國 立 交 通 大 學

電機與控制工程學系

碩 士 論 文

在 MPEG/Audio 上資料隱藏方法之研究與使用 ADSP-2181 數位訊號處理器實現資料隱藏解碼器

A Study of Data Embedded Method on MPEG/Audio and Implementation of Data Embedded Decoder on the ADSP-2181 DSP Processor

研 究 生： 黃榮煌

指導教授： 鄧清政 教授

共同指導教授： 吳炳飛 教授

中 華 民 國 九 十 三 年 七 月

在 MPEG/Audio 上資料隱藏方法之研究與使用
ADSP-2181 數位訊號處理器實現資料隱藏解碼器

研　　究　　生	：黃榮煌	Student	: Ruang-Huang Huang
指　導　教　授	：鄧清政 教授	Advisor	: Prof. Ching-Cheng Teng
共同指導教授	：吳炳飛 教授	Co-Advisor	: Prof. Bing-Fei Wu

國 立 交 通 大 學
電機與控制工程學系

碩 士 論 文

A Thesis

Submitted to Department of Electrical and Control Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of Master

in

Electrical and Control Engineering

July 2004

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 三 年 七 月

在 MPEG/Audio 上資料隱藏方法之研究與使用 ADSP-2181 數位訊號處理器實現資料隱藏解碼器

學生：黃榮煌　　　　　　　　指 導 教 授 ：　鄧清政 教授
　　　　　　　　　　　　　　共同指導教授：　吳炳飛 教授

國立交通大學電機與控制工程學系碩士班

摘　要

本論文提出把資料嵌入 MPEG/Audio 裡的方法與實作。資料嵌入 MP3 的目的是為了增加 MP3 的用途與功能性，也可以提供音樂公司對消費者的額外服務，如在 MP3 中加入歌詞同步顯示、歌手照片或是消費者的基本資訊等等。本論文中的資料嵌入方法可分為資料嵌入編碼器和解碼器：資料嵌入編碼器是在電腦上實現，主要是把資料嵌入到 MP3 音框的三個比較不重要的區塊裡，對音樂本身的音質較無影響。而這三個區塊分別是 big-value 區塊中 10KHz 以上的符號位元、count1 區塊中的符號位元和 bit reservoir 中用不到的位元。資料嵌入解碼器是在 ADSP-2181 數位訊號處理器上實現，主要是做資料的擷取並做分析，並做歌詞和圖片的同步顯示。此包含資料嵌入解碼器的 MP3 解碼器在 ADSP-2181 上使用了 20.7Kbytes 的程式記憶體和 23.6Kbytes 的資料記憶體，並且在 18MIPS 就可達到即時播放的速度，約佔此晶片 55%的運算能力。此資料嵌入的方法亦可應用到 MPEG-2/Audio AAC 或是 MEPG-4/Audio AAC 裡，並不只侷限於使用在 MP3 演算法上。

A Study of Data Embedded Method on MPEG/Audio and Implementation of Data Embedded Decoder on the ADSP-2181 DSP Processor

Student：Ruang-Huang Huang　　Advisor　：Prof. Ching-Cheng Teng

Co-Advisor　：Prof. Bing-Fei Wu

Department of Electrical and Control Engineering

National Chiao Tung University

Abstract

In this thesis the method and implementation of embedding data into MPEG/Audio will be discussed. The purpose of embedding data into MP3 is to add applications and functions of MP3, as well as to facilitate the music company to provide additional service to customers, for example, adding pictures of singers, background information or lyrics into MP3 for synchronous display. Data embedded encoder and decoder will be introduced in this thesis. The data embedded encoder, implemented on PC, is to embed data into three less important music regions in MP3, causing less influence to the quality of music. These three music regions are: sign bites which exceed 10 KHz in big-value region, sign bites in count1 region, and unused bites in bit reservoir region. Data embedded decoder, ported on ADSP-2181 DSP processor, is to catch data for analyzing and executing synchronous display of lyrics and pictures. The MP3 data decoder in ADSP-2181 uses 20.7Kbytes of program memories and 23.6Kbytes of data memories, and will accomplish real-time playing in 18MIPS, occupying about 55% processing power of the DSP. This approach can be applied in MPEG.-2/Audio ACC or MPEG-4/Audio ACC as well, not only in MP3 algorithm.

誌　謝

首先感謝鄭清政教授和吳炳飛教授的指導，從他們身上我學到了做研究的嚴謹態度，和解決問題的能力。老師也提供很豐富的研究資源，有良好的環境和最新的設備。也就是有條件才能完成這本論文。

在實驗室中也有很多要感謝的伙伴。一同參加比賽的煜翔同學、俊傑同學、映伶學妹，雖然沒得獎，但是重點是比賽時一起努力的過程，這是難以忘記的回憶。志旭學長和煜翔同學在我做研究遇到困難時也時常給我許多建議，告訴我許多經驗，讓我研究可以順利的完成。研究做累時常和我聊天的重甫學長、馬哥、坤卿學長，假日常來實驗室找我吃飯的明達學長、紹麒學長，晚上常和我運動的光輝學長，幫我搬家無數次的琪文同學，國防役要和我一起到凌陽上班的律嘉同學，還有其它實驗室的伙伴。還要感謝其它我在交大認識幫過我的許多朋友們。

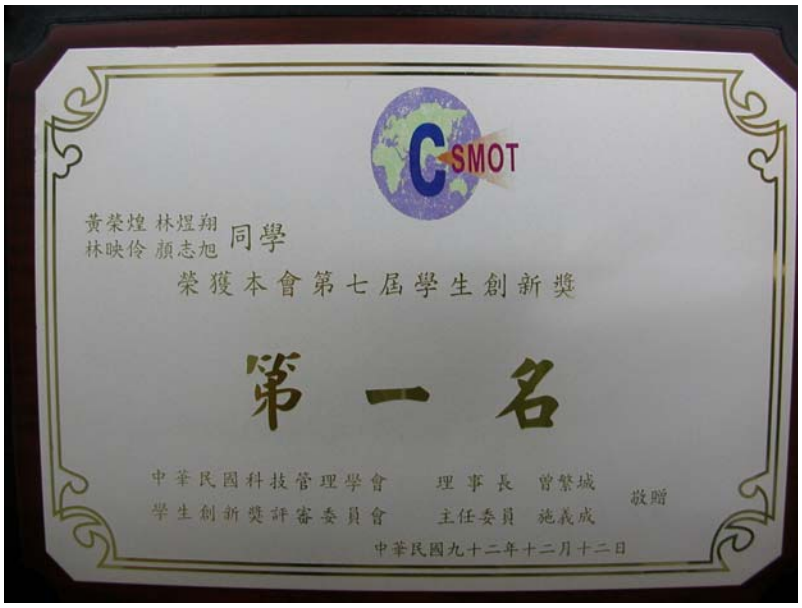
最重要的要感謝我的父母炳和先生、玉蒼女士，有你們辛苦的工作拉拔，我才有今天，弟妹榮仁、秀穗、還有家族裡的其它家人，陪我從小一路走來，我也希望你們陪我分享論文完成的喜悅。

最後還要感謝很多在求學過程中陪我走過許多歡笑與不如意的朋友，現在大家各分東西為自己的前程努力，也希望大家一起加油。

Award

第七屆「生創新獎」 第一名

得獎作品	向下相容的 mp3 音樂安全機制
得　獎　者	黃榮煌、林煜翔、林映伶、顏志旭 交通大學/電機與控制工程學系
指導老師	吳炳飛



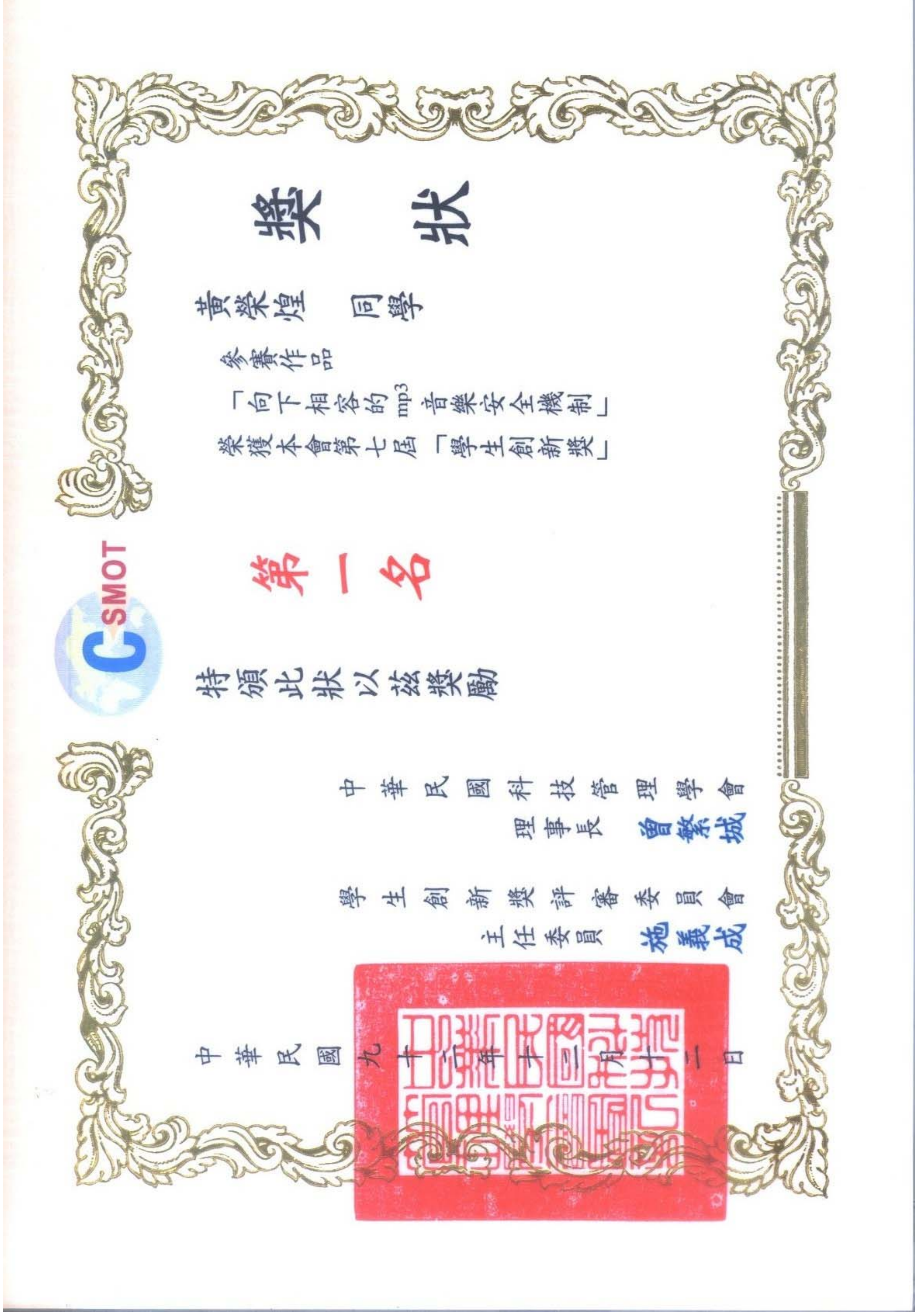
黃榮煌 林煜翔
林映伶 顏志旭 同學

榮獲本會第七屆學生創新獎

第 一 名

中華民國科技管理學會　理事長　曾繁城
學生創新獎評審委員會　主任委員　施義成
敬贈

中華民國九十二年十二月十二日



獎狀

黃榮煌 同學

參賽作品

「向下相容的mp3音樂安全機制」

榮獲本會第七屆「學生創新獎」

第一名

特頒此狀以茲獎勵

中華民國科技管理學會

理事長 曾繁城

學生創新獎評審委員會

主任委員 施義成

中華民國九十二年十月二十二日

Contents

摘　要 .... **I**

ABSTRACT .... **II**

誌　謝 .... **III**

AWARD .... **IV**

CONTENTS .... **VI**

LIST OF TABLES .... **X**

LIST OF FIGURES .... **XI**

CHAPTER 1 INTRODUCTION **1**

1.1 RESEARCH BACKGROUND 1

1.2 RESEARCH MOTIVATION 1

1.3 INNOVATION 2

1.3.1 MP3 Encoder 2

1.3.2 MP3 Decoder 2

1.4 CHARACTERISTIC 2

1.5 CONTENT ORGANIZATION 4

CHAPTER 2 DATA EMBEDDED ALGORITHM IN THE MPEG/AUDIO **5**

2.1 INTRODUCTION TO MPEG AUDIO 5

2.1.1 Introduction of the MP3 Encoder Algorithm 7

2.1.2 Introduction of the MP3 Decoder Algorithm 8

2.2 INTRODUCTION OF DATA EMBEDDED METHODS 9

2.2.1 The Principle of Watermarking Algorithm 9

2.2.2 Main Characteristics for Watermarking Algorithm 10

2.2.3 Applications of Watermarking Algorithm 11

2.2.4 Classification on Watermarking Techniques 12

2.3 DATA EMBEDDED CODEC ALGORITHM FOR MPEG/AUDIO 12

2.3.1 The Properties of Data Embedded Codec 13

2.3.2 The Structure of Data Embedded Codec 16

2.3.3 The Methods of Data Embedded Codec 18

2.3.3.1 Embedded Data into the Count1 Region 18

2.3.3.2 Embedded Data into the Big-Value Region 20

2.3.3.3 Embedded Data into Bit Reservoir 24

CHAPTER 3 ENVIRONMENT OF HARDWARE AND SOFTWARE 27

3.1 HARDWARE ENVIRONMENT 27

3.1.1 ADSP-2181 EZ-KIT Lite Board 27

3.1.2 ADSP-2181 Microprocessor 28

3.2 SOFTWARE ENVIRONMENT 31

CHAPTER 4 IMPLEMENTATION OF DATA EMBEDDED CODEC 35

4.1 DATA EMBEDDED ENCODER 35

4.1.1 Package Embedded Data 35

4.1.2 Embed Data into the MP3 File 38

4.2 DATA EMBEDDED DECODER 40

4.2.1 Extract Data from the MP3 File 41

4.2.2 Porting MP3 Decoder with Data Embedded Decoder on the ADSP-2181 41

4.2.3 Data Stream Analyzer 43

4.2.4 Lyric Analyzer 46

4.3 EXPERIMENTAL PROCESS 48

4.3.1 Encoding 48

4.3.2 Decoding 49

4.4 EXPERIMENTAL RESULTS 54

4.4.1 The Embedded Bits Counts of the Different Methods 54

4.4.2 The Encoding Speed of the Different Methods 56

4.4.3 The Music Quality of the Different Methods 58

4.4.4 The File Size of the Different Methods 60

4.4.5 Comparison with other methods 61

CHAPTER 5 CONCLUSIONS AND FUTURE WORKS 63

5.1 CONCLUSIONS 63

5.2 FUTURE WORKS 64

REFERENCES 65

APPENDIX A MP3 ENCODER/DECODER ALGORITHM 67

A.1 THE STRUCTURE OF MP3 ENCODER ALGORITHM 67

A.1.1 Analysis Polyphase Filter Bank 68

A.1.2 MDCT and Alias Reduction 71

A.1.3 Psychoacoustic Model 73

A.1.4 Nonuniform Quantization 76

A.1.5 Huffman Encoding 77

A.1.6 Bitstream Formatting 78

A.2 THE STRUCTURE OF MP3 DECODER ALGORITHM 78

A.2.1 Decoding of Bitstream 79

A.2.2 Inverse Quantization 81

A.2.3 Frequency to Time Mapping 81

APPENDIX B INTRODUCTION TO THE TESTING STANDARD OF THE MUSIC QUALITY : ODG 84

B.1 INTRODUCTION 84

B.2 DESCRIPTION OF PEAQ 84

List of Tables

TABLE 1 CLASSIFICATION OF WATERMARKING ACCORDING TO SEVERAL VIEWPOINTS [10] 12
TABLE 2 CLASSIFICATION OF THE WATERMARKING TECHNIQUE IN THIS THESIS........ 14
TABLE 3 THE PARAMETERS OF THE HEADER IN THE PACKAGE FILE 36
TABLE 4 THE SPEC. OF THE MP3 DECODER WITH DATA EMBEDDED DECODER ON THE ADSP-2181........ 43
TABLE 5 THE INFORMATION OF “01-CAN” 49
TABLE 6 THE EMBEDDED BITS COUNT OF THE DIFFERENT METHODS 55
TABLE 7 THE EMBEDDED BITS COUNT PER FRAME BY THE DIFFERENT METHODS........ 56
TABLE 8 THE SPEC. OF THE TEST PLATFORM 56
TABLE 9 THE ENCODING SPEED OF THE DIFFERENT METHODS BY FLOATING POINT ENCODER 57
TABLE 10 THE ENCODING SPEED OF THE DIFFERENT METHODS BY FIXED POINT ENCODER 58
TABLE 11 THE TESTING ENVIRONMENT OF THE MUSIC QUALITY 59
TABLE 12 THE MUSIC QUALITY OF THE DIFFERENT METHODS 60
TABLE 13 THE FILE SIZE OF THE DIFFERENT METHODS........ 61

List of Figures

FIG. 1 THE HIERARCHY OF THE ISO MPEG STANDARD.......... 6

FIG. 2 THE COMPARISON OF THE ISO MPEG AUDIO STANDARD COMPRESSION RATIO.......... 7

FIG. 3 MPEG-1/AUDIO LAYER 3 ENCODER BLOCK DIAGRAM [5].......... 8

FIG. 4 MPEG-1/AUDIO LAYER III DECODER BLOCK DIAGRAM.......... 9

FIG. 5 COMBINATION OF THE WATERMARKING PROCESS ON MPEG/AUDIO 10

FIG. 6 THE STRUCTURE OF DATA EMBEDDED ENCODER.......... 17

FIG. 7 THE STRUCTURE OF DATA EMBEDDED DECODER.......... 18

FIG. 8 THE ABSOLUTE THRESHOLD OF HEARING 19

FIG. 9 THE FREQUENCY LINE MAPPING TO FREQUENCY.......... 22

FIG. 10 THE DISTRIBUTION OF MUSIC QUALITY AND BITS IN THE BIG-VALUE REGION 23

FIG. 11 THE DISTRIBUTION OF MUSIC QUALITY AND BITS IN THE BIG-VALUE REGION 24

FIG. 12 THE BITSTREAM AND BIT RESERVOIR OF MP3 25

FIG. 13 THE RESERVOIR OVER 512 BYTES AND STUFF “1” 25

FIG. 14 EZ-KIT LITE’S FUNCTIONAL BLOCK DIAGRAM [15].......... 28

FIG. 15 ADSP-2181 FUNCTIONAL BLOCK DIAGRAM [17] 29

FIG. 16 HARVARD ARCHITECTURE.......... 30

FIG. 17 VISUALDSP USER INTERFACE 32

FIG. 18 THE FORMAT OF THE PACKAGED FILE.......... 37

FIG. 19 THE FLOWCHART OF THE PACKAGED FILE 38

FIG. 20 THE BLOCK DIAGRAM OF THE MP3 ENCODER WITH DATA EMBEDDED ENCODER 39

FIG. 21 THE DISTRIBUTION OF THE UNNECESSARY BITS OF THE BIT RESERVOIR IN MP3 SONG [19].......... 40

FIG. 22 THE BLOCK DIAGRAM OF THE MP3 DECODER WITH DATA EMBEDDED DECODER 41

FIG. 23 THE FLOWCHART OF THE DATA EMBEDDED ANALYZER 45

FIG. 24 THE FLOWCHART OF THE LYRIC ANALYZER.......... 47

FIG. 25 THE EXTRACTED LYRICS OF THE MP3 SONG.......... 50

FIG. 26 PHOTO1.JPG 51

FIG. 27 PHOTO2.JPG 51

FIG. 28 PHOTO3.JPG 52

FIG. 29 PHOTO4.JPG 52

FIG. 30 PHOTO5.JPG 53

FIG. 31 PHOTO6.JPG 53

FIG. 32 MPEG-1/AUDIO LAYER 3 ENCODER BLOCK DIAGRAM [5] 68

FIG. 33 ANALYSIS SUBBAND FILTER AND MDCT 68

FIG. 34 COEFFICIENT OF C[N] AND H[N] (N = 0~511) 70

FIG. 35 FREQUENCY RESPONSE OF SUBBAND 71

FIG. 36 ILLUSTRATION OF THE FOUR APPLICABLE WINDOW TYPES AND USING CONDITION 72

FIG. 37 ILLUSTRATION OF ALIAS REDUCTION BUTTERFLIES 73

FIG. 38 THE ABSOLUTE THRESHOLD OF HEARING 74

FIG. 39 FREQUENCY MASKING THRESHOLD AND THRESHOLD IN QUIET [8] 75

FIG. 40 TEMPORAL MASKING THRESHOLD [8] 76

FIG. 41 MAIN DATA ORGANIZATION OF A FRAME 78

FIG. 42 MPEG-1/AUDIO LAYER III DECODER BLOCK DIAGRAM 79

FIG. 43 DECODING OF BITSTREAM BLOCK DIAGRAM 79

FIG. 44 MPEG-1/AUDIO LAYER III HEADER FORMAT 80

FIG. 45 FREQUENCY TO TIME MAPPING 82

FIG. 46 BLOCK DIAGRAM OF MEASUREMENT SCHEME 85

CHAPTER 1

Introduction

1.1 Research Background

In recent ten years, the popularization of Internet and the rapid development of computer industries make our life more convenient and comfortable. People can communicate by sending messages to each other via electronic(E-) texts, E-mail, E-news, digital image, audio, video etc. under the connection of Internet. Meanwhile, digital music also replaces the traditional music which can be diffused quickly on the net, causing multi-media industry a great loss. The popularity of MP3 has a great impact on the music industry indeed. Some network companies which provide P2P service share the market of the traditional music company using the network connectivity convenience, such as Kuro, ezpeer, etc.

1.2 Research Motivation

The technology of MPEG Audio [2] provides low bit rate and low computation requirement for high quality audio compression. Therefore, it is widely used for storing nearly all kinds of music.

The invention of digital music has changed the consumption model of the traditional music market. Selling music on the net becomes a new model, and also a main trend. Competitors for music content providers increase since they can sell music on the internet, too. The music content providers must offer additional services to compete with the competitors and attract more customers. Data embedded technique can be one of the weapons in the market. Nevertheless, techniques of data embedded which differs from the Cryptography System [1] are to embed extra information into multimedia work, such as MP3 (MPEG -1 Audio Layer III).

1.3 Innovation

1.3.1 MP3 Encoder

In this thesis, the data embedded methods are developed to embed data into MP3 file without depending on the absolute threshold of hearing of the psychoacoustic model, because the psychoacoustic model of the MP3 algorithm is removed in our MP3 encoder.

General watermarking techniques or data hiding techniques reference the psychoacoustic model [14] in the music compressing technique. But the computation of the psychoacoustic model is a great quantity of ratio in the MP3 encoder, and it accounts about 20% computation of the MP3 encoder. Our MP3 encoder not only can embed data without making influence to the music quality but improves the encoding speed.

1.3.2 MP3 Decoder

In general, the MP3 decoder playing the MP3 file with lyric must have an additional lyric file and install a plug-in software to play the lyric. Some users even don’t know where to find the lyric file. The data embedded method will solve the problem. The lyric of a MP3 song can be embedded into the MP3 and the MP3 decoder with data embedded decoder can play the song and show the lyric concurrently.

1.4 Characteristic

In this thesis, we try to find ways to embed digital data into MP3 and combine the embedded data codec with an MP3 codec. This technique for data embedding can include a great amount of information, such as lyrics, pictures of the singer or other information. There are several characteristics:

- **Won't changing the file size, and won't be noticed by users**

  Data embedding techniques won't cause negative influences on quality of audio work and will not be noticed by users or attackers.

- **Using for network streaming broadcast**

  The data is embedded everywhere not in the beginning of the music file. If the music receives from the half of the music file on the internet, the music player also can extract the embedded data. So the music file with embedded data can be used on the network streaming broadcast.

- **Providing additional service by music content provider**

  The MP3 encoder with data embedded encoder can be used for the music content provider to embed some information about singer in the MP3 songs. The content provider can provide this new service in their music product. In other way, content provider will have more advantages to fight the pirates.

- **Portable MP3 decoder with data embedded decoder**

  The MP3 decoder with data embedded decoder has been ported on the ADSP-2181 processor. It can combine with the USB storage device and LCD display to become a portable device.

The data embedded technique improves the encoder and the decoder respectively. Its dedication to encoder is the technique that data embeds into the MP3 without referencing the psychoacoustic model. In decoder the data embedded technique is ported on DSP platform, ADSP-2181, to realize the data embedded decoding. It could further be developed as a portable product. To sum up, the purpose of this method is to increase the applications of MP3 music media.

1.5 Content Organization

This thesis contains six chapters. Chapter 1 is in the premise. Chapter 2 introduces three methods which are used to embed data into the MP3 file in this thesis. In Chapter 3, the hardware and software environment where the MP3 decoder with data embedded decoder ported is developed are introduced. Chapter 4 presents the implementation and performance verification of these methods. This thesis finishes with conclusion and future works in Chapter 5. Appendix A introduces the MPEG-1 Layer III codec algorithm, which is including the brief principles and functionality. Appendix B introduces the ODG standard which is used for testing the music quality.

CHAPTER 2

Data Embedded Algorithm in the MPEG/Audio

In this chapter, we will describe the MPEG-1/Audio compression algorithm briefly and the MPEG-1/Audio format. This serves as the necessary background of understanding our MPEG-1/Audio data embedded schemes. The data embedded technique will be introduced in Section 2.2. It includes the principle, the application, and the classification of the data embedded algorithm. The data embedded algorithm that used to implement the data embedded codec about the MPEG/Audio in this thesis is introduced in Section 2.3. Section 2.3 introduces the data embedded encoder which includes the principles, the application, the advantages, and other methods to embed data into the MPEG/Audio. It introduces the data embedded decoder which extracts the embedded data.

2.1 Introduction to MPEG Audio

The ISO MPEG standard [3][4] contains four parts for compression standards shown in Fig. 1. The MPEG-1 is divided into five parts, namely system, video, audio, compliance testing, and software simulation. The MPEG-1 audio algorithm is an international standard for digital audio compression and does not make any assumptions about the nature of the audio source. It is suitable for audio-only applications as well as combined with video data (MPEG Systems Coding).

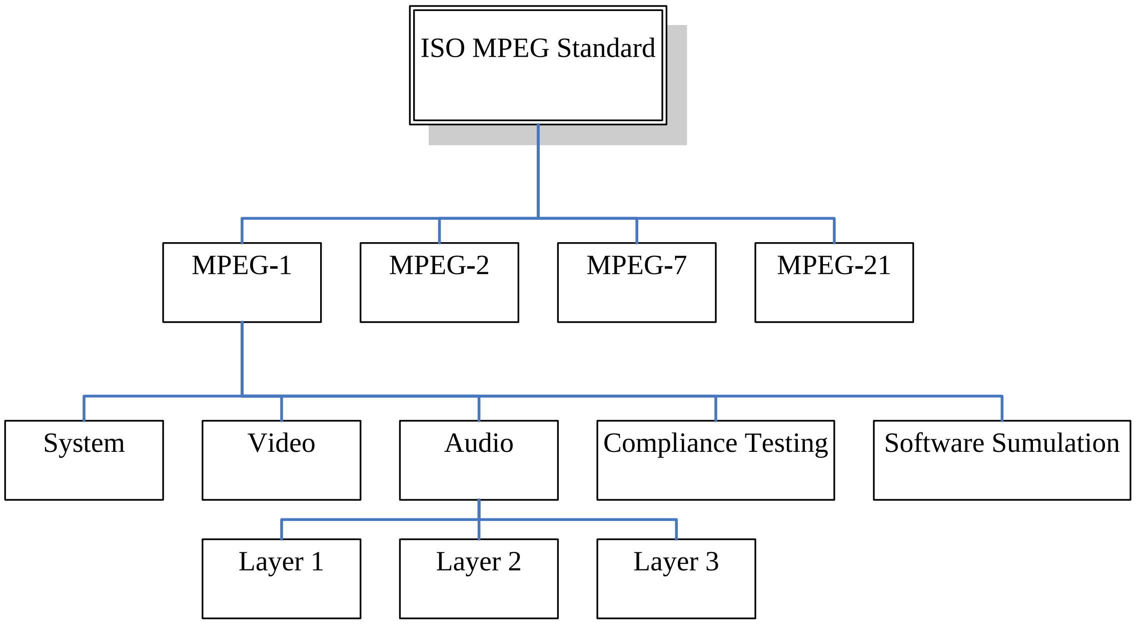


Fig. 1 The hierarchy of the ISO MPEG Standard

Depending on the applications, MPEG audio coding system can also be divided into three layers with increasing encoder complexity:

- **Layer I**

  Layer I contains the basic mapping of the audio samples into 32 subbands, fixed segmentation to format the data into blocks, a psychoacoustic model for the bit allocation, and quantization. It best suits the bit rate above 128Kbps per channel.

- **Layer II**

  Layer II provides additional coding of bit allocation, scale-factors and samples. It targets the bit rate around 128 Kbps per channel.

- **Layer III**

  Layer III introduces increased frequency resolution based on a hybrid filterbank. It uses non-uniform quantizer and entropy coding (Huffman Coding). It offers the best audio quality at the bit rate around 64 Kbps per channel.

The MPEG audio compression is a lossy algorithm and uses the special nature of the human auditory system (HAS). It removes the perceptually irrelevant parts of the audio and makes the audio signal distortions inaudible to the human ear, so it can provide compression ratios ranging form 2.7 to 24, see the Fig. 2. The compression ratios depend on different predefined fixed bit rates ranging from 32 kbps to 224 kbps.

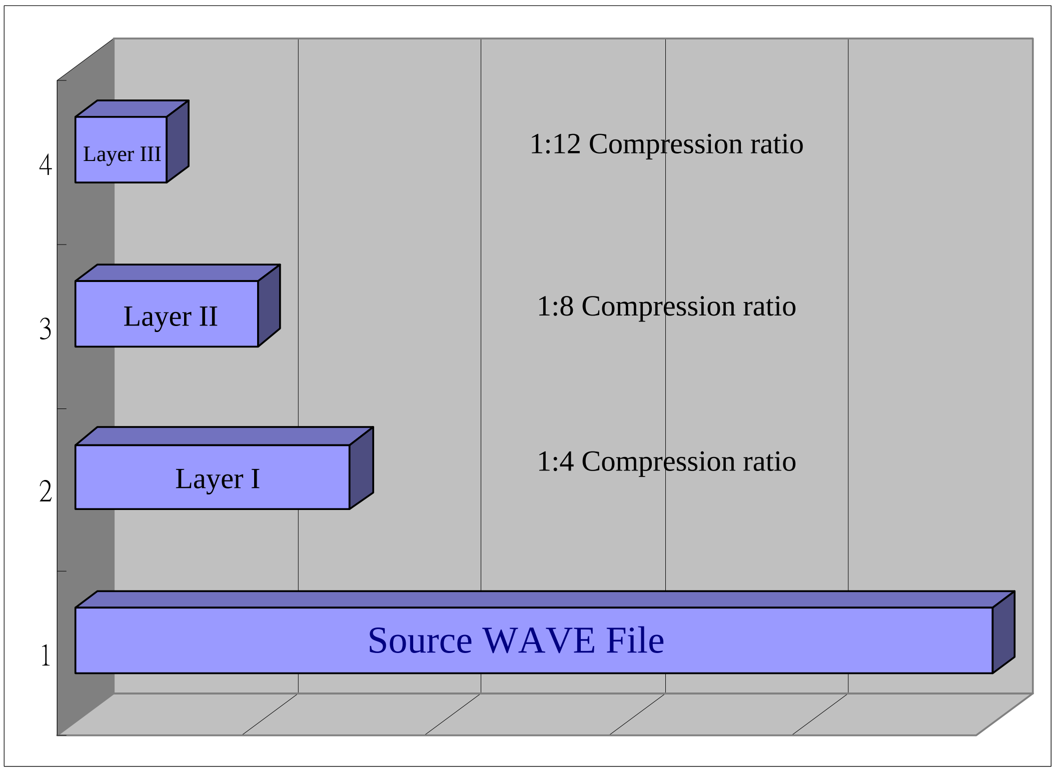


Fig. 2 The comparison of the ISO MPEG Audio standard compression ratio

2.1.1 Introduction of the MP3 Encoder Algorithm

The description of the encoding process is based on the block diagram in Fig. 3. 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 number 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.

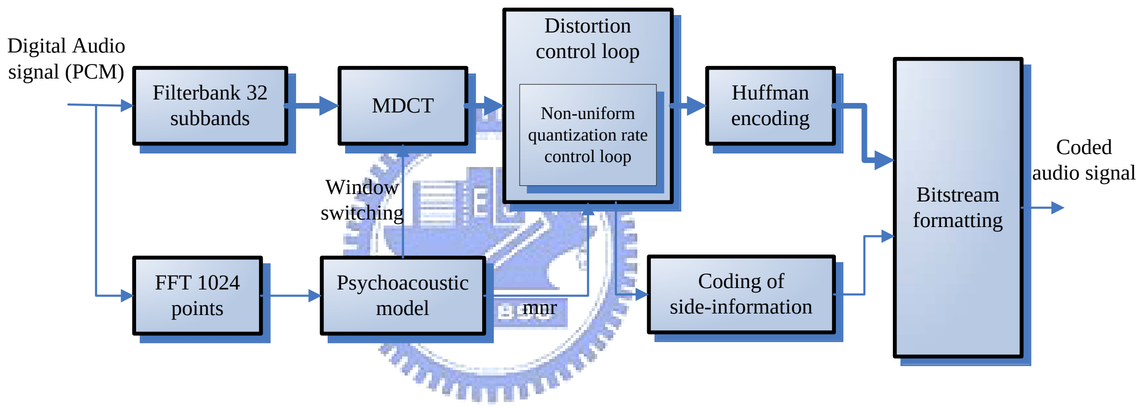


Fig. 3 MPEG-1/Audio Layer 3 encoder block diagram [5]

2.1.2 Introduction of the MP3 Decoder Algorithm

In this section the MPEG-1/Audio Layer III decoder will be described with its functionality. The decoding process is based on the block diagram in Fig. 4. 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 dequantized 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.

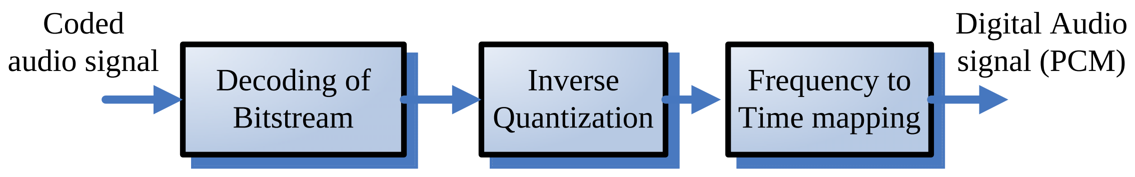


Fig. 4 MPEG-1/Audio Layer III decoder block diagram

2.2 Introduction of Data Embedded methods

There are many watermark techniques [8] in terms of their application areas and purposes. The technology of data embedded is a kind of watermarking. It is also related to the science of steganography. The word steganography is derived from the Greek words stegano (hidden) and pgrphein (to write) and therefore means “covered writing”. Data embedded of MPEG/Audio is a technique for the transmission of additional data along with audio signals existing distribution channels.

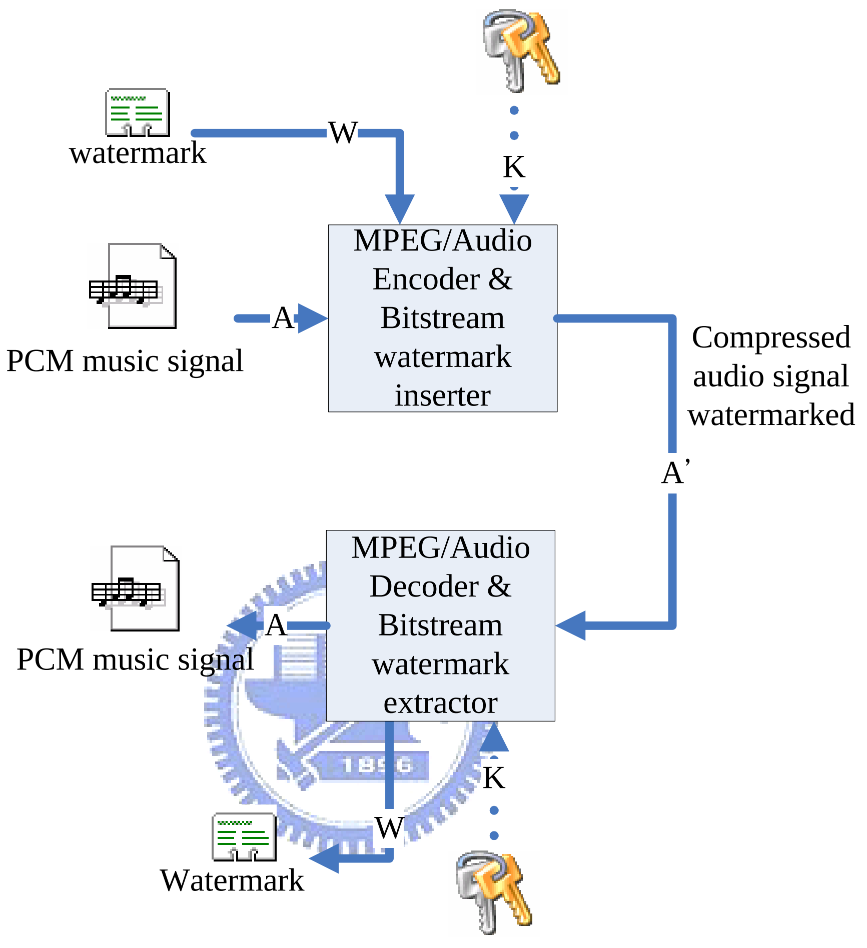
The principle, the characteristics, the applications, and the classifications are introduced in the following:

2.2.1 The Principle of Watermarking Algorithm

Mathematically, data embedded can be expressed like EQ 1. If an original audio signal A and a watermark W are given, the watermarked audio signal A' is represented as the following Eq. 1.

$$A' = A + f(A,W) \qquad \text{Eq. 1}$$

Fig. 5 shows the combination of the watermarking process which includes inserting and extracting watermark.



watermark
W
K
MPEG/Audio Encoder & Bitstream watermark inserter
PCM music signal
A
Compressed audio signal watermarked
A'
MPEG/Audio Decoder & Bitstream watermark extractor
PCM music signal
A
K
W
Watermark

Fig. 5 Combination of the watermarking process on MPEG/Audio

2.2.2 Main Characteristics for Watermarking Algorithm

There are many watermark characteristics, which may be required for an effective watermark, but the following main characteristics are important ones.

- **Invisibility**

  It is not able for human sense system to find the difference between the host media and watermark media. This is the essential requirement of all the data hiding system including watermarking system. This is why the watermark hidden in the audio must be music inaudible.

- **Robustness**

  Robustness, also an essential requirement is the ability to resist some of the signal processing operations, such as filtering, compression and the identifiable degree of the retrieved watermarks. The embedded algorithm must make chance to fight against the different kinds of signal processing operations. In general, the more robust the watermarking techniques have, the fewer capacities we can embed.

- **Security**

  After the watermark embedding, if someone wants to take out the embedded watermark, he must own some of the secret information related to the original signal. In general, to keep secret of the embedding algorithm is not easy, so the safety of the embedding system relies on the secret key which represents the location that watermark embedded. Using the secret key as the seed of the random number generator, we can get a serial random number and cooperate with an algorithm to embed the watermark. Therefore, the secret key is necessary to extract the watermark from the embedded media.

2.2.3 Applications of Watermarking Algorithm

- **Compatible Transmission of Data (Watermarks)**

  Basically watermarking algorithms provide a data transmission channel that can be used in existing distribution channels. The data hiding (watermark) transmission is backward compatible in the sense that every existing channel that is able to carry watermarked music. Hence watermarking can be utilized in a wide field of applications.

- **Digital Rights Management (DRM)**

Digital Rights Management is often considered as the main application of watermark. Watermark can provide means to fulfill the demands of DRM, such as proof of ownership, access control for digital media, tracing illegal copies and so on.

- **Broadcasting**

  A variety of applications for audio watermark are in the field of broadcasting. These include program type identification, advertising research, broadcast coverage research and etc.

2.2.4 Classification on Watermarking Techniques

The data embedded technique has different insertion and extraction methods, and may be classify and analyze these methods from the various points of view like in Table 1.

Table 1 Classification of watermarking according to several viewpoints [9]

Classification		Contents
Inserted media category		text, image, audio, video
Perceptivity of watermark		visible, invisible
Robustness of watermark		robust, semi-fragile, fragile
Inserting watermark type		noise, image, format
Processing method	Spatial domain	LSB, patchwork, random function
	Transform domain	Look-up table, spread spectrum
Necessary data for extraction		Private, semi-private, public watermarking
File size		Vary or not

2.3 Data Embedded Codec Algorithm for MPEG/Audio

In this section, the properties and the data embedded codec algorithm which includes several methods to embed data into the MPEG/Audio will be introduced.

2.3.1 The Properties of Data Embedded Codec

In this thesis, the MPEG/Audio signal is the inserted media because the technique of the data embedded bases on the specification of the MPEG/Audio. The embedded data is private information which is invisible and fragile. And the file type of embedded data can be any format or just be a series of bitstream. In other words, any data can be embedded into the MPEG/Audio media no matter what data type it is as long as the size of the embedded data is not bigger than the upper limit of the embedded data of the media.

There are three methods for data embedding, embedded data into count1 region, embedded data into bit reservoir, and modify the MP3 encoder from floating point to fixed point.

Recent research has produced a number of algorithms for embedding and retrieval of watermarks in audio signals [10] [11][12][13]. While most known systems operate in the uncompressed domain (PCM Watermarking), few are capable of embedding watermarks into compressed domain (Bitstream Watermarking) such as this thesis. The classification of the watermarking algorithm proposed in this thesis as mentioned above in Table 1 can be summarized and shown in Table 2:

Table 2 Classification of the watermarking technique in this thesis

Classification	Contents
Inserted media category	MPEG/Audio
Perceptivity of watermark	invisible
Robustness of watermark	fragile
Inserting watermark type	Any format
Processing method: Frequency domain	spread spectrum of high frequency
Necessary data for extraction	Public watermarking
File size of inserted media	No change

- **Inserted media category : MPEG/Audio**

  The data embedded method designed flow is based on the property of MPEG/Audio Specification. The MPEG-1 Layer-3 (MP3) is used for embedding data in this thesis. After MP3 encoder doses MDCT transformation which transforms signal from time domain to frequency domain, the frequency lines of the main data are distributed from low frequency to high frequency in a frame as shown in Fig. 41. Data is embedded into frequency domain by MP3 encoders, and extracted by MP3 decoders.

- **Perceptivity of watermark : Invisible**

  The embedded data as watermarks must be invisible because the inserted media file is the audio file. The embedded data can not either affect the quality of the original music or at least the affection can not be heard. The MP3 decoder with data embedded decoder can be used to extract the embedded data stream and reconstruct the embedded data stream to the original file.

- **Robustness of watermark : fragile**

  Embedding data into MP3 music is additional service by the content providers. But the purpose of embedding data is not to provide additional protection for MP3 music, on the contrary, the embedded data becomes fragile and easily distorted when the music is compressed. More robust the watermark is, less space for data embedding. Therefore the fragile method is preferred because more fragile the watermark is, more space for data embedding.

- **Inserting watermark type : any format**

  The data type that is embedded into the audio file can be any file format, because the embedded data stream has a header which records the synchronization, the embedded file size, the embedded file length, and the file data stream. The embedded data stream just is a series signal of “0” and ”1” whatever any files types are. The extractor in the MP3 decoder can extract the embedded data and an analyzer of embedded data can reconstruct the embedded files.

- **Processing method : Frequency domain**

  The embedded data is embedded into frequency lines of the frequency domain after the MDCT transformation which transforms from time domain to frequency domain.

- **Necessary data for extraction : public watermark**

  The embedded data belongs to public watermark. The embedded data only can be extracted by a special decoder.

- **File size : no change**

  After the data embeds into the MP3 file, the MP3 file size that is embedded data is the same to the MP3 file that is encoded by other MP3 encoder. One file is encoded by MP3 encoder with data embedded encoder, and the other one is encoded by any other MP3 encoders in the same bitrate and sampling rate. The size of the two MP3 files is the same, if they compare to each other. They just can be differentiated by the MP3 decoder with embedded data analyzer. The one which embedded data can extract embedded information but the other one can't.

2.3.2 The Structure of Data Embedded Codec

The data embedded codec are divided into two parts: one part is the data embedded encoder, and the other part is data embedded decoder. The data embedded encoder usually is used for content provider to provide additional service which embed lyrics, the basic information of singer, the photos of the singer, and even the information of customer into the MP3 audio. Almost all information can be embedded into the MP3 file under the upper bond of the size of the embedded data. The data embedded decoder is used for users and combines with the MP3decoder. It can extract all the information that is embedded in the MP3 files and display the information on the monitor. The MP3 decoder with data embedded decoder has also ported on the ADSP-2181 to become a portable device.

Fig. 6 indicates the structure of data embedded encoder. There are two source data for encoding: one is the audio raw data, and one is the embedded data. If there are too many embedded files input into the encoder at the same time, the encoder will confuse the files. And it causes the decoder could not extract the embedded data. The embedded data does not just include only one file. It may include two files or more, so a package program is designed in order to pack all the files to become a file with special format for encoding. The packaged file and the audio raw data input to the MP3 encoder with data embedded encoder together, and the encoder will output a MP3 file with embedded data. The file size after embedding data is the same to the file size which is encoded by other MP3 encoder. The MP3 file which embeds data can also be played by any general MP3 player, and the embedded data won’t affect the quality of the music.

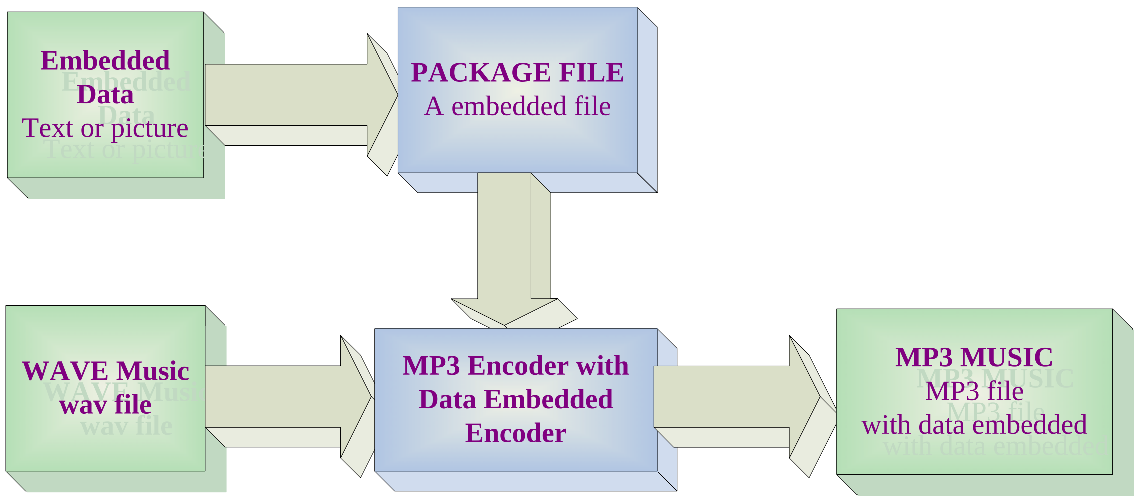


Fig. 6 The structure of data embedded encoder

Fig. 7 indicates the structure of data embedded decoder. The decoder structure is the inverse flow of the encoder. The MP3 file with embedded data as the input data inputs to the MP3 decoder with data embedded decoder. The decoder has two output ends: one is the music raw data, and the other one is the embedded data stream. The music raw data is the same music of CD quality which decodes by other general MP3 decoder. The embedded data stream has to input the data stream analyzer to analyze, and the data stream analyzer reconstructs the original embedded files. And the files would be shown on the displayer or save as files in the disk.

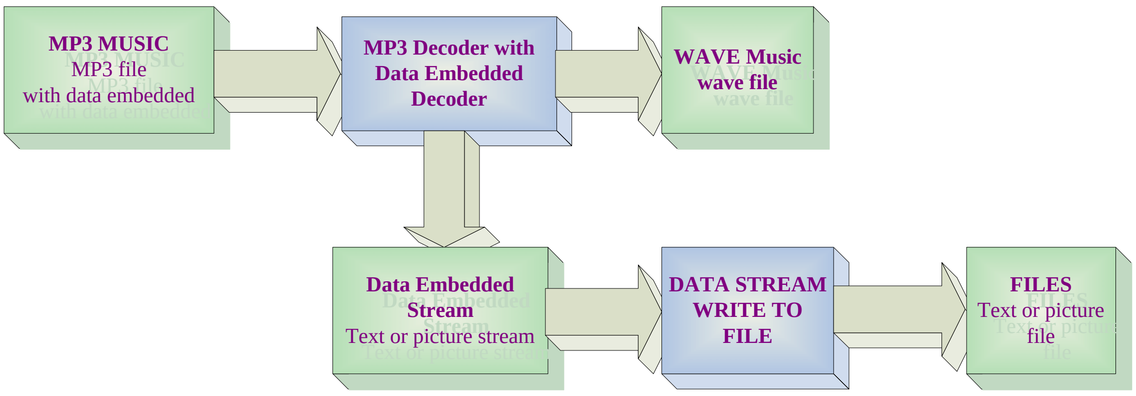


Fig. 7 The structure of data embedded decoder

2.3.3 The Methods of Data Embedded Codec

In this section, there are some methods for data embedding. They are introduced in the following subsection:

2.3.3.1 Embedded Data into the Count1 Region

The count1 region saves the frequency lines which distribute on the relative high frequency in a frame. And the energy of the count1 region is small than the energy of big-value region. So the method of embedded data into count1 region can affect the quality of the music small.

General watermarking techniques reference the absolute threshold of hearing of the psychoacoustic model [14] in the music compressing technique, as shown in Fig. 8. The signal energy can't be heard by people under the absolute threshold of hearing, and the watermark usually hides under the absolute threshold of hearing, too. The signal of the embedded data can't be heard by people, so it would not affect the quality of the music.

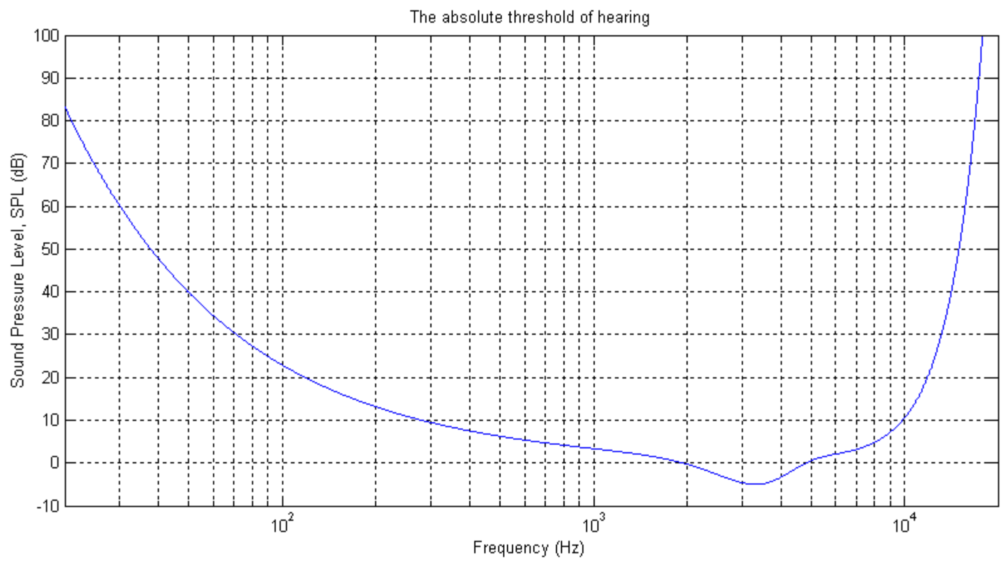


Fig. 8 The Absolute Threshold of Hearing

The computation of the psychoacoustic model is a great quantity of ratio in the MP3 encoder, and it accounts about 20% computation of the MP3 encoder. The quality of the MP3 music after the MP3 encoder encodes without psychoacoustic model and the bit rate sets 128kbps. The general bitrate of MP3 is almost 128kbps now, but a few songs even uses 128kbps for more high quality music. There are few songs encoded by 96kbps or less, because the quality is a little ugly. In order to speed up the encoding time of the MP3 encoder, the psychoacoustic model of the MP3 encoder would be removed for embedded system.

The MP3 encoding speed is speed up after the psychoacoustic model of the MP3 encoder is removed. On the other side, it is not good for data embedded techniques. The data embedded techniques would easily destroy the quality of the music without the reference of the psychoacoustic model. So should embed information in a situation without psychoacoustic model, must look for other places that can embed information in the music. The main condition of the place would not affect the original quality of the music or the affect to the quality should be the lowest.

In this thesis, the method of data embedded bases on a principle that the sensitive degree of different frequency bands for ears of people is different. The sound of low frequency for common people’s ears, no matter how the loud voice of the sound is or where the source of the sound is more relatively sensitivity to distinguish coming out. But people’s ears are relatively insensitive to high frequency sound. The property is used during MP3 encoding. The property is that people’s ears can’t distinguish the phase of the high frequency.

The MP3 media data embedded technique is designed to utilize the different degree of sensitiveness of human ears to different sound band. Normally human ears, despite the volume or source of the sounds, are more sensitive to those with the phase of lower frequency but are less sensitive to those with the phase of higher frequency. Using this characteristic we develop modified MP3 coding technique, embedding the data in high frequency sound band when compressing MP3 data files to decrease the negative influence of the quality of the sounds. Then a MP3 media data decoder is being developed. The embedded data will be shown on the screen at the same time during decoding.

2.3.3.2 Embedded Data into the Big-Value Region

The method of embedding data into the bit value region is the extended method of Section 2.3.3.1, which embeds data into the count1 region. This method uses the sign bits of the big-value region for embedding data, and it is the same as Section 2.3.3.1, which the sign bits of the count1 region, is used for embedding data.

The property that people is less sensitive to the phase of the high frequency is used in the last section. According to this property, the embedded data is used to replace the sign bits which represent the phase of the high frequency in the count1 region. For the same reason, the sign bits that represent the phase of the relative high frequency can be used for embedding data. People are sensitive to the sound of low frequency, so the embedded data can not replace all the sign bits in the big-value region. It will cause lots of distortion. The relation between changing the sign bit in the big-value region and the quality of the music is pretty close especially the quality of the low frequency music. For this reason, the lower limit of the frequency must be searched to make sure that the embedded data won’t result in distortion.

There are 6,930 frames in a song which plays three minutes and four granules in a frame. Therefore, there are 27,720 granules in a song. Changing the sign bits of the count1 region doesn't influence the music quality obviously because the count1 region represents the part of the relatively high frequency signal in every granule. The average starting boundary of the count1 region is the 310th frequency line in the granule through the frequency lines statistic. And it maps to the real frequency is 11.84 KHz, as shown in Fig. 9. That is to say, without considering where the starting point of the count1 region is, the 310th frequency line lies in the big-value region. When the data embeds in the sign bits of the frequency lines after the 310th one is replaced, the influence upon the music quality is small.

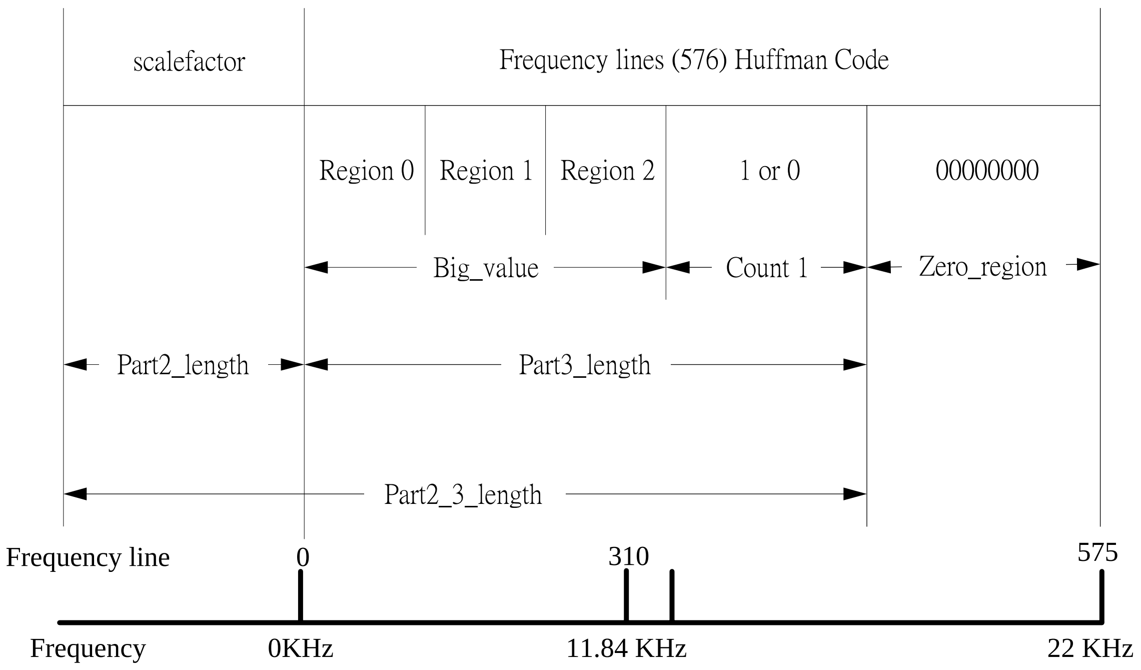


Fig. 9 The frequency line mapping to frequency

The data space for embedding data into the big-value region is bigger than embedding data into the count1 region. But the energy of the signals in the big-value region is bigger than those in the count1 region. On the other hand, data embedded into the big-value region has greater influence upon the music quality than the data embedded into the count1 region. If the quality of the music is highly considered, the latter one is preferred.

Fig. 10 and Fig. 11 are shown two distribution of music quality and bit numbers in big-value region. The meanings of the two figures are the different variations of music quality and bit numbers which embeds data from different frequency in the big-value region. In Fig. 10, if user embeds data from 0K Hz, he can get a quantity of space about 620Kbytes for embedding data. But the music quality will be too worse to hear, the ODG [20] is about -3.2. If user dose not need a big space for embedding data and needs a better music quality, he can embed data from higher frequency, for example, 10K Hz. The selected frequency for embedding data is a trade off between the music quality and the bit numbers. Or user can embed data into the other two regions that provides in this thesis. Embedded data into the big-value region affects the music quality easily, so the big-value region is the last region that we suggest for embedding data in the three methods.

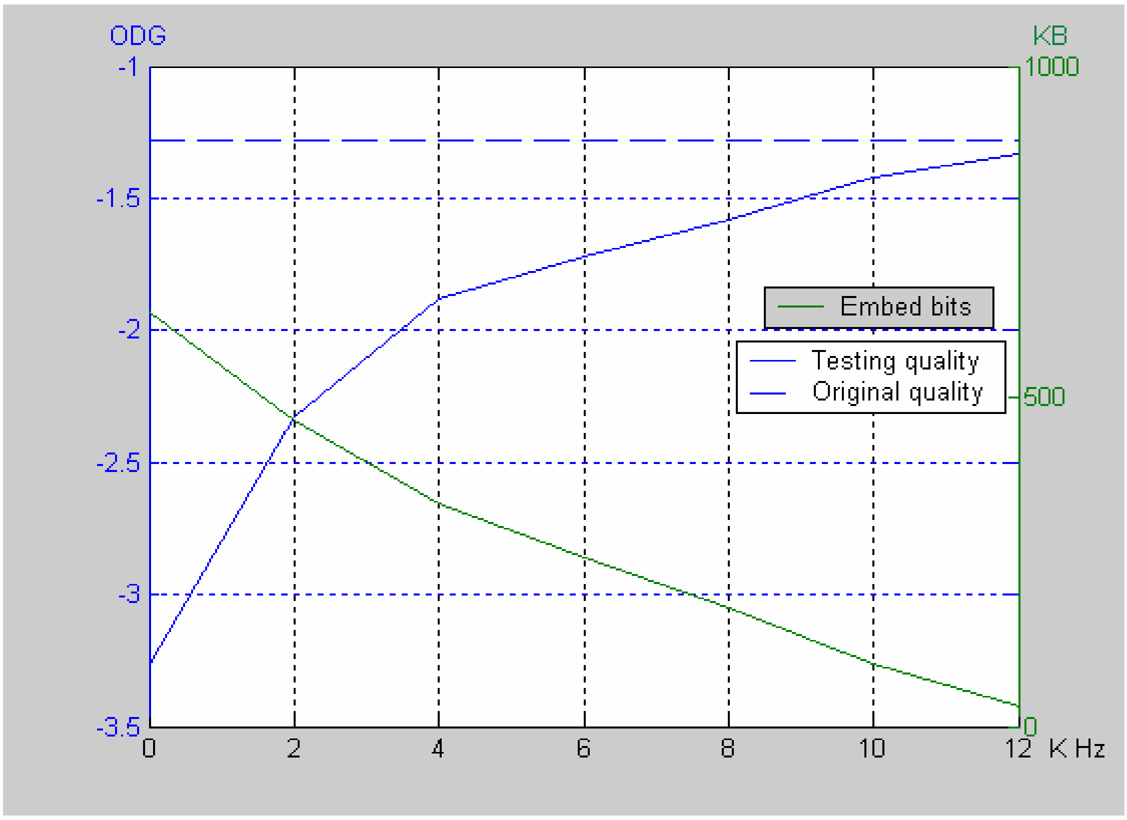


Fig. 10 The distribution of music quality and bits in the big-value region

(Sample name: 01-can)

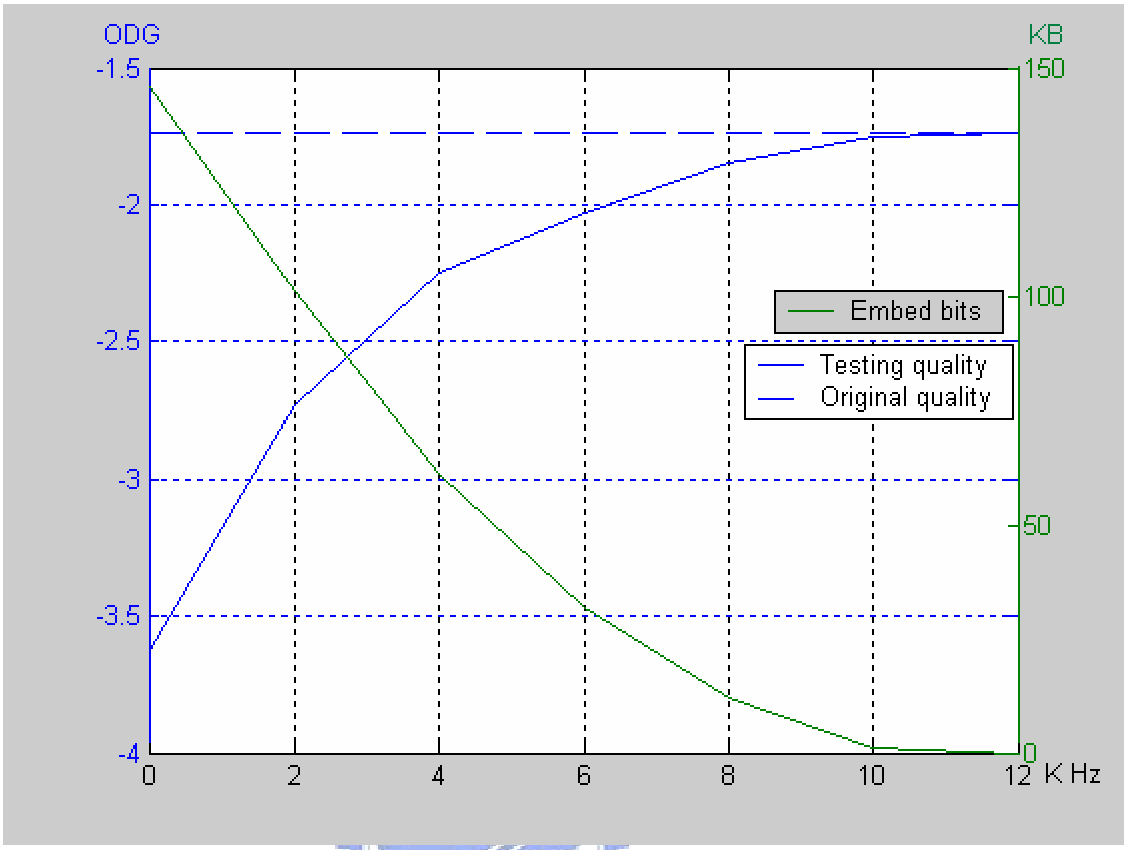


Fig. 11 The distribution of music quality and bits in the big-value region

(Sample name: speech)

2.3.3.3 Embedded Data into Bit Reservoir

In the MP3 encoder algorithm, an enhancement method called “bit reservoir” is used to fit 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, as shown in Fig. 12.

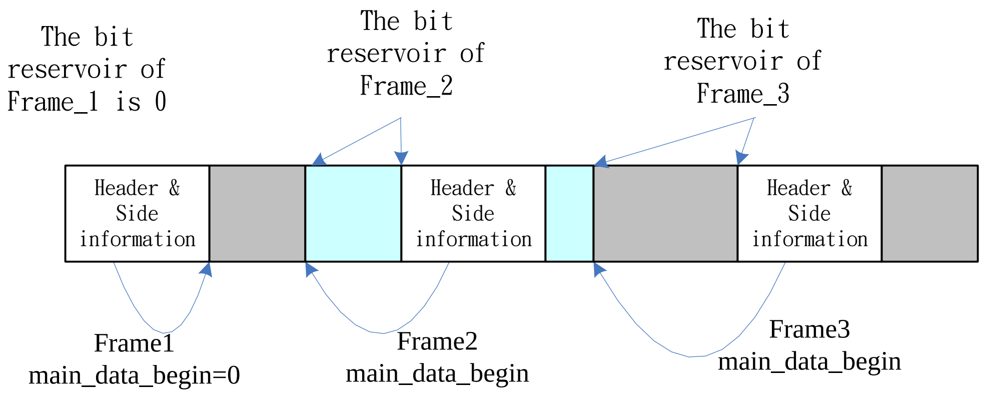

The bit reservoir of Frame_1 is 0
The bit reservoir of Frame_2
The bit reservoir of Frame_3
Header & Side information
Header & Side information
Header & Side information
Frame1 main_data_begin=0
Frame2 main_data_begin
Frame3 main_data_begin

Fig. 12 The bitstream and bit reservoir of MP3

In bit reservoir, there is a 9-bit flag to record the beginning point of the reservoir. If the space of the reservoir is larger than 512 bytes, the excess space will have to be filled in "1" and cannot be further utilized. Therefore these bits are all wasted, as shown in Fig. 13. It is often the case at the quiet sound part in the beginning and the end of the music, which mostly the entire frame is filled in "1". So these bits could be used for data embedding.

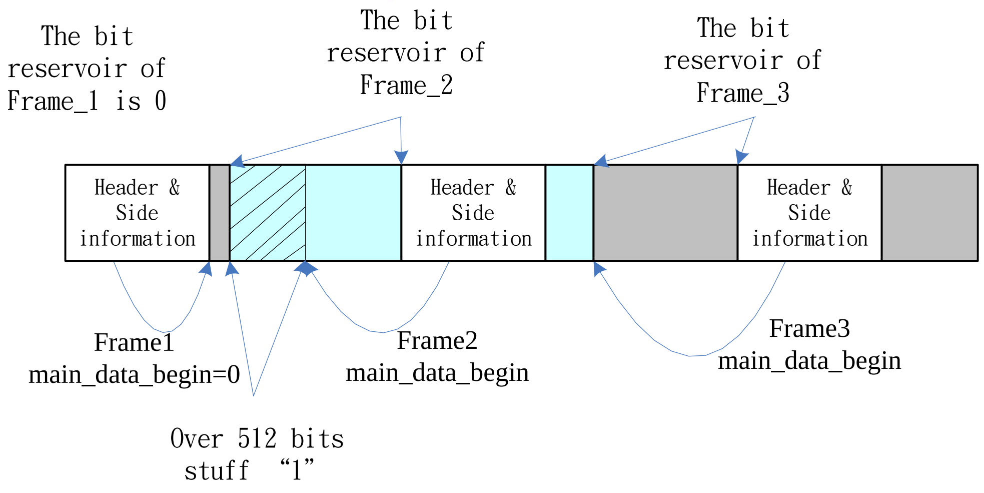

The bit reservoir of Frame_1 is 0
The bit reservoir of Frame_2
The bit reservoir of Frame_3
Header & Side information
Header & Side information
Header & Side information
Frame1 main_data_begin=0
Frame2 main_data_begin
Frame3 main_data_begin
Over 512 bits stuff "1"

Fig. 13 The reservoir over 512 bytes and stuff "1"

Embedding data into unutilized bit reservoir space has a huge advantage： It will not influence the quality of the music at all. Because in MP3 decoding process, when encountering redundant bit reservoir, the decoder will just read “1” from the bitstream but not decode it. Therefore using these space filled in “1” to embed data will not influence the quality of the music.

CHAPTER 3

Environment of Hardware and Software

In this chapter, the hardware and software environment are described briefly. The data embedded algorithm is ported on the ADSP hardware, and the data embedded algorithm is developed by the VisualDSP which is the software environment. The hardware is concerned with the development of programs while the software influences the development speed and performance.

3.1 Hardware Environment

3.1.1 ADSP-2181 EZ-KIT Lite Board

The hardware used in this thesis is AnalogDevice ADSP-2181 EZ-Kit Lite board. The EZ-KIT Lite consists of a small ADSP-2181 based development/demonstration board with full 16-bit stereo audio I/O capabilities. The board's features which shown in Fig. 14 include:

- **ADSP-2181 16-bit 33 MIPS DSP**
- **AD1847 Stereo SoundPort**
- **RS-232 Interface**
- **Socketed EPROM**
- **User Pushbuttons**
- **Power Supply Regulation**
- **Expansion Connectors**
- **User Configurable Jumpers**

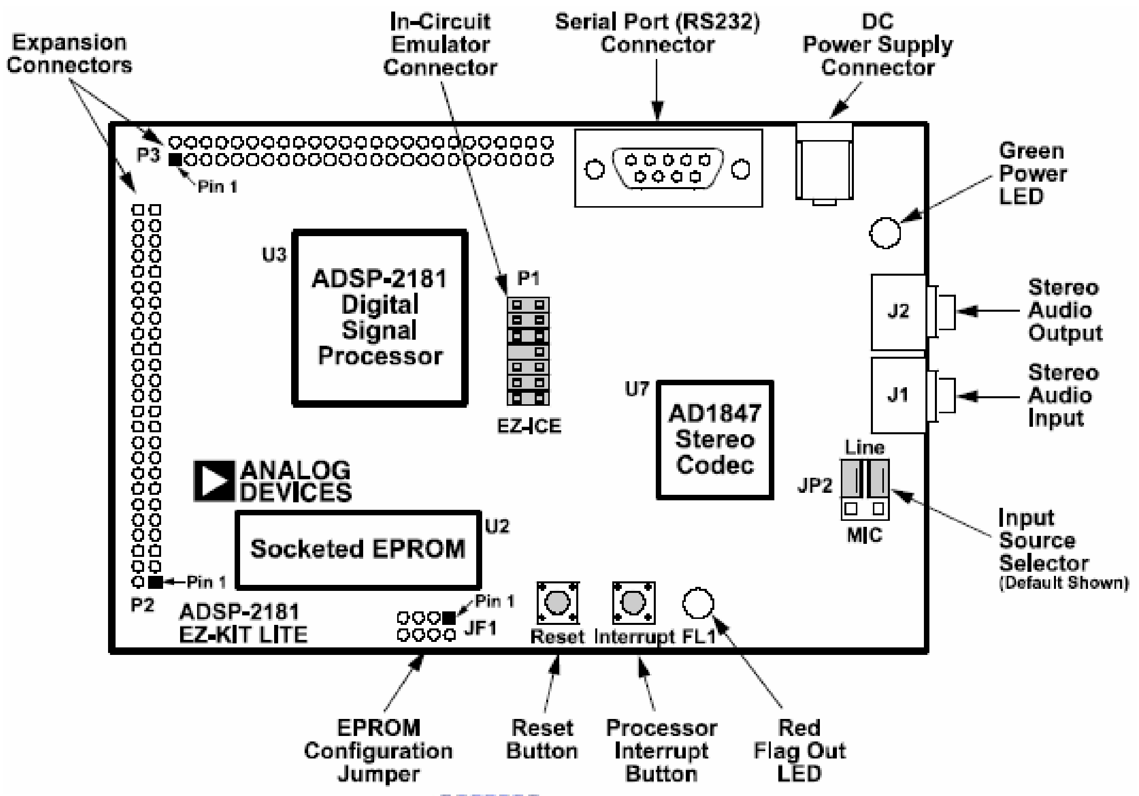


Fig. 14 EZ-KIT Lite's functional block diagram [14]

The board can run standalone or can simply connect to the RS-232 port of the PC. A monitor program running on the DSP in conjunction with a host program running on the PC interactively download programs as well as interrogate the ADSP-2181. The board comes with a socketed EPROM so that we can download the MP3 codec with data hiding algorithm into the EPROM.

3.1.2 ADSP-2181 Microprocessor

The ADSP-2181 is a programmable single-chip microprocessor that uses a common base architecture optimized for digital signal processing (DSP) and other high-speed numeric processing applications. Fig. 15 shows the main functional units of the ADSP-2181 architecture which functions are included in the processor.

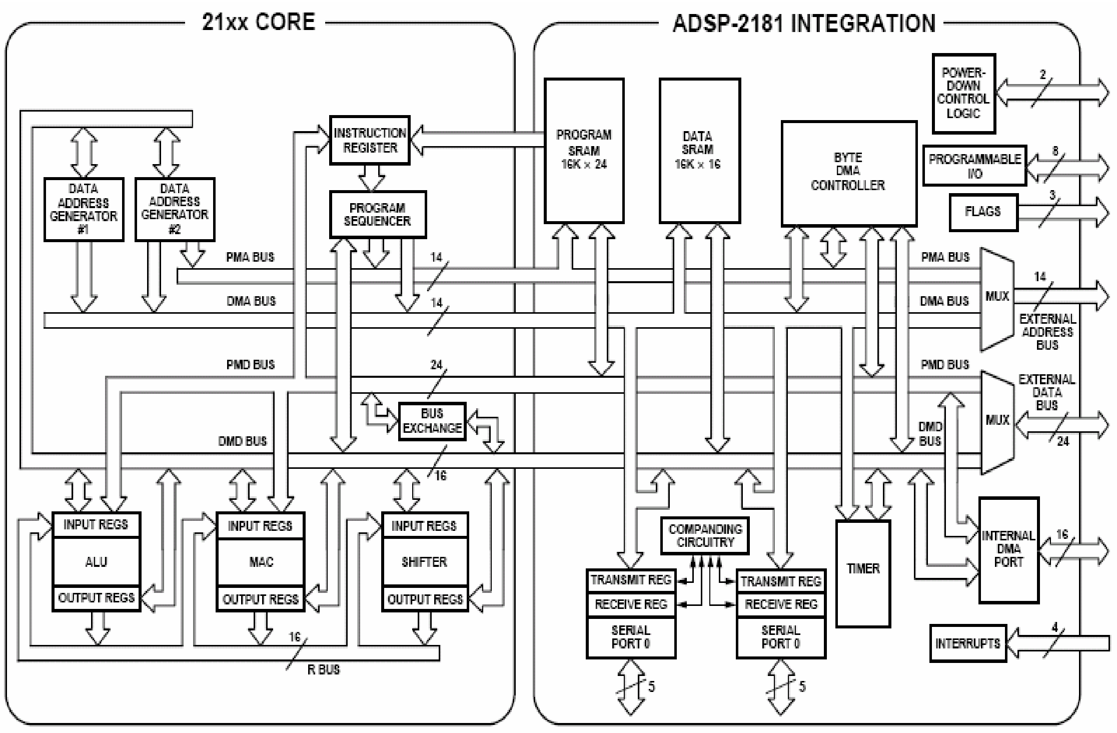


Fig. 15 ADSP-2181 functional block diagram [16]

- **Computational Units**

  The ADSP-2181 processor contains three independent, full-function computational units: an arithmetic/logic unit (ALU), a multiplier/accumulator (MAC) and a barrel shifter. The computational units process 16-bit data directly and also provide hardware support for multiprecision computations.

- **Data Address Generators & Program Sequencer**

  Two dedicated address generators and a program sequencer supply addresses for on-chip or external memory access. The sequencer supports single-cycle conditional branching and executes program loops with zero overhead. Dual data address generators allow the processor to generate simultaneous addresses for dual operand fetches. Together the sequencer and data address generators keep the computational units continuously working, maximizing throughput.

- **Memory**

The ADSP-2181 uses a modified Harvard architecture in which data memory stores data, and program memory stores both instructions and data, as shown in Fig. 16. The ADSP-2181 contains on-chip RAM that comprises a portion of the program memory space and data memory space. The speed of the on-chip memory allows the processor to fetch two operands (one from data memory and one from program memory) and in instruction (from program memory) in a single cycle.

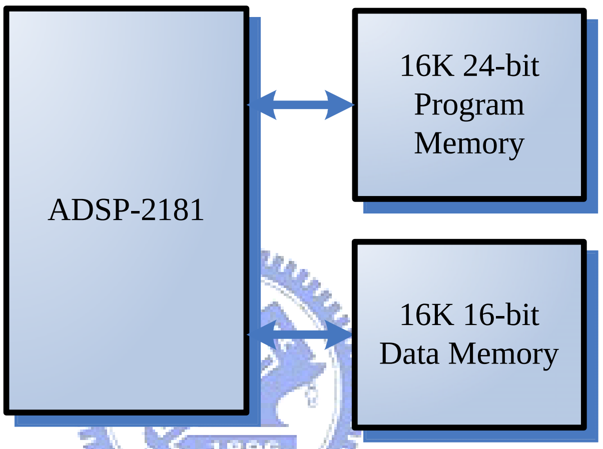


Fig. 16 Harvard architecture

- **Serial Ports**

  The serial ports (SPORTs) provide a complete serial interface with hardware companding for data compression and expansion. Both μ-law and A-low companding are supported. The SPORTs interface easily and directly to a wide variety of popular serial devices. Each SPORT can generate a programmable internal clock or accept an external clock. SPROT0 includes a multichannel option.

- **Timer**

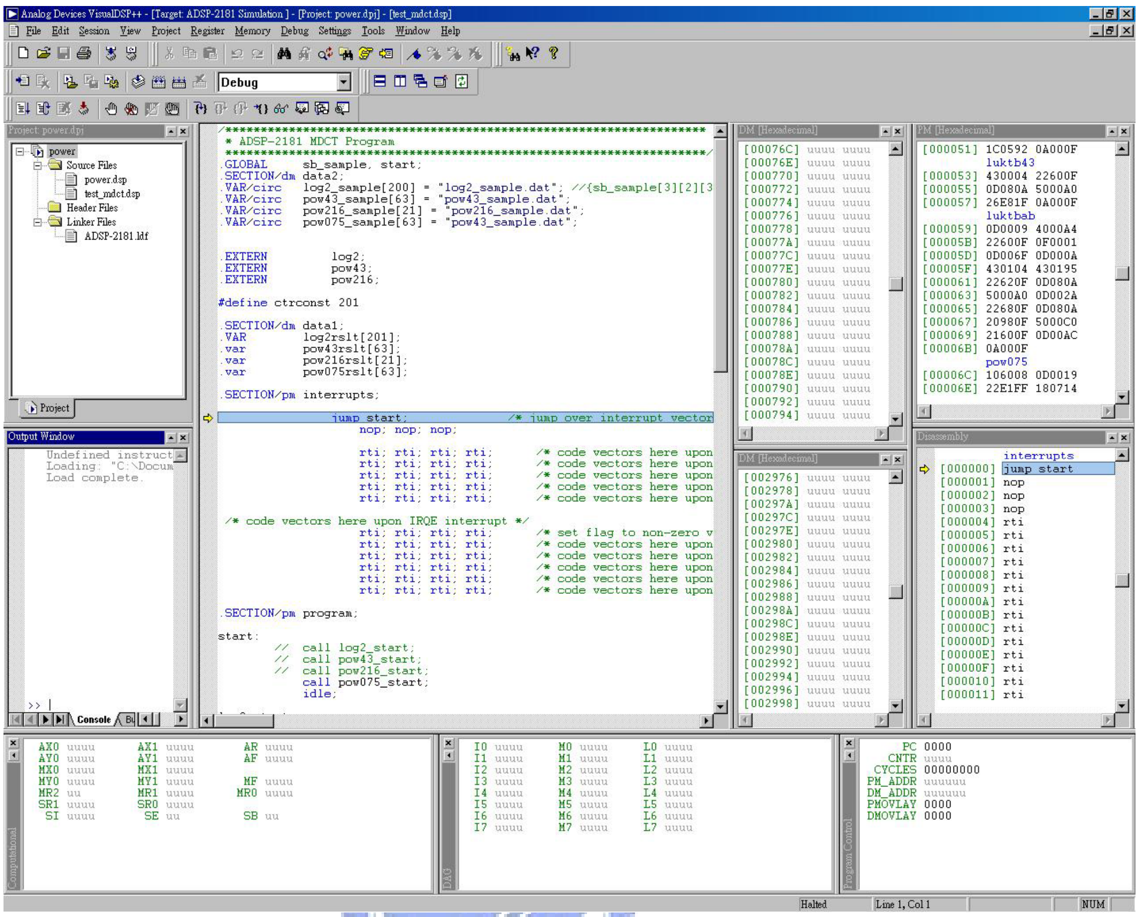
  A programmable timer/counter with 8-bit prescaler provides periodic interrupt generation.

- **DMA Ports**

The ADSP-2181's Internal DMA Port (IDMA) and Byte DMA Port (BDMA) provide efficient data transfers to and form internal memory. The IDMA port has a 16-bit multiplexed address and data bus and supports 24-bit program memory. The IDMA port is completely asynchronous and can be written to while the ADSP-2181 is operating at full speed. The byte memory DMA port allows boot loading and storing of program instructions and data.

3.2 Software Environment

The ADSP software offers a PC based debugger environment, called VisualDSP++ [17] which user can develop quickly and debug easily in programming stage. Fig. 17 shows the VisualDSP++ user interface. This software development support enables user to develop DSP applications.



```
Undefined instruct
Loading: "C:\Docum
Load complete.
```

```
/*********************************************************************
 * ADSP-2181 MDCT Program
 *********************************************************************/
.GLOBAL      sb_sample, start;
.SECTION/dm data2;
.VAR/circ    log2_sample[200] = "log2_sample.dat"; //{sb_sample[3][2][3
.VAR/circ    pow43_sample[63] = "pow43_sample.dat";
.VAR/circ    pow216_sample[21] = "pow216_sample.dat";
.VAR/circ    pow075_sample[63] = "pow43_sample.dat";


.EXTERN          log2;
.EXTERN          pow43;
.EXTERN          pow216;

#define ctrconst 201

.SECTION/dm data1;
.VAR         log2rslt[201];
.var         pow43rslt[63];
.var         pow216rslt[21];
.var         pow075rslt[63];

.SECTION/pm interrupts;

                 jump start;                /* jump over interrupt vector
                     nop; nop; nop;

                     rti; rti; rti; rti;         /* code vectors here upon
                     rti; rti; rti; rti;         /* code vectors here upon
                     rti; rti; rti; rti;         /* code vectors here upon
                     rti; rti; rti; rti;         /* code vectors here upon
                     rti; rti; rti; rti;         /* code vectors here upon

 /* code vectors here upon IRQE interrupt */
                     rti; rti; rti; rti;         /* set flag to non-zero v
                     rti; rti; rti; rti;         /* code vectors here upon
                     rti; rti; rti; rti;         /* code vectors here upon
                     rti; rti; rti; rti;         /* code vectors here upon
                     rti; rti; rti; rti;         /* code vectors here upon
                     rti; rti; rti; rti;         /* code vectors here upon

.SECTION/pm program;

start:
        //  call log2_start;
        //  call pow43_start;
        //  call pow216_start;
            call pow075_start;
            idle;
```

```
[00076C] uuuu uuuu
[00076E] uuuu uuuu
[000770] uuuu uuuu
[000772] uuuu uuuu
[000774] uuuu uuuu
[000776] uuuu uuuu
[000778] uuuu uuuu
[00077A] uuuu uuuu
[00077C] uuuu uuuu
[00077E] uuuu uuuu
[000780] uuuu uuuu
[000782] uuuu uuuu
[000784] uuuu uuuu
[000786] uuuu uuuu
[000788] uuuu uuuu
[00078A] uuuu uuuu
[00078C] uuuu uuuu
[00078E] uuuu uuuu
[000790] uuuu uuuu
[000792] uuuu uuuu
[000794] uuuu uuuu
```

```
[000051] 1C0592 0A000F
         luktb43
[000053] 430004 22600F
[000055] 0D080A 5000A0
[000057] 26E81F 0A000F
         luktbab
[000059] 0D0009 4000A4
[00005B] 22600F 0F0001
[00005D] 0D006F 0D000A
[00005F] 430104 430195
[000061] 22620F 0D080A
[000063] 5000A0 0D002A
[000065] 22680F 0D080A
[000067] 20980F 5000C0
[000069] 21600F 0D00AC
[00006B] 0A000F
         pow075
[00006C] 106008 0D0019
[00006E] 22E1FF 180714
```

```
[002976] uuuu uuuu
[002978] uuuu uuuu
[00297A] uuuu uuuu
[00297C] uuuu uuuu
[00297E] uuuu uuuu
[002980] uuuu uuuu
[002982] uuuu uuuu
[002984] uuuu uuuu
[002986] uuuu uuuu
[002988] uuuu uuuu
[00298A] uuuu uuuu
[00298C] uuuu uuuu
[00298E] uuuu uuuu
[002990] uuuu uuuu
[002992] uuuu uuuu
[002994] uuuu uuuu
[002996] uuuu uuuu
[002998] uuuu uuuu
```

```
         interrupts
[000000] jump start
[000001] nop
[000002] nop
[000003] nop
[000004] rti
[000005] rti
[000006] rti
[000007] rti
[000008] rti
[000009] rti
[00000A] rti
[00000B] rti
[00000C] rti
[00000D] rti
[00000E] rti
[00000F] rti
[000010] rti
[000011] rti
```

```
AX0 uuuu     AX1 uuuu      AR uuuu
AY0 uuuu     AY1 uuuu      AF uuuu
MX0 uuuu     MX1 uuuu
MY0 uuuu     MY1 uuuu      MF uuuu
MR2 uu       MR1 uuuu     MR0 uuuu
SR1 uuuu     SR0 uuuu
 SI uuuu      SE uu        SB uu
```

```
I0 uuuu     M0 uuuu     L0 uuuu
I1 uuuu     M1 uuuu     L1 uuuu
I2 uuuu     M2 uuuu     L2 uuuu
I3 uuuu     M3 uuuu     L3 uuuu
I4 uuuu     M4 uuuu     L4 uuuu
I5 uuuu     M5 uuuu     L5 uuuu
I6 uuuu     M6 uuuu     L6 uuuu
I7 uuuu     M7 uuuu     L7 uuuu
```

```
     PC 0000
   CNTR uuuu
 CYCLES 00000000
PM_ADDR uuuuuu
DM_ADDR uuuuuu
PMOVLAY 0000
DMOVLAY 0000
```

Fig. 17 VisualDSP user interface

VisualDSP++ provides the following features:

Extensive editing capabilities

Create and modify source files by using multiple language syntax highlighting, drag-and-drop, bookmarks, and other standard editing operations. View files generated by the code development tools.

Flexible project management

Specify a project definition that identifies the files, dependencies, and tools that is used to build projects. Create this project definition once or modify it to meet changing development needs.

Easy access to code development tools

Analog Devices provides the following code development tools: C/C++ compiler, VIDL compiler, assembler, linker, splitter, and loader. Specify options for these tools by using dialog boxes instead of complicated command line scripts. Options that control how the tools process inputs and generate outputs have a one-to-one correspondence to command line switches. Define options for a single file or for an entire project. Define these options once or modify them as necessary.

Flexible project build options

Control builds at the file or project level. VisualDSP++ enables to build files or projects selectively, update project dependencies, or incrementally build only the files that have changed since the previous build. View the status of the project build in progress. If the build reports an error, double-click on the file name in the error message to open that source file. Then correct the error, rebuild the file or project, and start a debug session.

VisualDSP++ Kernel (VDK) Support

Add VDK support to a project to structure and scale application development. The Kernel tab page of the Project window enables to manipulate kernel objects such as events, event bits, priorities, semaphores, and thread types.

Flexible workspace management

Create multiple workspaces and quickly switch between them. Assigning a different project to each workspace enables to build and debug multiple projects in a single session.

Easy movement between debug and build activities

Start the debug session and move freely between editing, build, and debug activities.

CHAPTER 4

Implementation of Data Embedded Codec

In this section, the design of data embedded codec will be introduced. In Section 4.1, the flow of design will be discussed, which includes packaging embedded data and designing MP3 encoder with embedded data. Section 4.2 will introduce the design flow of data embedded decoder, which includes the MP3 decoder with data extracting decoder and the data embedded analyzer.

4.1 Data Embedded Encoder

The data embedded encoder contains two parts: packaging program and the main data embedding program.

It is very important that embedded data cannot affect the quality of the music. Yet another requirement is that the embedded data after being extracted by decoder must be exactly the same as it was encoded by encoder. The bit stream of the embedded data embeds audio file in series type, and the embedding data format must be pre-defined otherwise the extracted bit stream would not be recognized by data analyzer. Thus, a program package the files which will be embedded into the MP3 is needed, and the package format also needs to be defined.

4.1.1 Package Embedded Data

The embedded data usually has several files, not just one. These files are embedded in series but not parallel, and cannot overlap with one another. If the files mix up with one another, the embedded data cannot be extracted by the MP3 decoder. Therefore the embedded files must first be packaged.

Here is another problem that should be taken into consideration: the start point and the end point of the files embedded, as well as the data embedded in series type, are not known by the decoder. Therefore a header is defined for the embedded files and is packaged in front of the embedded files. The header contains three parameters: “synchronization bits”, “file length bits”, and “file type bits”, as shown in Table 3. The first parameter, “synchronization bits” is defined to synchronize the start point of the embedded file. It is accounted 4 bytes and has a value of 02040608. The second parameter, “file length bits” is defined to record the size of the embedded file. It is accounted 2 bytes and its value is the same as the size of the embedded file. The last parameter, “file type bits” is defined to record the file type embedded into the MP3 file. The maximum file size it can define is 64K bytes. The “file type bits” is accounted 1 byte. The value 00 represents the txt file, 01 represents the jpg file, and 02 represents the gif file, etc.. The “file type bits” can define as much as 256 types of files.

Table 3 The parameters of the header in the package file

Parameters	Size	value	maximum
Synchronization bits	4 bytes	02040608	
File length bits	2 bytes	The size of the file	64K Bytes
File type bits	1 bytes	00:txt 01:jpg 02:gif	256 types

The packaged file contains all the files that will be embedded into the MP3 file. The embedded files connect one another and every embedded file has an individual header in front of the file, as shown in Fig. 18. The packaged files can be added on before the total size exceeds the maximum embedded capacity of the MP3 file.

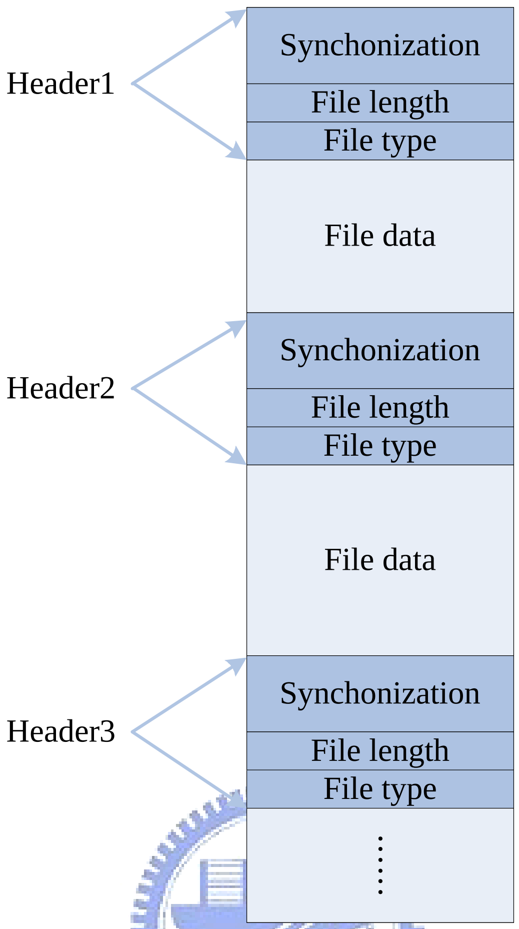


Fig. 18 The format of the packaged file

The following Fig. 19 is a flowchart shows how the packaging program is designed. The parameters of the embedded file are read and saved into a register first. The lyrics must start at the beginning of the music, so the text file is sorted at the first. Therefore lyrics have first priority, and pictures shown on the display screen during which the music is playing come in second. Next, the program identifies the file length and file type for adding header at the next stage in the flowchart. Then the embedded file is copied into the packaged file behind the header. If there are other files to be embedded into the MP3, the program jumps back to the third stage of the flowchart for getting the new synchronization, file length and file type. The loop will run continuously until there is no files need embedding further.

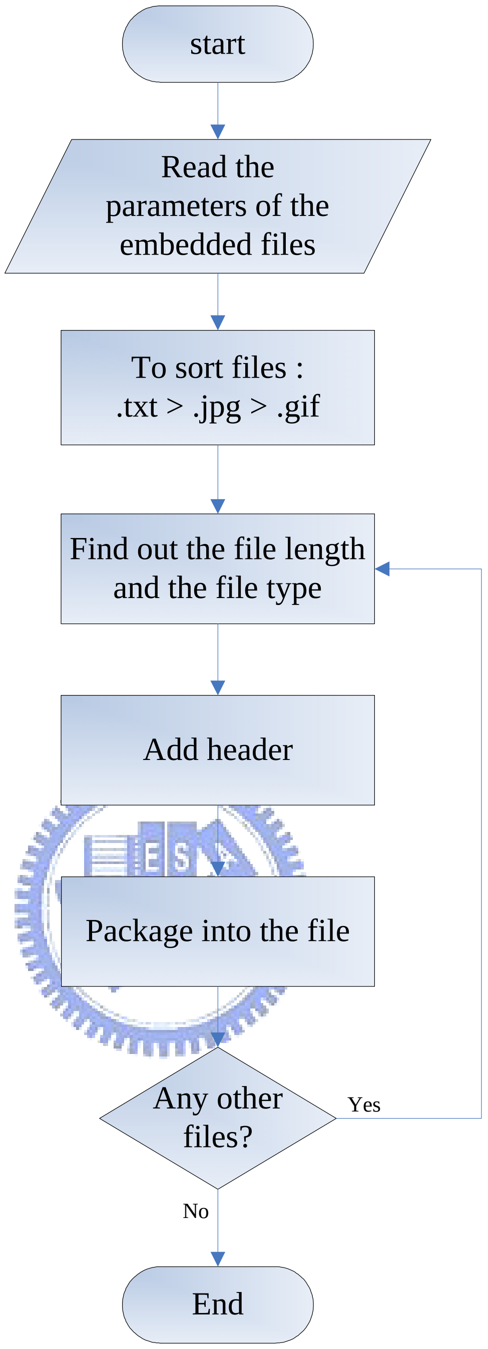

start
Read the parameters of the embedded files
To sort files : .txt > .jpg > .gif
Find out the file length and the file type
Add header
Package into the file
Any other files?
Yes
No
End

Fig. 19 The flowchart of the packaged file

4.1.2 Embed Data into the MP3 File

From the structure of the data embedded encoder, as shown in Fig. 20, the packaged file that embedded data has been packaged will be sent to MP3 encoder along with the raw PCM data to be encoded. There are two ways to embed the package file. One is embedding the data in count1 region and big-value region while processing the Huffman encoding. The other is embedding the data in the block of the bitstream formatting bit reservoir, if there is redundant bites.

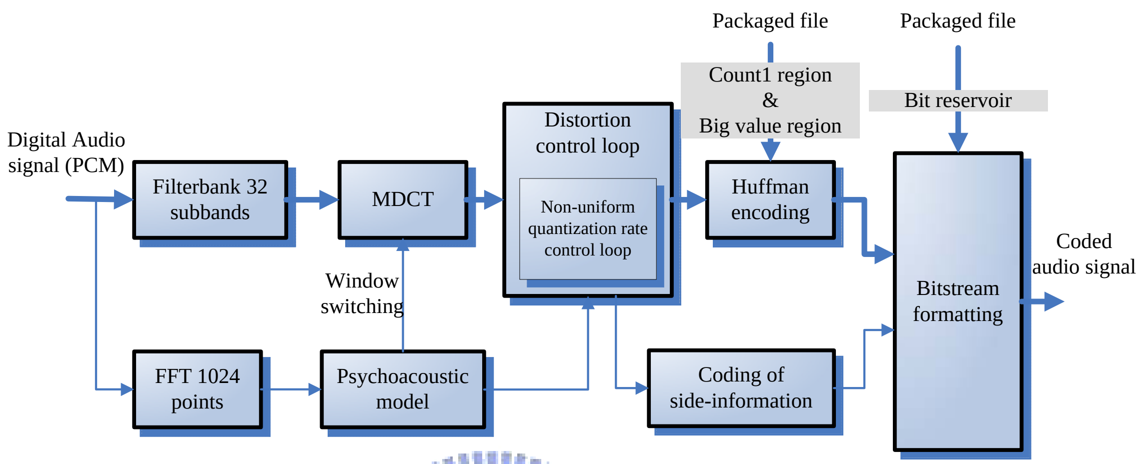


Fig. 20 The block diagram of the MP3 encoder with data embedded encoder

In count1 region, the coding process is taking four frequency lines at one time to run the Huffman encoding, then added the sign bit in back of the Huffman code. The four frequency lines are represented by v, w, x, and y. If the value of the frequency lines do not equal zero, then it will have to save the sign bit of the frequency lines. The data embedded method, using the storage space of sign bit to embed data, means to replace sign bit by the embedded data. Because in every song, after quantization the size of count1 region differs in every frame, thus the storage space for the embedded data differs. If the song has less energy in high frequency band, then the space of count1 region after quantization will be larger, that is, more space for embedded data.

Another embedded region is bit reservoir, which has a 9-bit pointer to record how much bits are redundant in the former frame. Embedding data in the redundant bits in bit reservoir would not cause any negative influence to the music. If the bit reservoir exceed 512 bytes, it would be neglected the redundant bits then wasted. Usually in the beginning of the music, the frame is quite sound without any signals so could be used for encoding, thus bit reservoir would has redundant space for embedded data. So does the ending of the music. But in frames with sound, there are not much redundant bits for embedded data in bit reservoir. Most redundant space for embedded data in bit reservoir is provided by the non-signal frames in the beginning or the ending of the music, as shown in Fig. 21.

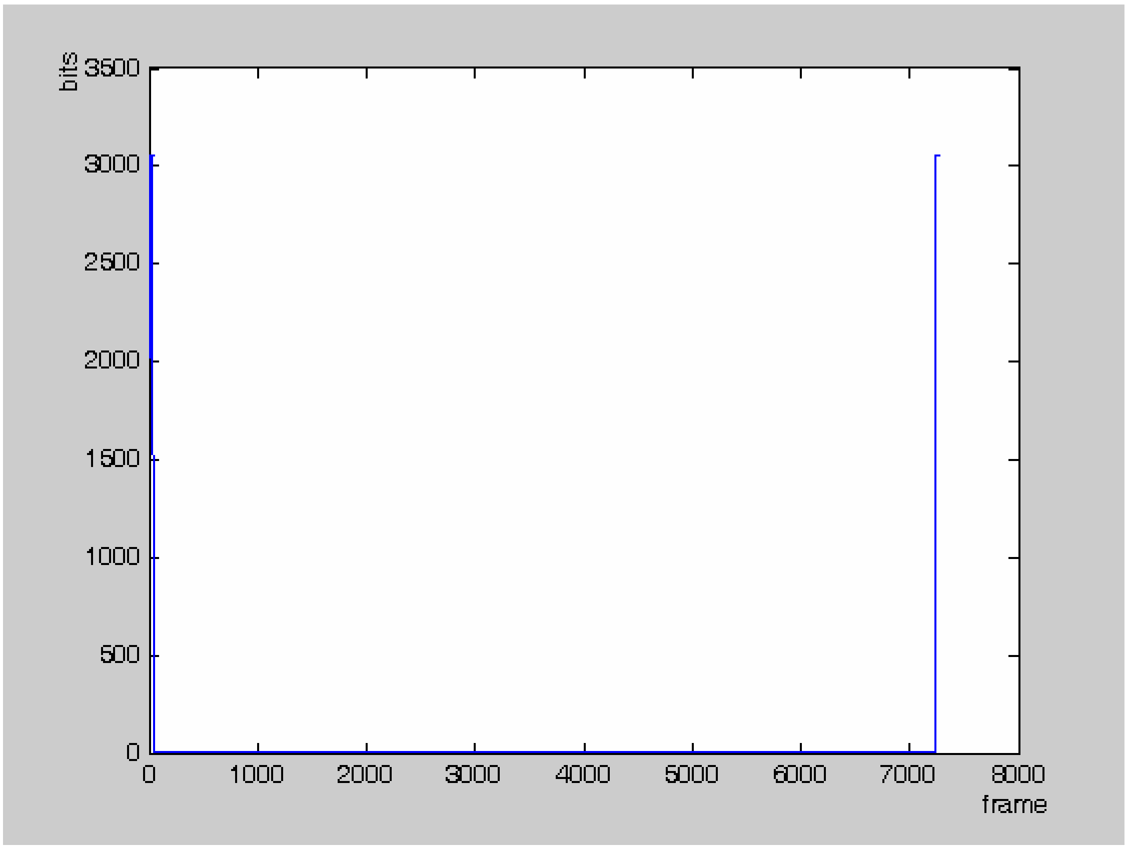


Fig. 21 The distribution of the unnecessary bits of the bit reservoir in MP3 song [18]

4.2 Data Embedded Decoder

The data embedded decoder includes four parts: extracting data from the MP3 file, porting the MP3 decoder with data embedded decoder on the ADSP-2181, the data stream analyzer, and the lyric analyzer.

4.2.1 Extract Data from the MP3 File

The data is embedded into the “bitstream coding” block at the end of the encoder. The data stream extractor is located in the “decoding of bitstream” block of the decoder, as shown in Fig. 22. When inputting MP3 data stream in for decoder bitstream decoding, there will be two outputs: 576 lines of frequency lines that decoded from Huffman decoding are prepared for inverse quantization, and extracted data from data stream extractor.

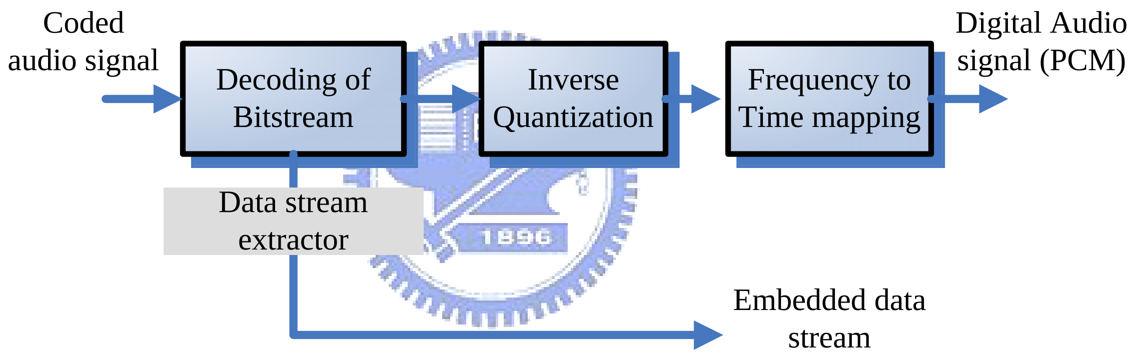


Fig. 22 The block diagram of the MP3 decoder with data embedded decoder

The data stream extractor extracts data from the count1 region and big-value region of the Huffman coding and the bit reservoir. And the data stream is collected and saved in the buffer. It will be analyzed by the data stream analyzer.

4.2.2 Porting MP3 Decoder with Data Embedded Decoder on the ADSP-2181

The structure of the MP3 decoder with the data embedded decoder ported on the

ADSP-2181 is the same as the structure on the PC. The MP3 decoder and the data embedded decoder are implemented in ADSP assembly language directly in order to have the better executing performance.

The ADSP-2181 is designed for digital signal processing. It has circular buffer function, which is used for DSP porting. This function is for write-in or when reading process is performed to the end of the buffer address, the address pointer will automatically point back to the beginning of the circular buffer, like circuit. The bit stream is decoded from the PC-based data embedded decoder and put into the data analyzer to analyze every frame. The data analyzer will analyze the extracted data and store the result in the buffer. Because ADSP-2181 belongs to the device end of the entire embedded system, it is controlled by StrongARM CPU which is as a host of the system. In ADSP-2181, could not perform data reading and analyzing every frame in the host, which would be a waste of time for the host and device to do hand-shaking constantly. At the device, the extracted data is written into a shared buffer by the extractor on the ADSP-2181 and the writing address is recorded. When the host read the data stored in the shared buffer, it will just have to identify if the data write pointer is changed, and then the reading process could be performed. As a result, the host won’t have to read the buffer in every frame and cause an influence to the CPU performance.

The MP3 decoder with data embedded decoder is realized real-time decoder which the decoding speed is 18 MIPS, 20.7K bytes of program memory, and 23.6K bytes of data memory, as shown in Table 4. It could further be developed as a portable product.

Table 4 The spec. of the MP3 decoder with data embedded decoder on the ADSP-2181

	Program memory	Data memory	The peak computing power
MP3 decoder with data embedded decoder	20.7K bytes	23.6K bytes	18 MIPS

4.2.3 Data Stream Analyzer

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

The flowchart of the data stream analyzer is shown as Fig. 23, which is based in the state machine structure. The purpose of stream analyzer is for data stream analyzing, identifying synchronization, file length, file type, and processing every different files type. The start point of the embedded must be found to perform data stream analyzing, thus synchronization bits should be identified.

"Synchronization bits" is composed of 4 bytes. So at first 4 bytes should be read to judge if its synchronization bits. If not, left-shifting one byte and replenish a byte for judging, until the synchronization bits is found to jump to the next state. On the other hand, the file length parameter shows the analyzer how many bytes in the file to read. Last, the file type parameter shows the analyzer what file type is, which would be beneficial for the analyzer to handle it properly in next state. If the file type is lyric, the file will be saved to a lyric buffer, preparing to be shown synchronously on the screen during the song is playing. Other file types will be saved as file, then finishing the file analyzing. State machine will jump to the first state and proceeds to the synchronization bits of the next file for analyzing.

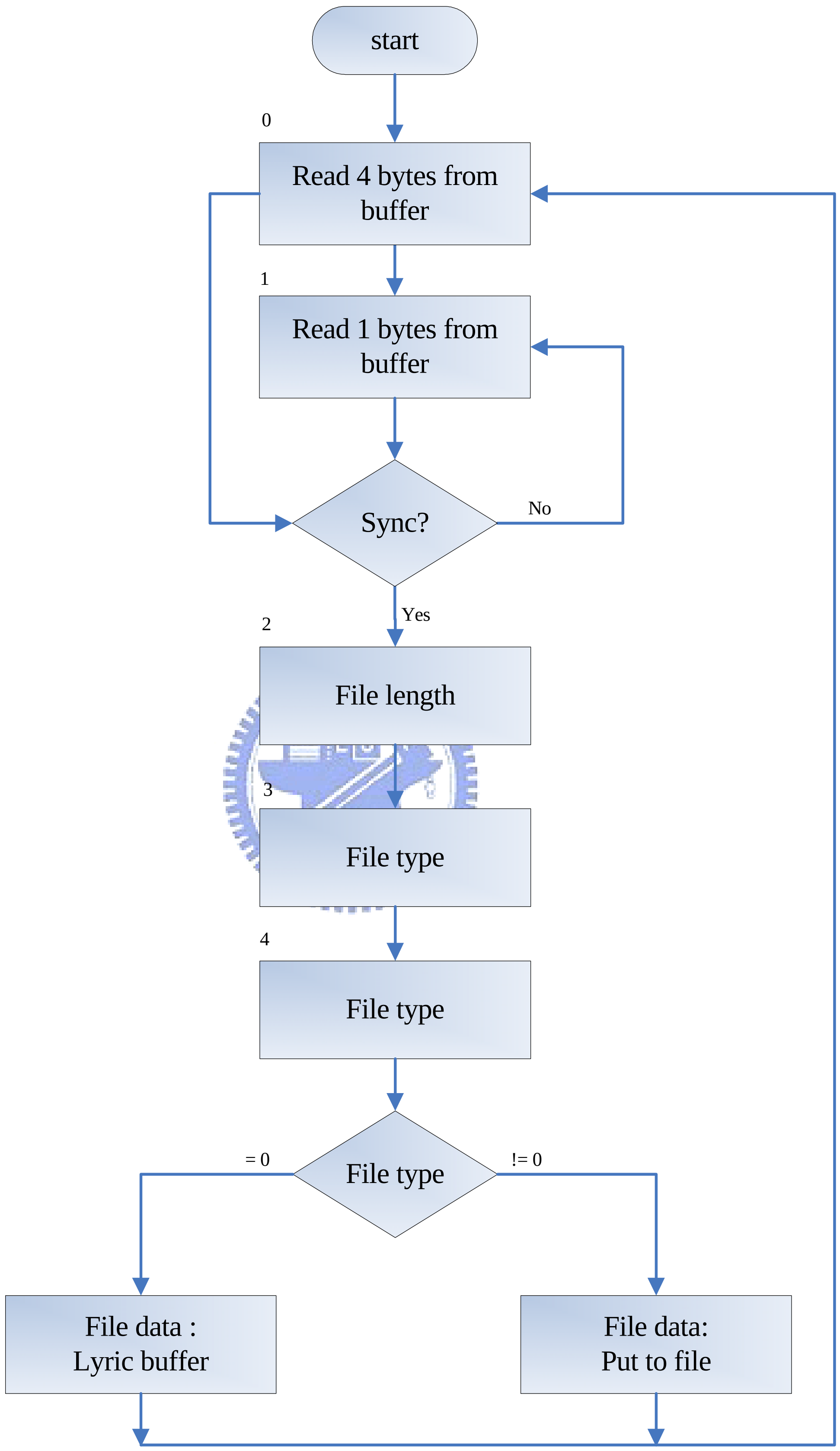


Fig. 23 The flowchart of the data embedded analyzer

4.2.4 Lyric Analyzer

The lyric analyzer is also designed in the structure of state machine. It's for computing show time of the lyrics, which should be synchronous to the playing time of the song, the same as lyrics showing in KTV. In data stream analyzer, if the data type is lyrical after analyzing, it will be saved temporarily to lyric buffer for analyzing by lyric analyzer.

The lyric format is defined as ”[mm:ss] the lyrics of a line”, shown by line as its unit. “[” represents the beginning of the lyrics in every line, ”mm” records the showing minutes of the lyrics, ”:” is for partition, and “ss” is to records the showing seconds of the lyrics. From ”]” to the changing line character ”0D 0A”, they all are the contents of a line of the lyrics.

At first the state machine will read one byte to identify if it's the beginning of one line ”[”. After finding that, it will jump to the next state to store the address of the line. Then next four states record the showing time of the lyrics. After finish reading the showing time of the lyrics would be the contents of the lyrics. The result will be saved to print buffer and then the state machine will jump to the first state for the next line of the lyric analyzing.

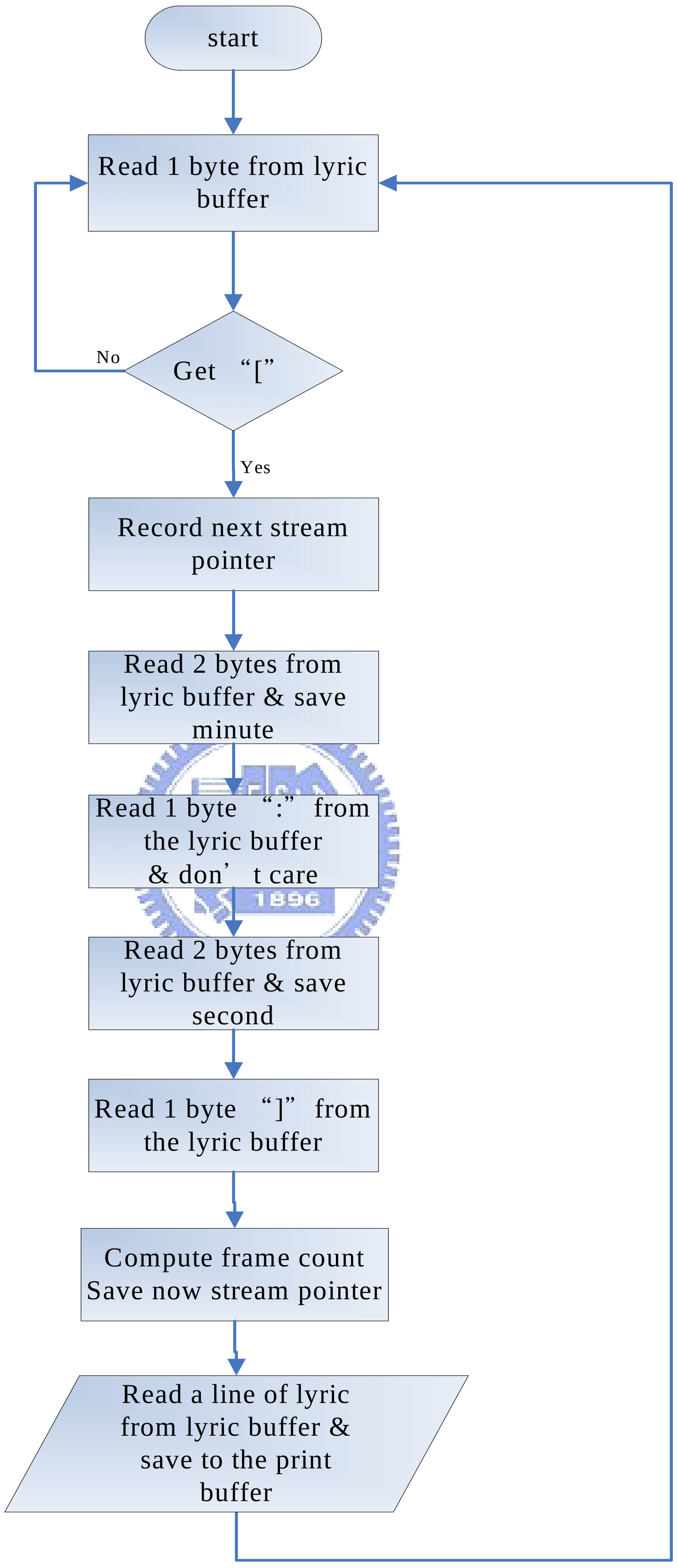

start
Read 1 byte from lyric buffer
Get "["
No
Yes
Record next stream pointer
Read 2 bytes from lyric buffer & save minute
Read 1 byte ":" from the lyric buffer & don' t care
Read 2 bytes from lyric buffer & save second
Read 1 byte "]" from the lyric buffer
Compute frame count Save now stream pointer
Read a line of lyric from lyric buffer & save to the print buffer

Fig. 24 The flowchart of the lyric analyzer

4.3 Experimental process

This section presents the overall experimental process from embedding data in the encoder to extract data in the decoder. The process has been introduced in the former section.

4.3.1 Encoding

In the encoder process, several samples are chosen for embedding data. And the files which are embedded into MP3 are embedded into the count1 region and the big-value region that the embedded region are brought out in the previously chapter. We select one from the samples to demo in the following.

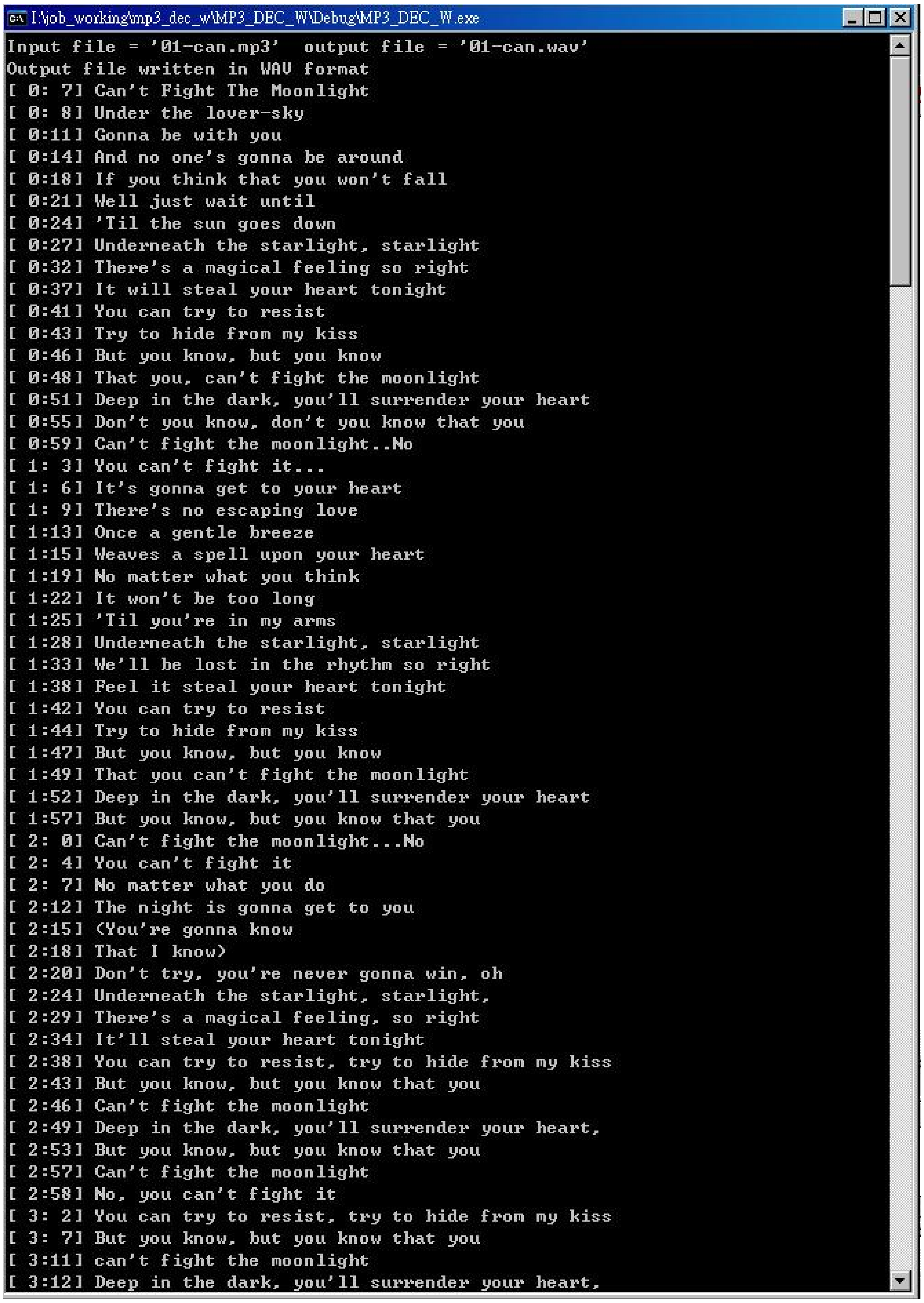
There are six MP3 samples which are chosen for embedding data in this experiment. Their MP3 names are “01-can”, “Aero Smith - Miss a Thing”, “Bon Jovi always”, “Dido thank you”, “Natalie-Torn” and “Speech”. The “01-can” is selected to demo and the information about “01-can” is shown in Table 5. The “01-can” is embedded a lyric file and six photo files. The total size of the embedded files is about 100K bytes and the files are embedded into the MP3 file that the size of the sample is 3.3 MB. To be mentioned that the embedded photos can be any resolution as long as the file size of the photo is not bigger than the maximum embedded size of the MP3 file.

Table 5 The information of “01-can”

File name		File size	Photo resolution	As shown in
Source MP3 file	01-can	3462374 Bytes		
Embedded file	Lyric	2255 Bytes		Fig. 25
	Photo1	10244 Bytes	190*190*24bit	Fig. 26
	Photo2	20462 Bytes	300*290*24bit	Fig. 27
	Photo3	18806 Bytes	300*300*24bit	Fig. 28
	Photo4	16842 Bytes	300*282*24bit	Fig. 29
	Photo5	15586 Bytes	238*184*24bit	Fig. 30
	Photo6	12070 Bytes	160*239*24bit	Fig. 31

4.3.2 Decoding

The following figures are the information which is extracted by the decoder. And the information is embedded in the MP3 file in advance. In Fig. 25, the lyric is displayed with the MP3 decoding concurrently. And Fig. 26~Fig. 31 are shown the extracting photos which are embedded in the MP3 file.



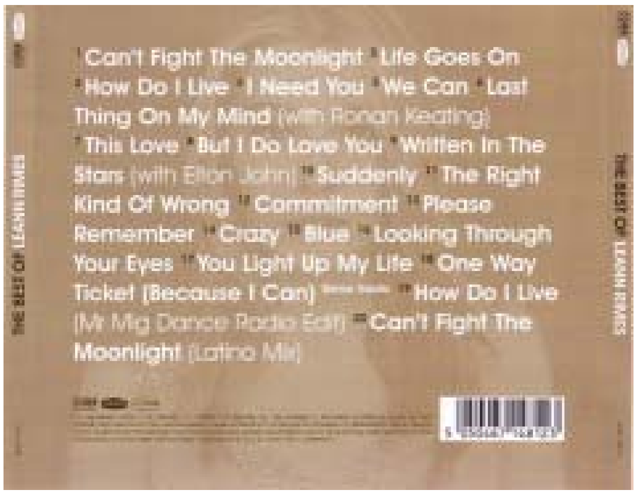
```
Input file = '01-can.mp3'  output file = '01-can.wav'
Output file written in WAV format
[ 0: 7] Can't Fight The Moonlight
[ 0: 8] Under the lover-sky
[ 0:11] Gonna be with you
[ 0:14] And no one's gonna be around
[ 0:18] If you think that you won't fall
[ 0:21] Well just wait until
[ 0:24] 'Til the sun goes down
[ 0:27] Underneath the starlight, starlight
[ 0:32] There's a magical feeling so right
[ 0:37] It will steal your heart tonight
[ 0:41] You can try to resist
[ 0:43] Try to hide from my kiss
[ 0:46] But you know, but you know
[ 0:48] That you, can't fight the moonlight
[ 0:51] Deep in the dark, you'll surrender your heart
[ 0:55] Don't you know, don't you know that you
[ 0:59] Can't fight the moonlight..No
[ 1: 3] You can't fight it...
[ 1: 6] It's gonna get to your heart
[ 1: 9] There's no escaping love
[ 1:13] Once a gentle breeze
[ 1:15] Weaves a spell upon your heart
[ 1:19] No matter what you think
[ 1:22] It won't be too long
[ 1:25] 'Til you're in my arms
[ 1:28] Underneath the starlight, starlight
[ 1:33] We'll be lost in the rhythm so right
[ 1:38] Feel it steal your heart tonight
[ 1:42] You can try to resist
[ 1:44] Try to hide from my kiss
[ 1:47] But you know, but you know
[ 1:49] That you can't fight the moonlight
[ 1:52] Deep in the dark, you'll surrender your heart
[ 1:57] But you know, but you know that you
[ 2: 0] Can't fight the moonlight...No
[ 2: 4] You can't fight it
[ 2: 7] No matter what you do
[ 2:12] The night is gonna get to you
[ 2:15] (You're gonna know
[ 2:18] That I know)
[ 2:20] Don't try, you're never gonna win, oh
[ 2:24] Underneath the starlight, starlight,
[ 2:29] There's a magical feeling, so right
[ 2:34] It'll steal your heart tonight
[ 2:38] You can try to resist, try to hide from my kiss
[ 2:43] But you know, but you know that you
[ 2:46] Can't fight the moonlight
[ 2:49] Deep in the dark, you'll surrender your heart,
[ 2:53] But you know, but you know that you
[ 2:57] Can't fight the moonlight
[ 2:58] No, you can't fight it
[ 3: 2] You can try to resist, try to hide from my kiss
[ 3: 7] But you know, but you know that you
[ 3:11] can't fight the moonlight
[ 3:12] Deep in the dark, you'll surrender your heart,
```

Fig. 25 The extracted lyrics of the MP3 song

Fig. 26 Photo1.jpg

Fig. 27 photo2.jpg

Fig. 28 Photo3.jpg



1 Can't Fight The Moonlight 2 Life Goes On
3 How Do I Live 4 I Need You 5 We Can 6 Last
Thing On My Mind (with Ronan Keating)
7 This Love 8 But I Do Love You 9 Written In The
Stars (with Elton John) 10 Suddenly 11 The Right
Kind Of Wrong 12 Commitment 13 Please
Remember 14 Crazy 15 Blue 16 Looking Through
Your Eyes 17 You Light Up My Life 18 One Way
Ticket (Because I Can) 19 How Do I Live
(Mr Mig Dance Radio Edit) 20 Can't Fight The
Moonlight (Latino Mix)

THE BEST OF LEANN RIMES

THE BEST OF LEANN RIMES

Fig. 29 Photo4.jpg

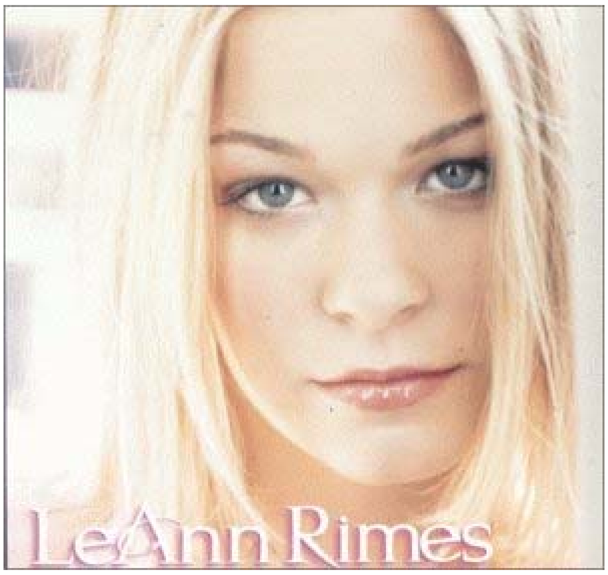


Fig. 30 Photo5.jpg

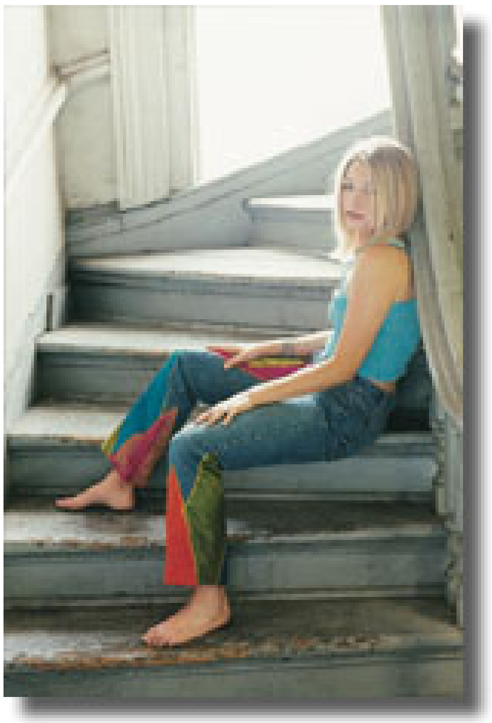


Fig. 31 Photo6.jpg

4.4 Experimental Results

In this section, there are several experiments for testing different methods. These experiments are “The embedded bit counts of the different methods”, “The encoding speed of the different methods”, “The music quality of the different methods”, and “The file size of the different methods”.

The statistic data and description will be performed in the following:

4.4.1 The Embedded Bits Counts of the Different Methods

The purpose of this experiment is to estimate the embedding size of the three different regions of the MP3 frame, as shown in Table 6. The count1 region embeds largest data size, on the other hand, embedding size differs in every music in bit reservoir and the big-value region, as shown in Table 6.

Generally, the frequency of the music signals distribute under the frequency of 10 KHz mostly. The energy of the frequency signal which is above 10KHz is usually small and the signal will be coded into coutn1 region by encoder, or the high frequency signal without energy will be coded into zero region. So the space of the count1 region used for embedding is larger than other region. The big-value region is the region that the energy of the music collected. If the embedded data replaces too many sign bits of the big-value region, the quality of the music will be affected seriously. In order to maintain quality of the music, we select the sign bits of the frequency signals above 10 KHz for embedding data. And the unused space in the bit reservoir is provided by a little span in the beginning and the ending of the music. There are very few unused bits of the bit reservoir for embedding during the music playing.

Table 6 The embedded bits count of the different methods

Song name	Frame num.	Count1 region (bits)	Big-value region (>11.84K Hz)(bits)	Bit reservoir (bits)
01-can	8285	595187	299805	313806
Aero Smith - Miss a Thing	11374	811605	83007	84605
Bon Jovi - Always	13578	927084	168541	240744
Dido- Thank you	8647	622612	192774	190202
Natalie-Torn	7352	514852	406254	406205
Speech	1944	132152	0	6789

Table 7 shows the number of bits per frame obtained by the three different methods. Count1 region has 70 bits per frame for embedding data in average. The bit numbers of the other two methods depend on the different types of music. If more bits will be embedded into big-value region, the data could be embedded from the starting frequency lower than 11.84 kHz. Then, the loss of quality is the cost to perform embedding more data.

Table 7 The embedded bits count per frame by the different methods

Song name	Count1 region (bits/frame)	Big-value region (>11.84K Hz) (bits/frame)	Bit reservoir (bits/frame)
01-can	71.8	36.2	37.9
Aero Smith - Miss a Thing	71.4	7.3	7.4
Bon Jovi - Always	68.3	12.41	17.7
Dido-Thank you	72.0	22.3	22
Natalie-Torn	70.1	55.3	55.3
Speech	68.0	0	3.5

4.4.2 The Encoding Speed of the Different Methods

The purpose of this experiment is to test the encoding speed of the MP3 encoder when the data is embedded into different regions. The specification of the testing platform is shown in Table 8. In Table 9, the 1X stands for the length of the song.

Table 8 The spec. of the test platform

CPU	AMD XP1700
RAM	768MB

The encoding speed of the MP3 encoder reduces, as shown in Table 9, after adding data embedded encoder into the MP3 encoder. But the main reason is that the MP3 encoder is a floating point encoder, thus the encoding speed is limited by the MP3 encoder.

Table 9 The encoding speed of the different methods by floating point encoder

Song name	Song time	None	Count1 region	Big-value region	Bit reservoir
01-can	216s	1.95X	1.83X	1.90X	1.85X
Aero Smith - Miss a Thing	297s	2.18X	2.17X	2.18X	2.17X
Bon Jovi Always	354s	2.20X	2.20X	2.14X	2.19X
Dido Thank you	225s	2.09X	2.09X	2.07X	2.09X
Natalie-Torn	192s	1.79X	1.71X	1.70X	1.75X
Speech	50s	2.42X	2.42X	2.42X	2.42X

If the block of the psychoacoustic model of the MP3 encoder is removed and the MP3 encoder is implemented in fixed-point arithmetic, then the MP3 encoder will be sped up more than 25X as shown in Table 10. The influence on speed when adding the data embedded encoder into the MP3 encoder becomes relatively small, barely none.

Table 10 The encoding speed of the different methods by fixed point encoder

Song name	Song time	None	Count1 region	Big-value region	Bit reservoir
01-can	216s	30.1X	30.1X	30.1X	30.1X
Aero Smith - Miss a Thing	297s	29.7X	29.7X	29.7X	29.7X
Bon Jovi Always	354s	29.5X	29.5X	29.5X	29.5X
Dido Thank you	225s	28.1X	28.1X	28.1X	28.1X
Natalie-Torn	192s	32X	32X	32X	32X
Speech	50s	25X	25X	25X	25X

4.4.3 The Music Quality of the Different Methods

The purpose of this experiment is to test the quality of the music after the data embedded into the MP3 file. The ODG (Objective Difference Grade) is a standard used in testing quality of the music in the experiment and it has a brief introduction in Appendix B. The ODG is software for testing the quality of the music, which is used to compare the original music and the encoded music. The other constantly testing condition is shown in Table 11.

Table 11 The testing environment of the music quality

Test program	ODG
Test standard	0~-4 (good to worse)
The MP3 decoder	cooledit
Sampling rate of MP3	44.1KHz
Bit rate of MP3	128kbps
Sound effect	stereo

The quality of the MP3 file is influenced after the data is embedded into the MP3 file, as shown in Table 12. The influence is about 2%, and the distortion of the MP3 would not be heard by human ears, which means the quality of the music can be accepted. In general, the bit reservoir will not be distorted, but the data has distortion in the table in fact. The distortion is human distortion. The reason is that the music will be shifted after MP3 encoding and decoding, so the music must be shifted back by manual. And then the human distortion is made. In general, the three methods which are used to embed data make a little influence to the quality of the music.

Table 12 The music quality of the different methods

Song name	None	Count1 region	Big-value region	Bit reservoir
01-can	-1.28	-1.33	-1.33	-1.28
Aero Smith - Miss a Thing	-0.96	-1.04	-0.97	-0.97
Bon Jovi Always	-1.10	-1.20	-1.12	-1.11
Dido Thank you	-1.18	-1.22	-1.20	-1.18
Natalie-Torn	-1.35	-1.37	-1.37	-1.35
Speech	-1.80	-1.83	-1.81	-1.81

4.4.4 The File Size of the Different Methods

The purpose of this experiment is to observe what change to the size of the MP3 file will have after embedding data into the different regions.

The size of the MP3 file which is embedded data by the three data embedded methods is the same with the size of the MP3 file without embedding data, as shown in Table 13. So the embedded data methods that are brought up in this thesis will not change the file size that the MP3 is encoded by general MP3 encoder.

Table 13 The file size of the different methods

Song name	None (MB)	Count1 region (MB)	Big-value region (MB)	Bit reservoir (MB)	Total embedded bits (KB)
01-can	3.30	3.30	3.30	3.30	147.6
Aero Smith - Miss a Thing	4.53	4.53	4.53	4.53	119.5
Bon Jovi Always	5.41	5.41	5.41	5.41	163.1
Dido Thank you	3.44	3.44	3.44	3.44	122.8
Natalie-Torn	2.92	2.92	2.92	2.92	162.0
Speech	0.79	0.79	0.79	0.79	17.0K

4.4.5 Comparison with other methods

There are many watermark techniques about data embedded methods on the IEEE journals, but the applications of the watermark techniques are almost different from this thesis.

The watermark techniques are used for embedding important information into the media because they can resist any attack such as lossy compressing. However, the size of the embedded media is very small. The purposes of the watermark techniques are differ from the proposed methods. The applications of the proposed methods are to embed large unimportant information into media in this thesis. The watermark techniques are based on different application from ours, so they can’t compare to each other.

SecureKit, Inc. published a shareware called “Steganography 1.61.23” [19]. It can embed unlimited data into the tail of a file and encrypt the embedded data to prevent other user accessing the data, but the size of the file which is used for embedding data will be added. This is a series defect, so the method used in this software is not appropriate.

CHAPTER 5
Conclusions and Future Works

5.1 Conclusions

This thesis introduces three high capacities reversible data embedded methods. As data embedded methods generate extra storage space, the embedding area on MP3 file is not very important. The data embedded methods are designed under the MPEG/Audio standard. Data embedded decoder is ported on the ADSP-2181. The ADSP-2181 is a programmable microprocessor and offers a software development environment tool: VisualDSP++, which we can develop and debug the algorithm quickly and easily in the programming stage.

These three methods bringing up for embedding data into the MP3 files are count1 region, bit reservoir, and big-value region, which is an expansion of the count1 region method. Then, the suggested sequences of the three embedding methods are the bit reservoir, count1 region and big-value region. It depends on the degree that different methods make the different influences of the music quality. The main principle of selecting regions for embedding is not affecting quality of the music. The size used for embedding differs in every song. According to the experimenting results, the size used for embedding data is about 100K bytes in a song of which the length is 3 minutes.

In ADSP-2181 porting, the MP3 decoder with data embedded decoder is coded by assembly language achieves to increase processing speed and save memory size. Finally, the MP3 decoder with data embedded decoder realizes real-time decoding with a speed of 18 MIPS which uses 20.7K bytes of program memory and 23.6K bytes of data memory. The data embedded decoder displays the embedded lyric of the song and extract other embedded data while the MP3 player is playing.

5.2 Future Works

The data embedded codec is based on the MPEG/Audio standard and usually works with an audio codec. These methods can also be used on other advanced MPEG/Audio standard, such as MPEG-2/Audio AAC or MPEG-4/Audio AAC standard. It will become AAC codec with data embedded codec.

The MP3 with embedded data is not protected by any DRM mechanism. The data embedded methods can be combined with a secure codec, such as SMP3 designed by CSSP laboratory. It will increase the degree of safety. MP3 codec, combined with data embedded codec and SMP3 codec can make the audio codec safer and provide more service to users.

References

[1] G. Voyatzis and I. Pitas, "The use of watermarks in the protection of digital multimedia products", *Proceedings of the IEEE*, Vol.87, NO.7, pp.1192-1207, July 1999

[2] ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 11172-3 coding of moving pictures and associated audio for digital storage media at up to about 1.5Mb/s, Part 3: Audio

[3] http://www.cmj.com/mp3/mp3basic.php#what

[4] http://www.chiariglione.org/mpeg/standards.htm

[5] K. Brandenburg and H. Popp, "An introduction to MPEG Layer-3", *Fraunhofer Institutfur Integrierte Schaltungen(IIS),* EBU TECHNICAL REVIEW, Jun. 200

[6] J. Princen and A. Bradley, "Analysis/synthesis filterbank design based on time domain aliasing cancellation", *IEEE Trans. on Acoust. Speech, and Signal Processing.* Vol. ASSP-34, pp.1153-1161, 1986.

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

[8] http://www.iis.fhg.de/amm/techinf/water/index.html

[9] S.–J. Lee and S.–H. Jung, "A survey of watermarking techniques applied to multimedia", *Proceedings 2001 IEEE International Symposium on Industrial Electronics (ISIE2001*), Vol. 1, pp. 272-277, 2001.

[10] R. Lancini, F. Mapelli and S. Tubaro, "Embedding indexing information in audio signal using watermarking technique", *Proceedings of VIPromCom*, 16-19 June 2002.

[11] M. Ikeda, K. Takeda and F. Itakura, "Audio data hiding by using of band-limited random sequences", *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vlo.4, pp.2315~2318, 1999.

[12] L. Boney, A. H. Tewfik and K. N. Hamdy, “Digital watermark for audio signals”, *IEEE Proceedings of Multimedia*, pp. 473-480, 1996.

[13] J. F. Tilki and A. A. Beex, “Encoding a hidden auxiliary channel onto a digital audio signal using psychoacoustic masking”, *Southeastcon ’97. Engineering new Century., Proceeding. IEEE*, pp.331~333, 1997.

[14] E. Zwicker and H. Fastl, “*Psychoacoustics facts and models*”, Berlin, Germany: Springer-Verlag, 1990.

[15] Analog Devices, Inc., “ADSP-2100 Family EZ-KIT Lite Reference Manual”, First Edition.

[16] Analog Devices, Inc., “ADSP-2100 Family User’s Manual”, Third Edition, Sep. 1995.

[17] Analog Devices, Inc., “VisualDSP++ 3.0 Getting Started Guide for ADSP-21xx DSPs”, Revision 2.0, July 2002

[18] Leann Rimes “Can’t fight the moon light”, The Best of Leann Rimes, spring, 2004.

[19] http://www.soft32.com/download_11169.html

[20] T. Thiede et al, "PEAQ - The ITU standard for objective measurement of perceived audio quality", *J. Audio Eng. Soc.*, vol. 48, pp. 3-29, Jan.-Feb. 2000.

[21] B. C. J. Moore, “*An introduction to the psychology of hearing*”, Academic Press, London, 1997.

[22] D. E. Rumelhart, G. E. Hinton and R. J. Williams “Learning internal representations by error propagation”, *In Parallel Distributed Processing*, vol. 1, pp. 318-362. Cambridge, MA, MIT Press. 1986.

APPENDIX A
MP3 Encoder/Decoder Algorithm

Because the main purpose of our data embedded method is to embed a large of qualify information into MP3 file directly, so we describe the basic principles and algorithms of encoder and decoder in the MPEG-1/Audio Layer 3 (MP3) coding standard. The most important reason why MPEG-1/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 audio compression standard, like AAC.

Section A.1 will introduce the MPEG-1/Audio Layer 3 encoding standard and its algorithm. Section A.2 will explain the decoding process of MP3.

A.1 The Structure of MP3 Encoder Algorithm

In this section, the MPEG-1/Audio Layer III encoder will be described with its functionality. The description of the encoding process is based on the block diagram in Fig. 32. In the following subsections, we will describe the operation and the functionality in detail for each block in the block diagram.

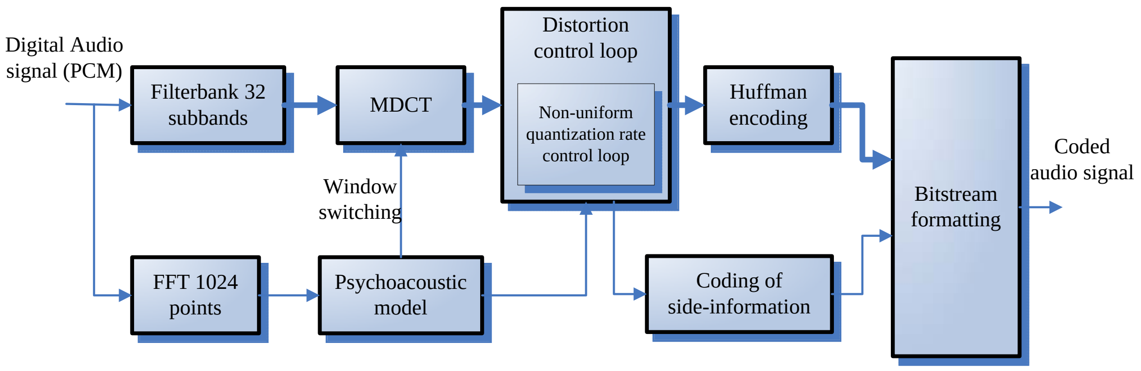


Fig. 32 MPEG-1/Audio Layer 3 encoder block diagram [5]

A.1.1 Analysis Polyphase Filter Bank

The subband filter band includes both analysis subband filter and Modified Discrete Cosine Transform (MDCT) [6] representation such as Fig. 33.

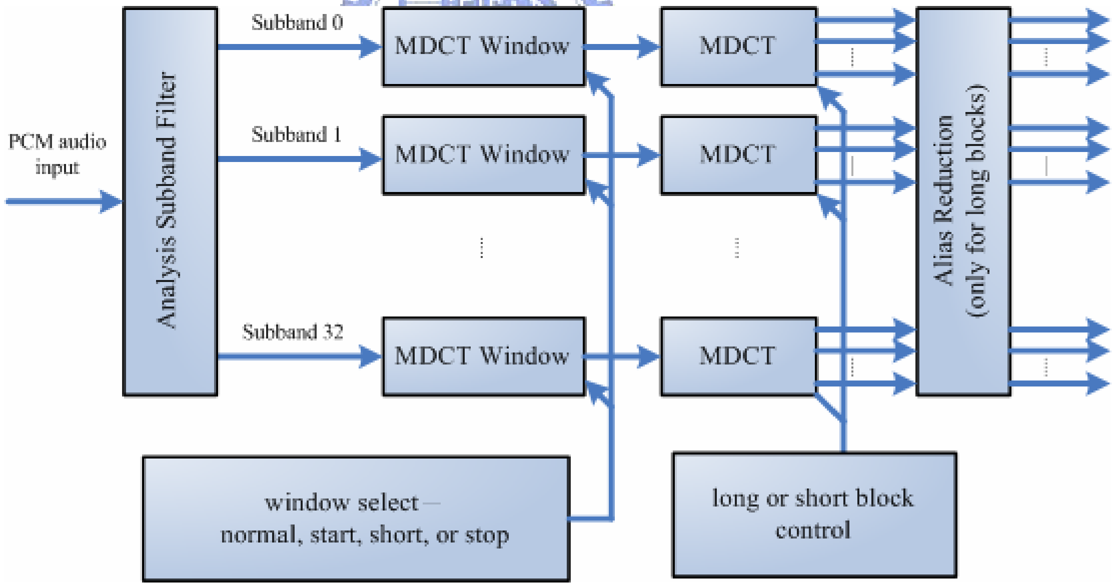


Fig. 33 Analysis subband filter and MDCT

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 PCM 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.

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 band outputs:

$$St[i] = \sum_{k=0}^{63} \sum_{j=0}^{7} M[i][k] \times (C[k+64j] \times x[k+64j]) \ , \qquad \text{Eq. 2}$$

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

$M[i][k] = \cos[\frac{(2 \times i + 1) \times (k - 16) \times \pi}{64}]$ is the analysis matrix coefficients

The coefficients of C[n] in EQ 1 are symmetric to origin, as shown in Fig. 34 (a). Eq. 2 manipulate into a intelligible filter convolution Eq. 3 for more convenient to analysis.

$$St[i] = \sum_{n=0}^{511} x[t-n] \times Hi[n] \ , \qquad \text{Eq. 3}$$

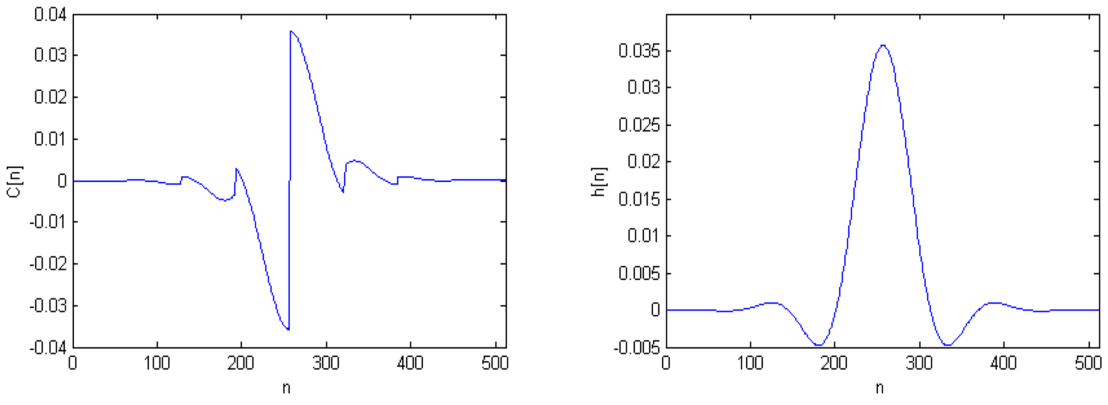
Where:

$x[\tau]$ is an audio sample at time τ

$$Hi[n] = h[n] \times \cos\left[\frac{(2\times i+1)\times(n-16)\times\pi}{64}\right]$$

with h[n]=-C[n], if the integer part of (n/64) is odd,

C[n], otherwise, for n=0 to 511



(a) C[n] (b) h[n]

Fig. 34 coefficient of C[n] and h[n] (n = 0~511)

The modulation of the prototype low-pass filter (h[n]) with a cosine term (M[i][k]) result 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. The frequency response of 32 subband, as shown in Fig. 35.

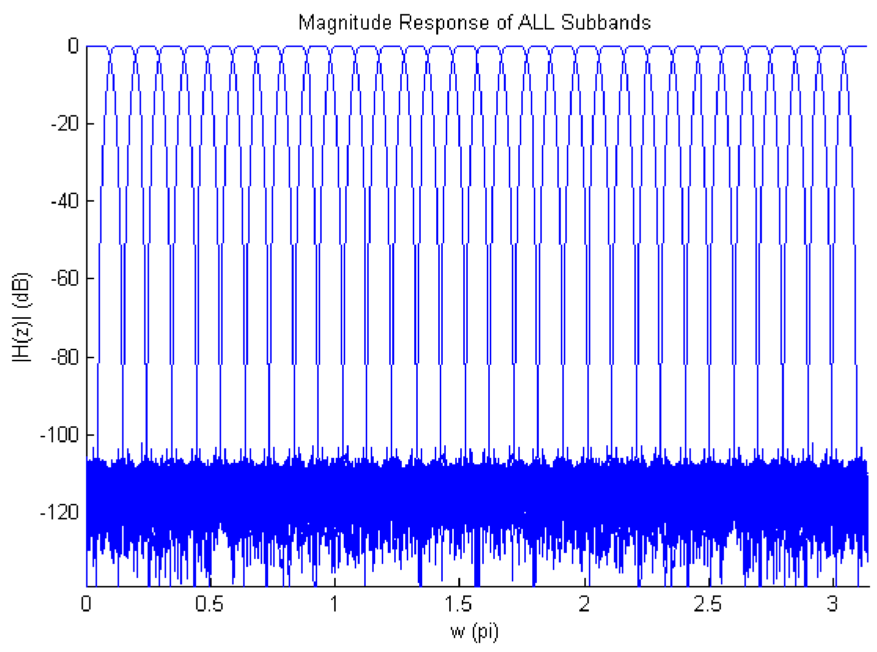


Fig. 35 frequency response of subband

A.1.2 MDCT and Alias Reduction

After the subband filter, the 32 subbands are mapped into MDCT. Performing this transformation will enhance the frequency resolution per subband. MDCT transformation can be divided into three parts: MDCT windows, MDCT, alias reduction, as shown in Fig. 33.

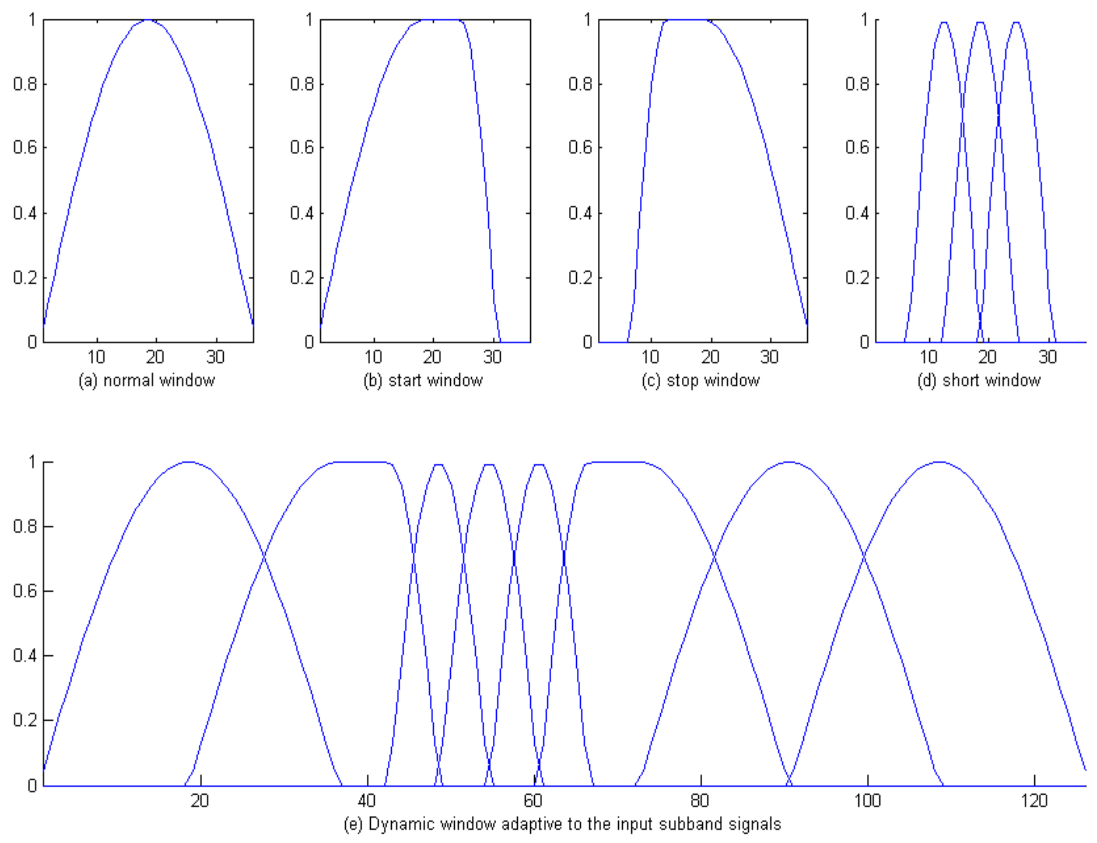


Fig. 36 Illustration of the four applicable window types and using condition

The MDCT windows should be differentiated into normal window, start window, short window, stop window, as Fig. 36 Illustration of the four applicable window types and using condition shown. The transformation of normal window can get better resolution of frequency spectrum. But the transformation of short window can get better resolution of time response. The switching mechanism which means the start window and the stop window helps to prevent the appearance of pre-echo phenomenon. Eq. 4 shows the formula for MDCT transformation.

$$Xi = \sum_{k=0}^{n-1} Z_k \cos\left(\frac{\pi}{2n}(2k+1+\frac{n}{2})(2i+1)\right), \text{ for i = 0} \sim (\frac{n}{2}-1) \qquad \text{Eq. 4}$$

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 Fig. 37. The cs_i and ca_i constants are tabulated in standard [2]. The butterfly operations with appropriate weighting cancel the alias caused by the overlap of two adjacent overlapped subbands.

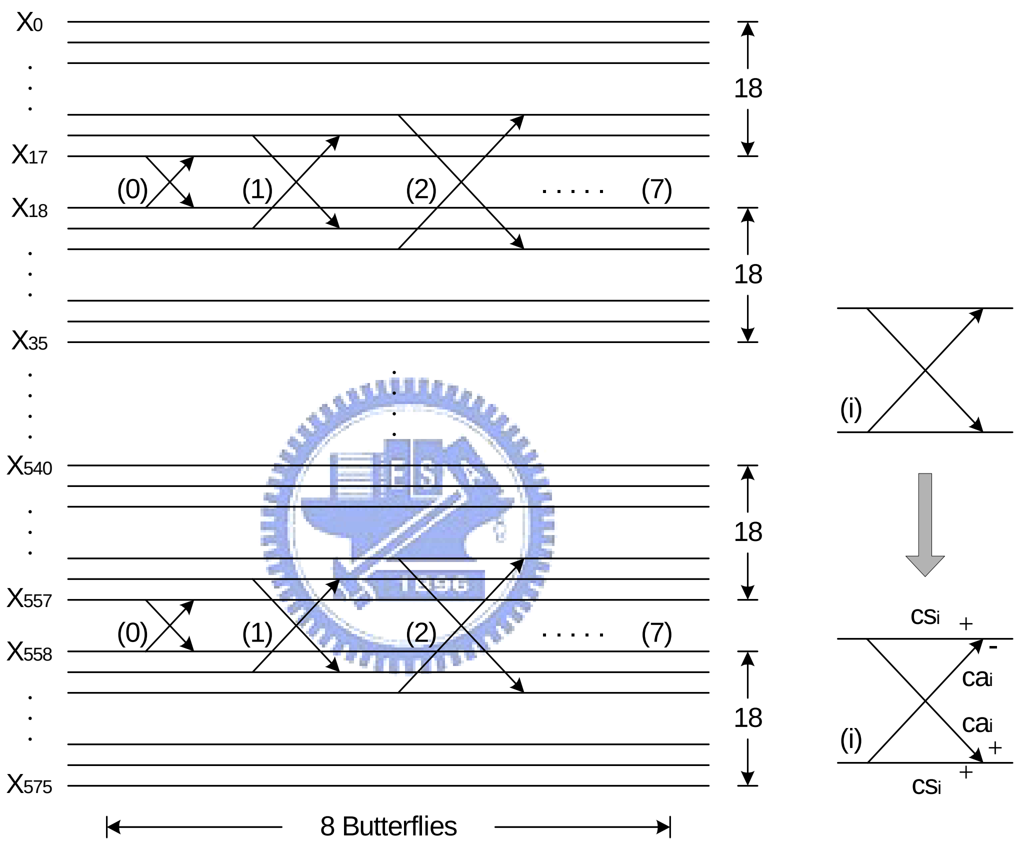


Fig. 37 Illustration of alias reduction butterflies

A.1.3 Psychoacoustic Model

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 Fig. 38

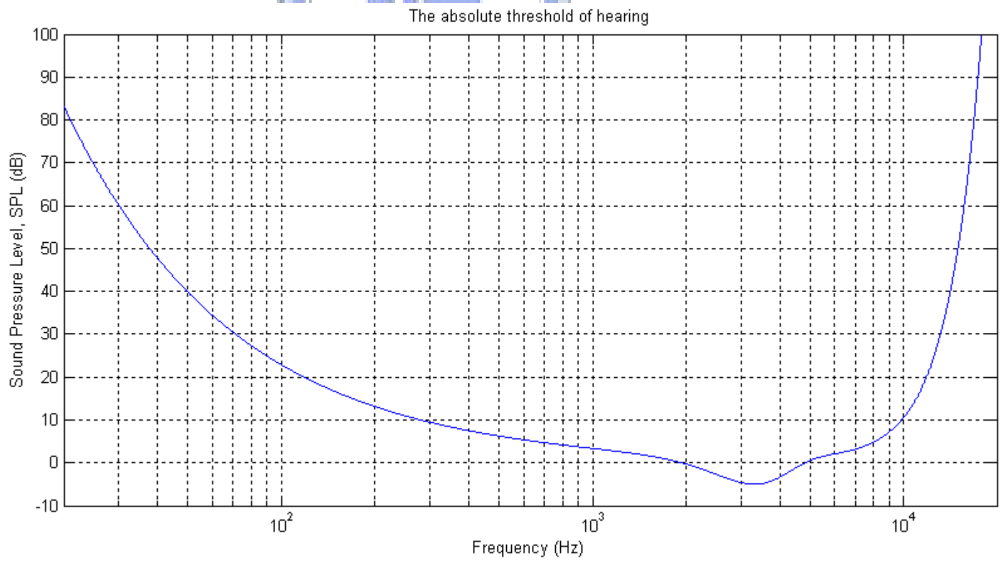


Fig. 38 The absolute threshold of hearing

- **Frequency Masking**

  Frequency masking, also called simultaneous masking, is a frequency domain phenomenon where a low-level signal (the maskee) can be made in audible by a simultaneously occurring stronger signal (the masker) as long as masker and maskee are close enough to each other in frequency. The masking threshold depends on the sound pressure level and the frequency of the masker, such as Fig. 39. Without any masker, a signal is also inaudible if its sound pressure level is below the absolute threshold.

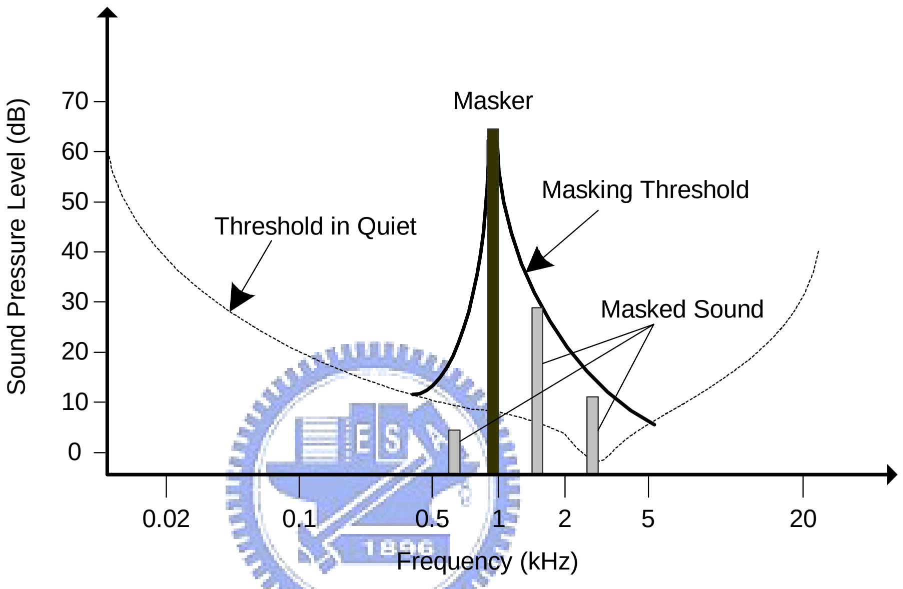


Fig. 39 Frequency masking threshold and threshold in quiet [7]

- **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 man 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, such as Fig. 40.

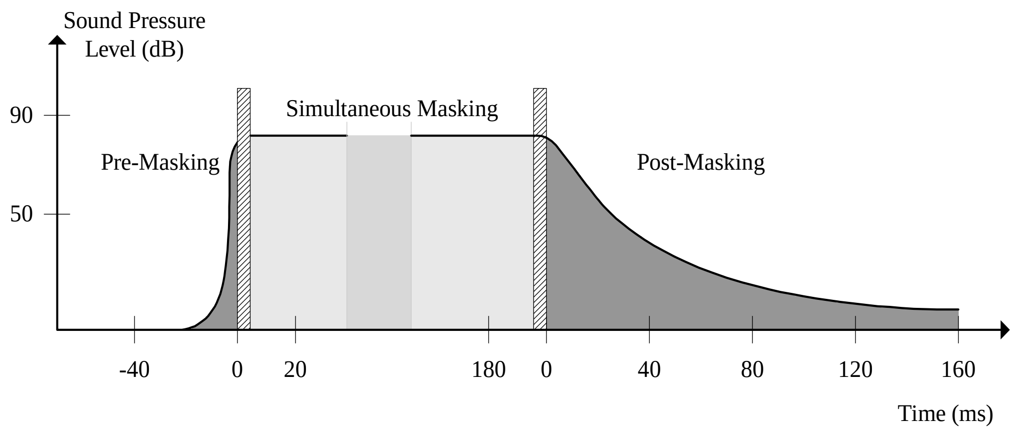


Fig. 40 Temporal masking threshold [7]

A.1.4 Nonuniform Quantization

The nonuniform quantization block which received the frequency line from the MDCT block and window switching, masking information from the psychoacoustic model, performs the important key techniques "quantization" and "Huffman coding". Quantization is done via a power-law quantizer. In this way, larger values are automatically coded with less accuracy, and some noise shaping is already built into the quantization process. The process to find the optimum gain and scalefactors for a given block, bitrate and output from the perceptual model is usually done by two nested iteration loops in an analysis-by-synthesis way:

- **Outer iteration loop (noise 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. To shape the quantization noise according to the masking threshold, scalefactors are applied to each scalefactor band. 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.

- **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 operation 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 until the resulting number of bits demand for Huffman coding is small enough.

A.1.5 Huffman Encoding

In this block, 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. Fig. 41 shows the relation of three Huffman coded regions: “zero region”, “count1 region”, and “big-value region”, and scalefactor.

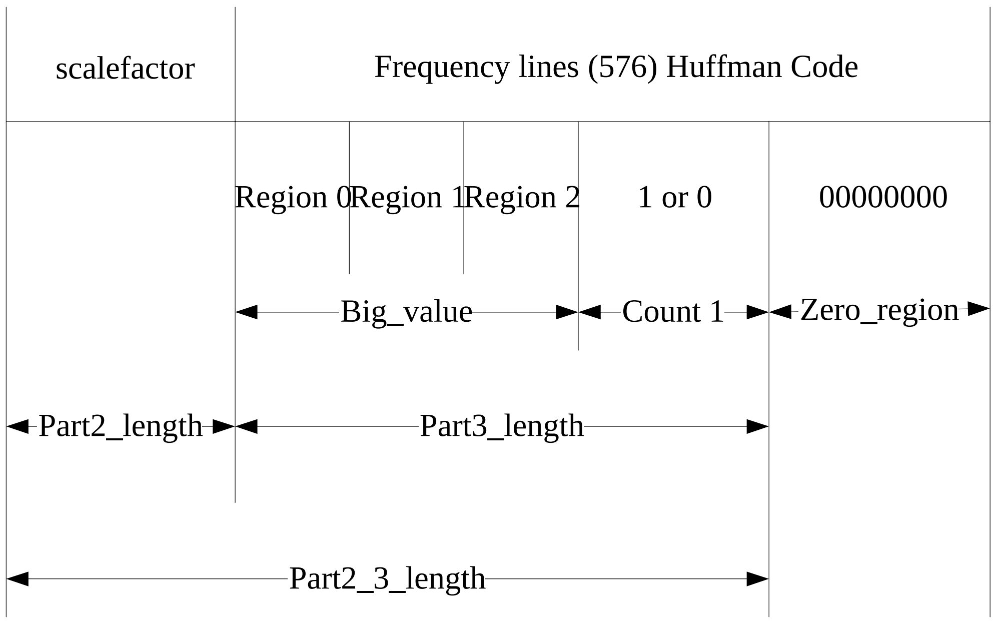


Fig. 41 Main data organization of a frame

A.1.6 Bitstream Formatting

In this block, the encoding process is to produce a MPEG-1/Audio III compliant bitstream. The Huffman coded frequency lines, the side information and a frame header are assembled to form the bitstream. 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.

The last block, an enhancement method called “bit reservoir” is used to fit 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.

A.2 The Structure of MP3 Decoder Algorithm

In this section the MPEG-1/Audio Layer III decoder will be described with its functionality. The decoding process is based on the block diagram in Fig. 42. The decoder has three main parts: “Decoding of Bitstream”, “Inverse Quantization”, and “Frequency to Time mapping”.

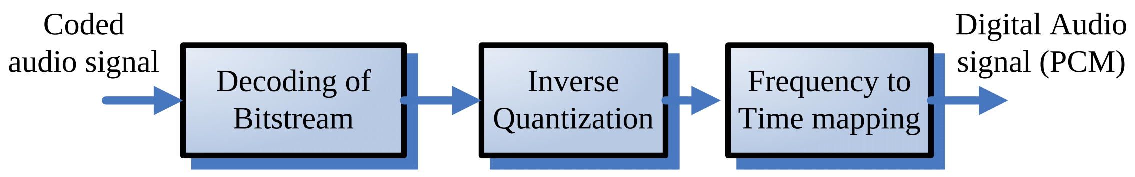


Fig. 42 MPEG-1/Audio Layer III decoder block diagram

A.2.1 Decoding of Bitstream

This decoding part effects synchronize and extract the quantized frequency lines and other information of each frame. First, it needs to synchronize where a frame begins and where the data resides, as shown in Fig. 43. The usage of the block diagram will be introduced as following.

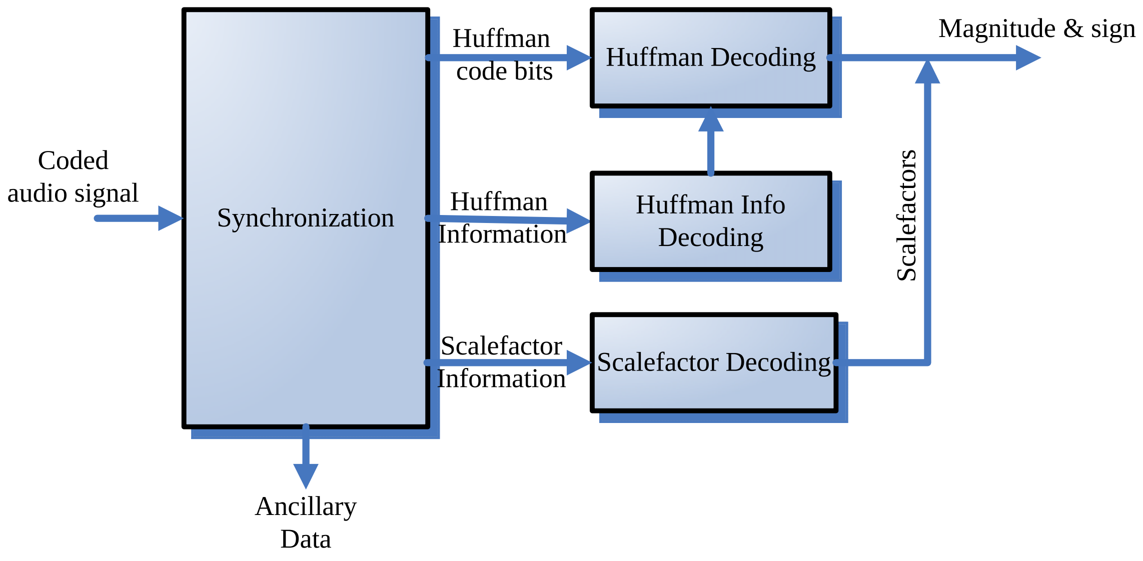


Fig. 43 Decoding of bitstream block diagram

- **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.

  The contents of a MPEG-1/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. The header part of the frame contains synchronization word and system information, as shown in Fig. 44. The side information section in the frame contains the necessary information to decode the main data. The main data section contains the coded scalefactor value and the Huffman coded data. The format of this ancillary data is user defined.

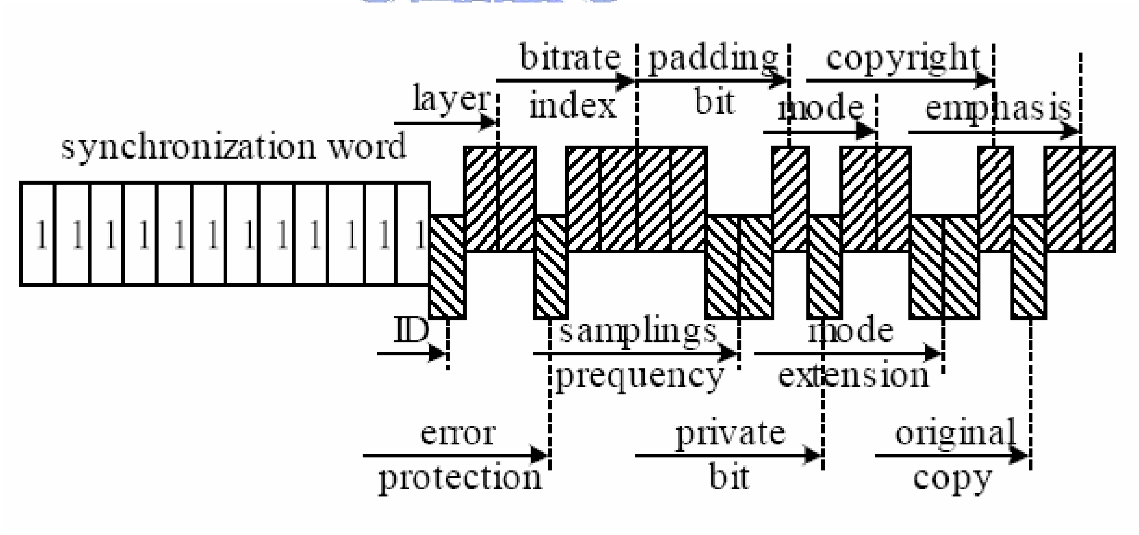


Fig. 44 MPEG-1/Audio Layer III header format

- **Huffman Decoding**

  In this block, the decoding of the Huffman code bits is performed. Since the Huffman coding is a variable-length coding, a single code word in the middle of the Huffman code bits can’t be identified without starting to decode from a point in the Huffman code bits known to be the start of a code word.

- **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. It performs to collect data in the side information which describes the characteristics of the Huffman code bits.

- **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.

A.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 formula of the frequency lines is shown in Eq. 5.

$$xr[i] = sign(is[i]) \times abs(is[i])^{\frac{4}{3}} \times \frac{2^{\frac{1}{4}(global\_gain - 210 - 8sbg[i])}}{2^{scalefac\_multiplier \times (sf[i] + perflag \times pt[i])}} \qquad \text{Eq. 5}$$

Where:

is[i] is the frequency line reconstructed by Huffman decoder

global_gain, sbg[i], scalfac_multiplier, sf[i], perflag, pt[i] are from scalefactor decoding.

A.2.3 Frequency to Time Mapping

The last decoding part performs to reproduce the audio signal from the dequantized frequency line. This part contains several sub-blocks as shown in Fig. 45 and will be introduce in the following.

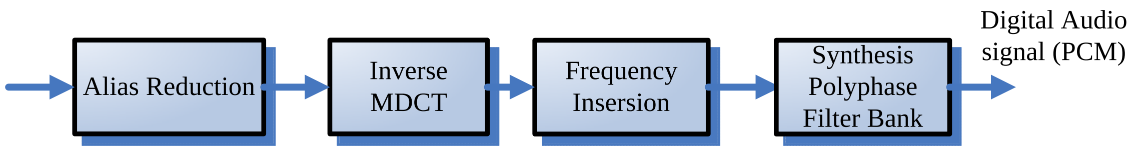


Fig. 45 Frequency to time mapping

- **Alias Reduction**

  In the MDCT block within the encoder it was described that an alias reduction was applied. In order to obtain a correct reconstruction of the analysis polyphase filter bank in the algorithm 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 Eq. 6.

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos\left(\frac{\pi}{2n}\left(2i+1+\frac{n}{2}\right)(2k+1)\right), \text{ for i = 0 ~ (n-1)} \qquad \text{Eq. 6}$$

  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.

APPENDIX B
Introduction to the Testing Standard of the Music Quality : ODG

B.1 Introduction

The ODG (Objective Difference Grade) [20] is the result which is computed by PEAQ (perceptual evaluation of audio quality). The PEAQ is based on generally accepted psychoacoustic principles (such as [14] [21]).

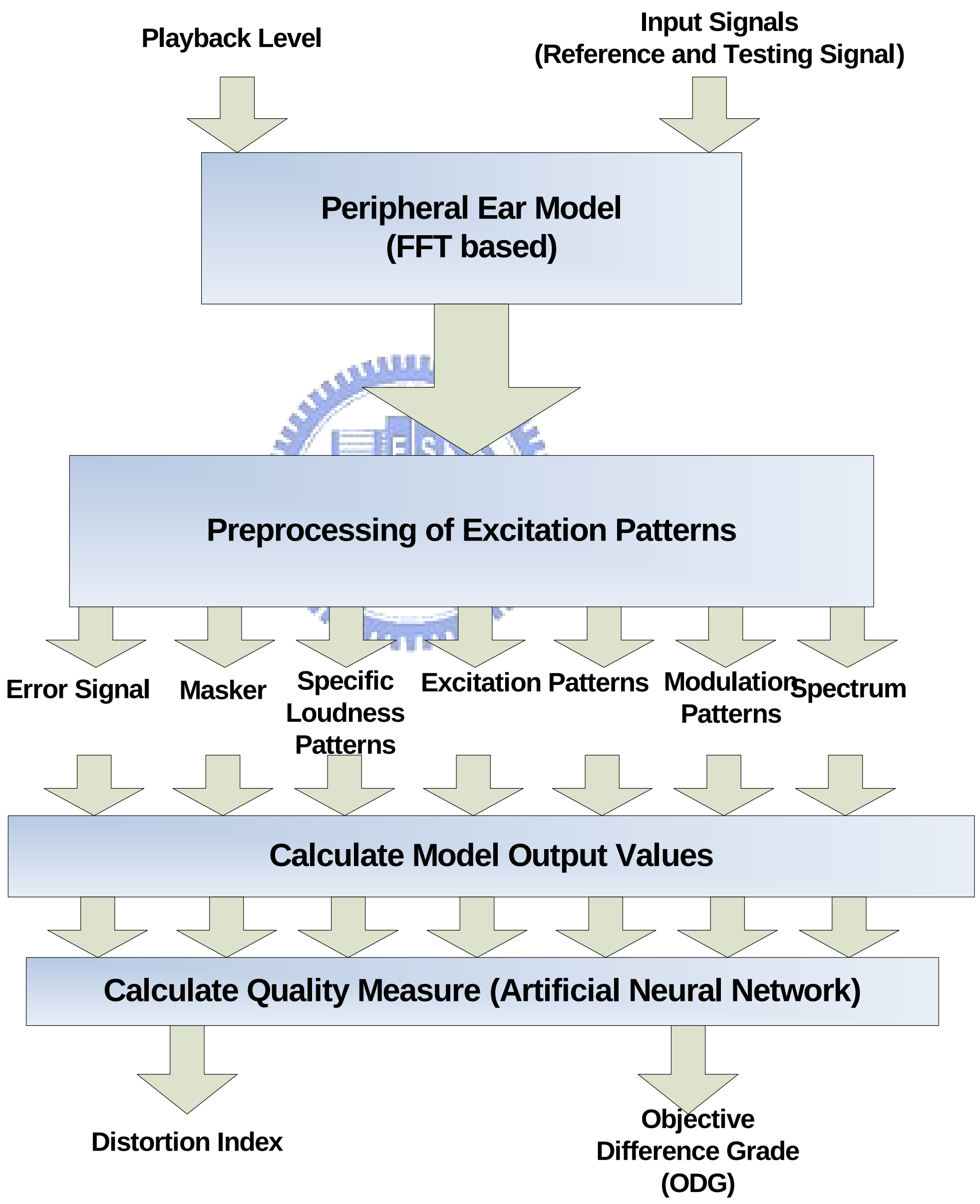
In general, PEAQ compares a signal that has been processed in some way with the corresponding time-aligned original signal. And it extracts perceptually relevant features, which are used to compute a measure of quality. A number of intermediary model output variables (MOVs) are available.

A selected set of MOVs is mapped to an ODG. The mapping was established by minimizing the difference between the distribution of objective measurements and the corresponding distribution of mean subjective qualities for an available data set.

B.2 Description of PEAQ

The block diagram of PEAQ is shown in Fig. 46. PEAQ includes ear models based on the fast Fourier transform (FFT). The model output values are based partly on the masked threshold concept and partly on a comparison of internal representations. In addition, it also yields output values based on a comparison of linear spectra not processed by an ear model. The model outputs the partial loudness of nonlinear distortions, the partial loudness of linear distortions (signal components lost due to an unbalanced frequency response), a noise to mask ratio, measures of alterations of temporal envelopes, a measure of harmonics in the error signal, a probability of error detection and the proportion of signal frames containing audible distortions.

Selected output values are mapped to a single quality indicator by an artificial neural network with one hidden layer [22].



Playback Level
Input Signals (Reference and Testing Signal)
Peripheral Ear Model (FFT based)
Preprocessing of Excitation Patterns
Error Signal
Masker
Specific Loudness Patterns
Excitation Patterns
Modulation Patterns
Spectrum
Calculate Model Output Values
Calculate Quality Measure (Artificial Neural Network)
Distortion Index
Objective Difference Grade (ODG)

Fig. 46 Block diagram of measurement scheme