on MPEG/Audio Compression”, *IEEE Multimedia,* Vol. 2, No. 2, Summer 1995.