國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩 士 論 文

環繞 MPEG 編解碼器 之增速及其在 TI DSP 平台上的實現

MPEG Surround Codec Acceleration and Implementation on TI DSP Platform

研 究 生：韓志岡

指導教授：杭學鳴 博士

中 華 民 國 九 十 六 年 六 月

環繞 MPEG 編解碼器 之增速及其在 TI DSP 平台上的實現

MPEG Surround Codec Acceleration and Implementation on TI DSP Platform

研究生：韓志岡	Student: Chih-Kang Han
指導教授：杭學鳴	Advisor: Dr. Hsueh-Ming Hang

國 立 交 通 大 學
電 子 工 程 學 系 電 子 研 究 所 碩 士 班
碩 士 論 文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of Master
in
Electronics Engineering

June 2007

HsinChu, Taiwan, Republic of China

中華民國九十六年六月

環繞 MPEG 編解碼器
之增速及其在 TI DSP 平台上的實現

研究生：韓志岡　　　　　　　　　　　　指導教授：杭學鳴 博士

國立交通大學

電子工程學系 電子研究所碩士班

摘要

隨著音響系統的發展，多聲道系統(Multi-channel audio system)已經廣泛的應用在消費性電子產品上，如 DVD 和數位音訊的廣播服務。環繞 MPEG(MPEG Surround)是由 ISO/IEC MPEG 所制定的一套標準，它能夠在非常低的位元率下壓縮多聲道的音訊訊號。在本篇論文中，對於環繞 MPEG 編碼器的模組，我們提供了較快速的演算法，並且符合 DSP 系統的加速。

我們首先分析環繞 MPEG 編碼器在 DSP 平台上的執行計算複雜度，發現濾波堆在整個系統中耗費最多的計算量，因此我們利用離散正/離弦轉換和快速傅立葉轉換來實現濾波堆的運算並且減少運算量。在低音聲道(LFE channel)中，我們也避免了不必要的計算。此外針對 DSP 的架構，我們使用了一些加速的方法像是定點數運算、使用 L2 快取記憶體、TI DSP 的特殊指令群、迴圈的分解與巨集指令。經由以上的加速方法，在 TI DSP 系統上可對環繞 MPEG 編碼器加速約 500 倍。

另外我們也結合了環繞 MPEG 編碼器和 MPEG-4 HEAAC 編碼器，將之實現在 DSP 平台上，並利用 downsampled SQMF(synthesis Quadrature Mirror Filter)簡化架構。相較於原本的架構，使用簡化的架構可將整體速度提升至約 1.8 倍。

MPEG Surround Codec Acceleration and Implementation on TI DSP Platform

Student: Chih-Kang Han Advisor: Dr. Hsueh-Ming Hang

Department of Electronic Engineering

Institute of Electronics

National Chiao Tung University

Abstract

With the fast development of the audio systems, multi-channel audio systems are commonly used for consumer electronic products such as DVD audio and digital audio broadcasting services. The MPEG Surround is an efficient audio coding standard defined by the ISO/IEC MPEG (Moving Pictures Experts Groups) committee. It is able to compress the multi-channel audio signals at a very low bit-rate. In this thesis, we propose several methods to speed up the MPEG Surround implemented on a DSP platform.

We analyze the complexity of the MPEG Surround encoder and find that among all modules the filterbanks require the most operational cycles on DSP. Hence, we adopt and modify several fast algorithms based on type-IV DCT/DST and FFT for implementing the filterbanks. We also eliminate the unnecessary computation in the LFE (low frequency effect) channel. For the DSP implementation, we use a few DSP code acceleration techniques to speed up such as fixed-point arithmetic, L2 cache, intrinsic function, loop unrolling and DSP macro function. The experimental results show that the modified MPEG Surround encoder is about 500 times faster than the original version.

Furthermore, we implement the combination of the accelerated MPEG Surround encoder and the HEAAC encoder together, and simplify the structure by using the downsampled

SQMF bank. Comparing to the original structure, the simplified encoder is about 1.8 times faster.

誌謝

這篇論文能夠順利完成，首先要感謝我的指導教授杭學鳴老師，在這兩年的研究生涯中，杭老師總是以積極的態度鼓勵我們，不僅在研究上有所指導，也關心我們的日常生活，老師認真踏實的研究態度，更讓我獲益匪淺。

感謝通訊電子與訊號處理實驗室(commlab)，提供了充足的軟硬體資源及良好環境，讓我在研究中不虞匱乏。同時也感謝實驗室的學長、同學、及學弟們，繼大、育彰、鴻志、建志學長時常為我解答課業上的疑問，使我在研究的過程能保持順利。還有在實驗室一同打拼的凱庭、育成、浩廷、耀俞、介遠、柏昇、政達、耀鈞、錫祺等同學們，在這兩年的期間陪伴著我，並不斷的互相討論及勉勵。也要謝謝其他幫助我支持我的好朋友們。

最後，要感謝我的父母，還有我的哥哥，有了你們的栽培及支持，我才能心無旁騖的完成學業。

謝謝所有陪我走過這一段歲月的師長、家人及朋友們!

誌於 2007.6 風城交大

志岡

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction and Motivation

Due to the recent advances of digital audio coding technology, the audio related devices play an important role in our daily life such as hand set and MP3 player. Today, the vast majority of audio playback equipment use traditional two-channel presentations (stereo). Stereo has been a mainstream consumer format for more than 40 years, and so it is not surprising that there is a search for new technologies that further enhance the listener experience. The move towards more reproduction channels ("multi-channel audio" or "surround sound") is quite visible in the market place. This trend is driven by the movie sounds and multi-channel DVDs. However, the complexity and bitrate of the traditional audio coding schemes scale with the number of audio channels. Therefore, it requires a high-quality, low-bitrate audio coding mechanism that can serve both those using conventional stereo equipment and those using next-generation multi-channel equipment.

MPEG stands for ISO "Moving Pictures Experts Groups." It is a group working under the directives of the International Standard Organization (ISO) and the International Electro-technical Commission (IEC). This group work concentrates on defining the standards for coding moving pictures, audio and related data. MPEG audio technology has proposed many low bitrate audio coding such as MPEG-1 layer III (MP3), MPEG-4 Advanced Audio Coding (AAC) and MPEG-4 High-Efficiency AAC (HEAAC). The most recent effort within MPEG is to define a state of the art new standard, the "MPEG Surround", which provides an extremely efficient method for coding multi-channel sound via the transmission of a compressed stereo (or even mono) audio program plus a low-rate side-information channel. In this way, the backward compatibility is retained to pervasive stereo playback systems while the side information permits the next-generation players to present a high-quality multi-channel surround experience.

In this thesis, we implement the MPEG Surround encoder on TI DSP platform. However, the complexity of MPEG Surround is very high due to the new technologies employed. So we adopt several algorithms to reduce its complexity significantly.

1.2 Overview of the thesis

This thesis focuses on developing fast methods for improving the encoding speed of the MPEG Surround codec on the DSP platform. It is organized as follows. In Chapter2, we introduce the spatial hearing phenomena perceived by human. Chapter 3 discusses the algorithm of the MPEG Surround encoder. In Chapter 4, we describe the DSP development environment and the acceleration methods for the TI C6416T DSP system. Chapter 5 discusses our proposed algorithms to accelerate the MPEG Surround encoder on the DSP platform. Finally, we give a conclusions and future work in Chapter 6.

Chapter 2

Spatial Hearing

The sense of hearing creates an auditory image of an external environment. Spatial hearing is the process by which the auditory image is perceived at a particular place. The physical source is called the "sound event", and the perceived sound is the "auditory event" or "auditory object". Since our ability to distinguish and segregate the source is mainly based on the localization of sound source in space, this chapter deals with sound source localization on a horizontal, which provides the base for the Spatial Audio Coding. We will introduce the parameters that are relevant to sound perception and discuss how these phenomena related to the commonly used audio playback systems. The details of spatial hearing can be found in [1].

2.1 Spatial Hearing with One Sound Source

In order to understand how the auditory system distinguishes the direction of a source, the properties of the signals at the ear entrances have to be considered. Generally, the ear input signals can be viewed as being filtered versions of the source signal. The head-related transfer function (HRTF) describes the path of a given sound source to the ear entrances. Figure 2.1 schematically shows a human head and a distant sound source $x(n)$ at angle θ . The entrance signals of the left and right ears are $x_L(n)$ and $x_R(n)$ respectively and they are denoted by two filtered signals through the HRTFs with $h_L(n)$ and $h_R(n)$, respectively. The distances from the source to the left and the right ears are d_L and d_R.

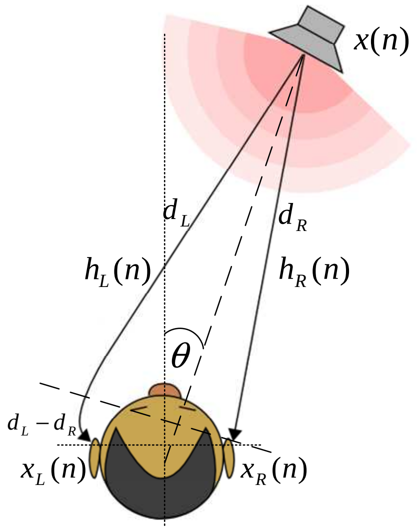


Figure 2.1: HRTFs for left and right ears

2.1.1 Interaural Time Difference and Interaural Level Difference

As a result of different path lengths to the ear entrances, d_L-d_R, there is a difference in the arrival time between both ear entrances, and denoted as interaural time difference (ITD). In addition, the wave fronts from the source impinge upon the left ear directly while the sound received by the right ear is diffracted around the head. This diffraction causes an intensity difference between the left and right ear entrance signals, denoted as interaural level difference (ILD).

In 1907, Lord Rayleigh proposed the duplex theory, which provides an explanation for the ability to localize sounds in the horizontal plane based on the notion of the ITD and ILD (when considering only the frontal directions, -90°≦ θ ≦90°).

Interaural Time Difference (ITD)

ITD is related to the phase of the HRTF ratio and is generally thought to be one of the most important localization cues. It is used to localize low frequency sounds (below 1.5 kHz). In such a frequency range, the wavelength of the sound source is greater than the time delay between the ears. Therefore, there is a phase difference between the sound waves to provide acoustic localization cues. In contrast, at a high frequency, because the wavelength of the sound source is shorter than the distance between the ears, a localization error may occur. The ITD can be estimated by the normalized cross-correlation function as follows.

$$\tau(n) = \arg\max_{d} \left\{ \frac{p_{LR}(d,n)}{\sqrt{p_L(n-d_1)\cdot p_R(n-d_2)}} \right\}, \qquad d_1 = \max\{-d,0\}; d_2 = \max\{d,0\}$$

Where $p_{LR}(d,n)$ is a short-time estimate of the mean of $x_L(n-d_1)x_R(n-d_2)$.

Interaural Level Difference (ILD)

ILD, derived from the amplitude of the HRTF ratio.

$$\Delta L(n) = 10\log_{10}\left(\frac{p_L(n)}{p_R(n)}\right).$$

It is a complicated function of frequency because for any given source positions the peaks and valleys in the HRTF may appear at different frequencies in two ears. Moreover, the ILD is small at low frequencies, regardless of the source position, because the dimensions of the head and pinna are small when compared to the wavelengths of sound at frequencies below about 1.5 kHz. For these reasons, the ILD at individual frequency bands are more likely to be useful localization cues than the overall ILD.

When considering only the frontal direction, a specific ITD-ILD pair can be associated with the perceptual source direction. Figure 2.2 shows an experimental setup for generating left and right ear entrance signals, $x_L(n)$ and $x_R(n)$, with a single source signal x(n). Different ITD-ILD pairs determine the different locations of the auditory events which appear in the frontal sections of the upper head. The ITD is determined by the delays t_L-t_R and ILD is equal to $20\log_{10}(a_L/a_R)$ dB. When left and right signals have the same level and no delay (i.e. ILD=0, ITD=0), an auditory event occurs in a location central to the listener (Region 1). By increasing the intensity of the right side, the auditory event moves from Region 1 to Region 2. When only the left signal is active, the auditory event appears at the left ear as shown in Region 3. The ITD can be used to control the perceptual position of the auditory event in a similar way.

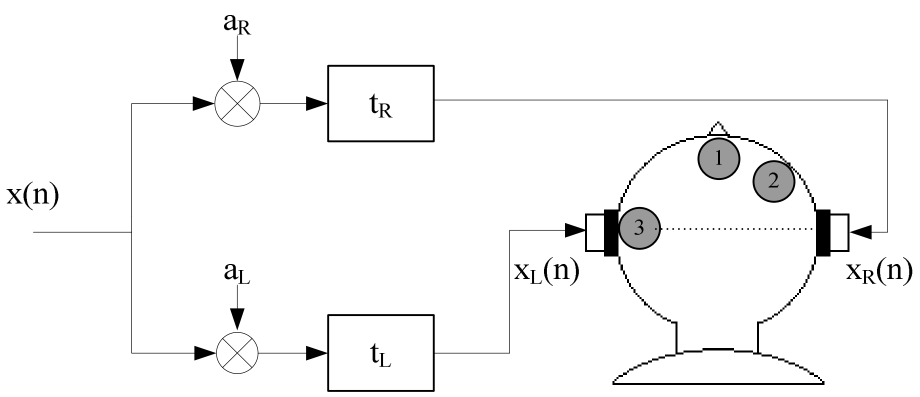


Figure 2.2: Generating the location of auditory event with specific ITD and ILD [1]

2.1.2 Interaural Coherence (IC)

In a reverberant environment, additional effects such as reflection, diffraction and resonance may cause the signals between left and right ears to be incoherent. In order to measure the degree of similarity between the left and right ear entrance signals, another spatial parameter, interaural coherence (IC), is considered. This coherence is derived from the maximum absolute value of the normalized cross-correlation function:

$$IC = \max_{d} \frac{\left| \sum_{n=-\infty}^{\infty} x_R(n) x_L(n+d) \right|}{\sqrt{x_R^2(n) x_L^2(n+d)}},$$

where d corresponds to a small delay.

Figure 2.3 shows that the width (or spatial diffuseness) of the perceived auditory spatial image mostly depends on the IC cues. When the ear entrance signals are identical (IC=1), a compact auditory event is perceived, as illustrated in Region 1. On the other hand, the width of the auditory event increases as the IC decreases (Region 2 and 3). Finally, when the ear entrance signals are independent (IC=0), two distinct auditory events are perceived at the sides (Region 4).

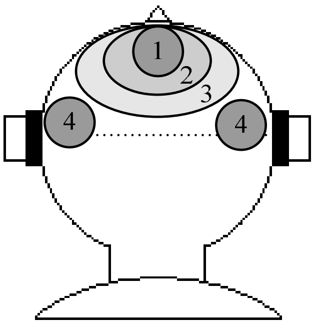


Figure 2.3: Width of auditory event [2].

2.2 Spatial Hearing with Two Sound Sources

The most commonly used consumer playback system for spatial audio systems is the stereo loudspeaker setup. Thus, it is interesting to investigate the spatial hearing with two sound sources. The previous section shows the perceptual effects of ITD, ILD, and IC cues. Similar to these interaural cues, there are three properties of the signals between two loudspeakers

- Inter-channel time difference (ICTD)
- Inter-channel level difference (ICLD)
- Inter-channel coherence (ICC)

In the following paragraphs a few phenomena related to ICTD, ICLD and ICC are reviewed for two sources located in front of a listener.

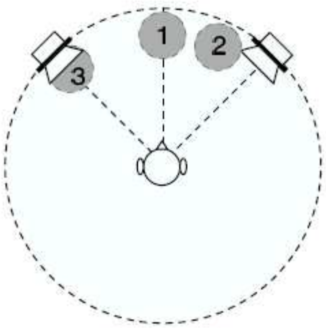


Figure 2.4: ICTD and ICLD between a pair of coherent source signals [1]

Figure 2.4 illustrates different locations of the perceived auditory events controlled by the ICLD-ICTD pair. When the signals emitted from two loudspeakers are identical (ICTD=0 and ICLD=0), an auditory event appears in the center between the two sources as indicated as Region 1. By increasing the intensity of the right channel, the auditory event moves from Region 1 to Region 2. In the extreme case that only the signal on the left is active, the auditory event appears at the left source position (Region 3). As with the IC, the ICC between two loudspeakers is also related to the width of the auditory event, as shown in Figure 2.5. When the ICC between the two loudspeakers decreases, the width of the auditory event increases.

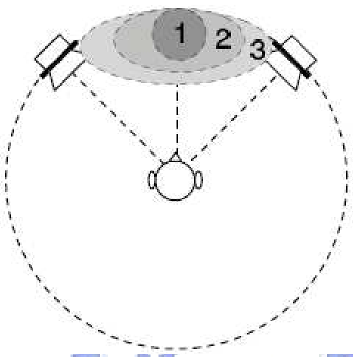


Figure 2.5: ICC with width of the auditory event [1]

2.3 Multi-channel Environment

Multi-channel audio is the name for a variety of techniques used to expand and enrich the sound of audio playback by recording additional sound channels that can be reproduced through additional speakers. Such systems are known as the “home theater systems” for movies. Figure 2.6 illustrates the standard loudspeaker setup for a 5.1 five-to-one (5.1) surround audio system. In the front, three loudspeakers are located at angles -30°, 0°, and +30°. The two surround loudspeakers at the rear, offset by ±110°, are intended to provide the important lateral signal components related to spatial impression. Additionally, one low-frequency effects (LFE) channel is used to carry extremely low sub-bass cinematic effects.

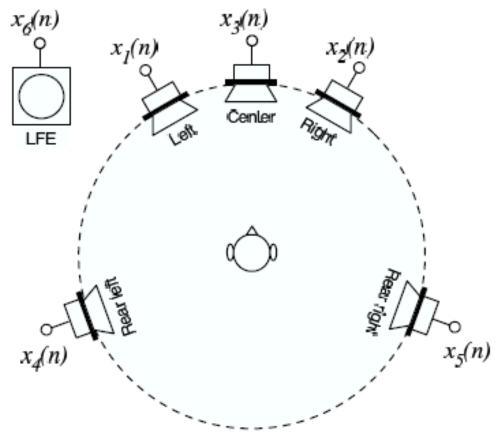


Figure 2.6: Standard 5.1 surround audio system

2.3.1 Generating Sound for 5.1 Systems

Sound source location can often be reproduced successfully using multi-channel recordings. Techniques applied for recording or mixing a two-channel stereo can be applied to a specific channel pair of the five main loudspeakers of a 5.1 setup. For example, to obtain an auditory event from a specific direction, the loudspeaker pair enclosing the desired direction is selected and the corresponding signals are recorded or generated in a way similar to that of the stereo case (resulting in auditory events between the two selected loudspeakers).

2.4 Conclusions

The main point emphasized here is that the perceptual direction of a sound source is determined by ITD, ILD, ICTD, and ICLD. The other parameters, IC and ICC, are used to measure the width (spatial diffuseness) of perceived auditory spatial image. These parameters also play an important role for capturing and generating sound in spatial audio systems, such as stereo or multi-channel audio playback.

Chapter 3

MPEG Surround

In this chapter, we will briefly review several stereo and multi-channel algorithms which are related to the MPEG surround coding, including Intensity Stereo Coding (ISC), Parametric Stereo coding (PS), and Binaural Cue Coding (BCC). Then we will introduce the basic concepts and major modules of the MPEG Surround coder. It is used for compressing a multi-channel audio signal at very low bitrate. It provides an extremely efficient method for coding multi-channel sounds. Finally, an inter-connection of MPEG Surround and MPEG-4 HEAAC structure will be described.

3.1 Related Techniques

3.1.1 Intensity Stereo Coding [3]

Intensity stereo coding (ISC) is a joint-channel coding technique that is a part of the ISO/IEC MPEG family of standards [4]. By removing perceptually irrelevant information between audio channel pairs, it reduces the bit-rate needed for encoding stereo or multi-channel signals. It is more efficient than coding of each channel separately. ISC exploits the fact that the human hearing system is sensitive to low frequency signals at both amplitude and phase; it also sensitive to amplitude of high frequency signals, but low sensitive to phase. Thus, at high frequencies, the original left and right subband signals are replaced by a sum signal and a direction angle (azimuth) which controls the intensity stereo position of the auditory event created at the decoder.

3.1.2 Parametric Stereo Coding [5] [6]

Since ISC is prone to aliasing artifacts and typically is only applied for higher frequency bands, Parametric Stereo (PS) technology is proposed to overcome these limitations. PS is standardized in MPEG-4 and is the next major step to enhance the efficiency of audio compression for low bit-rate stereo signals. It is in conjunction with the context of the MPEG-4 HE-AAC (aacPlus) codec, known as HE-AAC v2, or Enhanced aacPlus.

PS employs a dedicated (complex-valued, over-sampled) filter bank to avoid artifacts due to aliasing resulting from the spectral modification in generating the output channels. In additions, it synthesizes not only the intensities but also the phase differences and coherence between the output channels. Due to these improvements, the PS tool can operate on the full audio bandwidth. In such a system, the stereo signal (a pair of signal) is reconstructed from the transmitted mono signal with the help of the stereo parameters.

3.1.3 Binaural Cue Coding [7] [8]

The Binaural Cue Coding (BCC) approach can be viewed as a generalization of the parametric stereo idea, delivering multi-channel output (with an arbitrary number of channels) from a single audio channel plus some side-information. Figure 3.1 shows the generalized block diagram of BCC encoder and decoder.

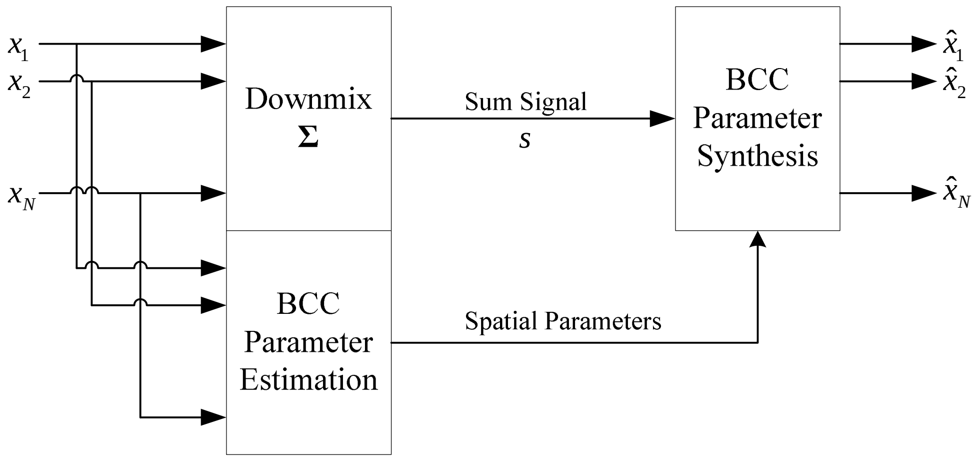


Figure 3.1: Generic Scheme for binaural cue coding (BCC)

In the encoder, multi-channel input channels are combined into a single sum signal by using a downmix process. At the same time, the multi-channel sound image is extracted and parameterized as BCC side-information. The decoder is able to reproduce multi-channel output signals by these data. Because BCC requires only a few bit-rates (2 kb/s) to encode the side-information, the total bit-rate is only slightly higher than what is required to represent a mono audio signal. Another advantage of this scheme is its backwards compatibility with non-multi-channel audio systems. For the receivers that do not support multi-channel sound audio, it simply ignores the side-information and decodes the sum signal.

3.1.3.1 Estimation of BCC parameters

As shown in Figure 3.2(a), the BCC parameters including the inter-channel level difference (ICLD), the inter-channel time difference (ICTD), and the inter-channel coherence (ICC) are estimated in the subband domain. The estimation process is applied independently to each subband.

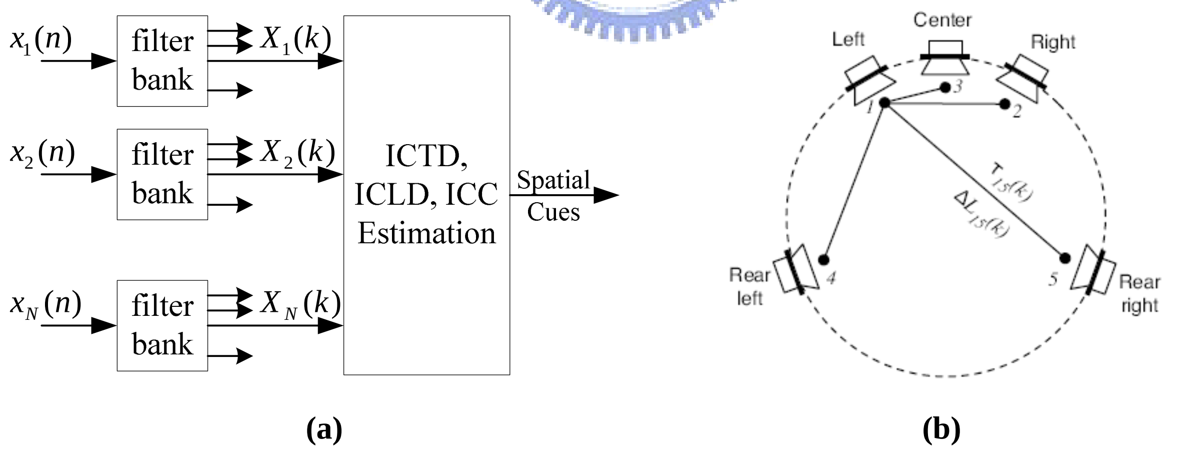


Figure 3.2: BCC parameters estimation

Figure 3.2(b) shows an example of 5-channel environment. The ICTD and ICLD between a reference channel (e.g. Left channel) and the other channels are estimated. One single ICC is estimated between the channel pair with the largest power, to describe the overall coherence among all audio channels.

3.1.3.2 Synthesis of BCC parameters

BCC synthesis scheme is shown in Figure 3.3. First, the downmixed sum signal is converted into the frequency domain via a filter bank. For each output channel, individual time delays and scale factors are imposed on the spectral coefficients to re-synthesis ICTD and ICLD respectively. Followed by a coherence synthesis process, ICC is synthesized. Finally, all output channels are converted back into the time domain signals.

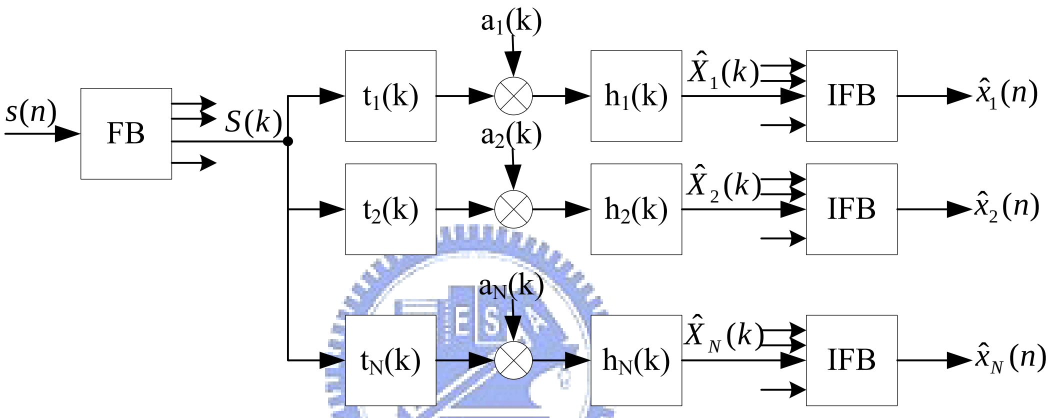


Figure 3.3: BCC synthesis

3.2 MPEG Surround

MPEG Surround can be viewed as an enhancement of the techniques we previously mention, such as a multi-channel extension of Parametric Coding or a generalized version of BCC. In the following sections, we will describe the standardization process of MPEG Surround and its structure.

3.2.1 MPEG Surround Standardization Process

Motivated by the demonstrated potential of what was then called the Spatial Audio Coding approach, ISO/MPEG started a new work item on the parametric coding of multi-channel audio signals by issuing a CfP (Call for proposal) on Spatial Audio Coding in

March 2004 [9]. Four responses were received and evaluated with a number of performance measures including subjective quality of the decoded multi-channel audio signal, the subjective quality of the downmix signals, the spatial cue side information bitrate and the other parameters, such as additional functionality and computational complexity.

As a result of these extensive evaluations, MPEG committee decided that the technology that would be the starting point in the standardization process, called Reference Model 0 (RM0), would be a combination of the submissions from two proponents: Fraunhofer IIS/Agere Systems and Coding Technologies/Philips. These systems not only outperformed the other submissions but also showed complementary performance in terms of per-item quality, bitrate and complexity. Consequently, the merged RM0 (now called MPEG Surround) is designed to combine the best features of both individual systems and was found to fully meet the performance expectation. RM0 provides sound quality substantially the surpassing existing matrixed surround solutions, even for the transmission of a mono downmix signal or for the spatial cue bitrates as low as 6kbit/s. It serves as the basis for the further technical development within the MPEG-4 audio. An extended description of the technology can be found in [10] and [11].

3.2.2 MPEG Surround Reference Model 0 Scheme

Rather than performing a discrete coding of the individual audio input channels, Spatial Audio Coding is a technique to efficiently code a multi-channel audio signal as stereo (or even monaural) signal plus a small amount side information for multi-channel spatial image parameters. Figures 3.4 and 3.5 show the block diagram of the MPEG Surround RM0 encoder and decoder, respectively. The input signals are processed by the analysis filter banks to decompose the input signals into separate frequency bands. The frequency selectivity of these filter banks is tailored specifically towards mimicking the frequency resolution of the human auditory system. Then the MPEG Surround encoder captures the spatial image of a multi-channel audio signal and condenses it into a compact set of parameters. These parameters typically include level/intensity differences and measures correlation/coherence between the audio channels. In parallel, a stereo (or monaural) downmix signal of the sound material is created. The downmix signal is transformed back to the time-domain signal by using the synthesis filter banks. And it is transmitted to the decoder together with the spatial information. On the decoder side the transmitted downmix signal is expanded into high quality multi-channel outputs based on the known spatial parameters.

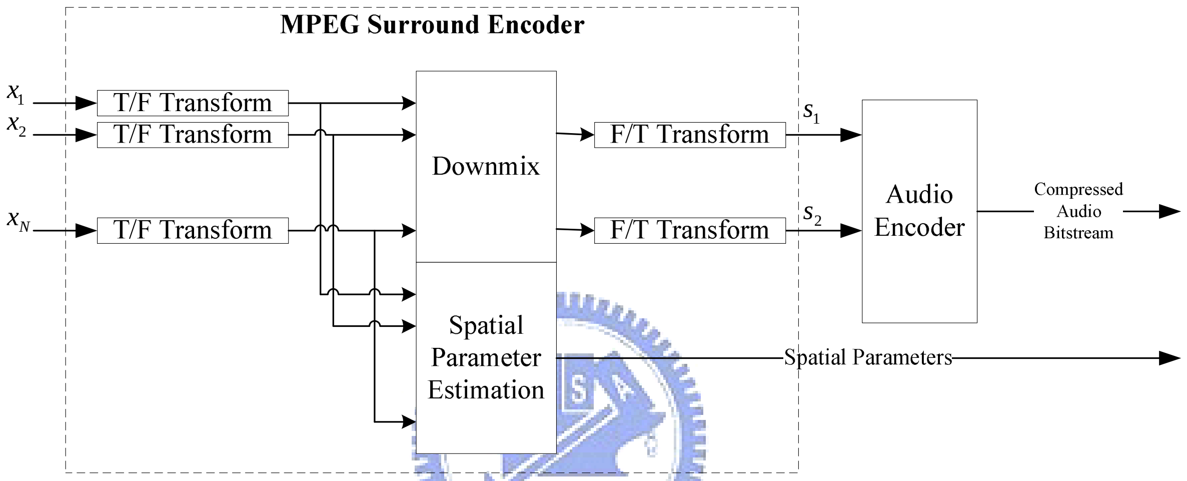


Figure 3.4: MPEG Surround Encoder Overview [15]

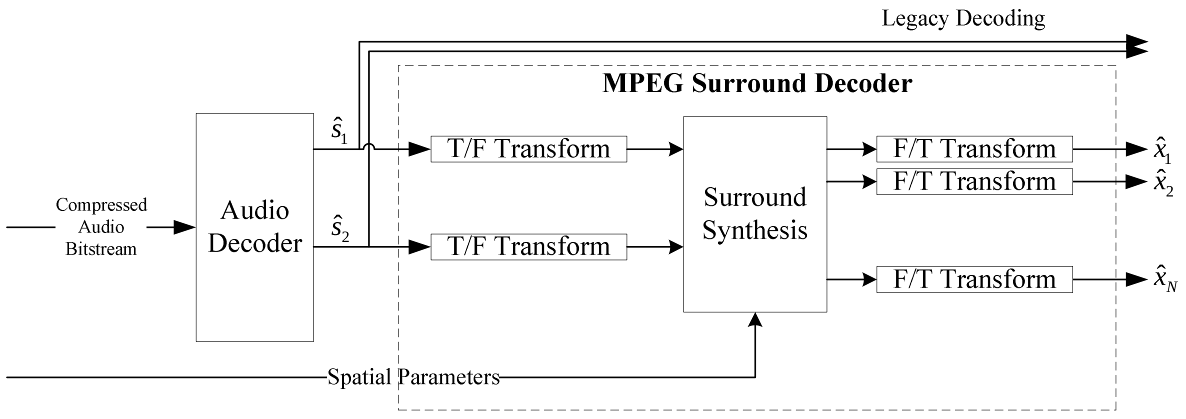


Figure 3.5: MPEG Surround Decoder Overview [15]

Moreover, to achieve a higher compression rate, a MPEG Surround Coding can be combined with a conventional state-of-the-art coder (Audio Encoder and Audio Decoder in Figures 3.4 and 3.5). The downmix signal is encoded with a core coder such as the MPEG-1

Layer III (mp3), MPEG-2/4 AAC or MPEG-4 High Efficiency AAC, or it could even be PCM. In this way, MPEG Surround coder acts as a pre-process to the audio encoder, and as a post-process to the core decoder. Thus, the MPEG Surround Coding is able to provide complete backward compatibility with the non-multi-channel audio systems using the downmix signal: A receiver device without a MPEG Surround decoder will simply decode and present downmix signal.

3.2.3 Time to Frequency Transform

In the human auditory system, the processing of binaural cues is performed on a non-uniform frequency scale. Since the spatial parameters are estimated (at the encoder side) and applied (at the decoder side) as a function of time and frequency, both the encoder and decoder require a transform or filter bank that resemble this non-uniform scale. Furthermore, the transform or filter bank should be over-sampled, since time- and frequency-dependent signal modifications will be made to the signals which would lead to audible aliasing distortion in a critically-sampled system.

It employs a two-stage filter bank to satisfy the above requirments. Figure 3.6 and Figure 3.7 shows the structure of the hybrid QMF analysis and synthesis filter banks, respectively. The first-stage filter bank is a complex-modulated Quadrature Mirror Filter (QMF) bank to obtain a uniform, over-sampled, frequency representation of the audio signal. The signals of the lowest QMF subbands are subsequently fed through a second complex-modulated filter bank to provide a higher resolution of low frequencies.

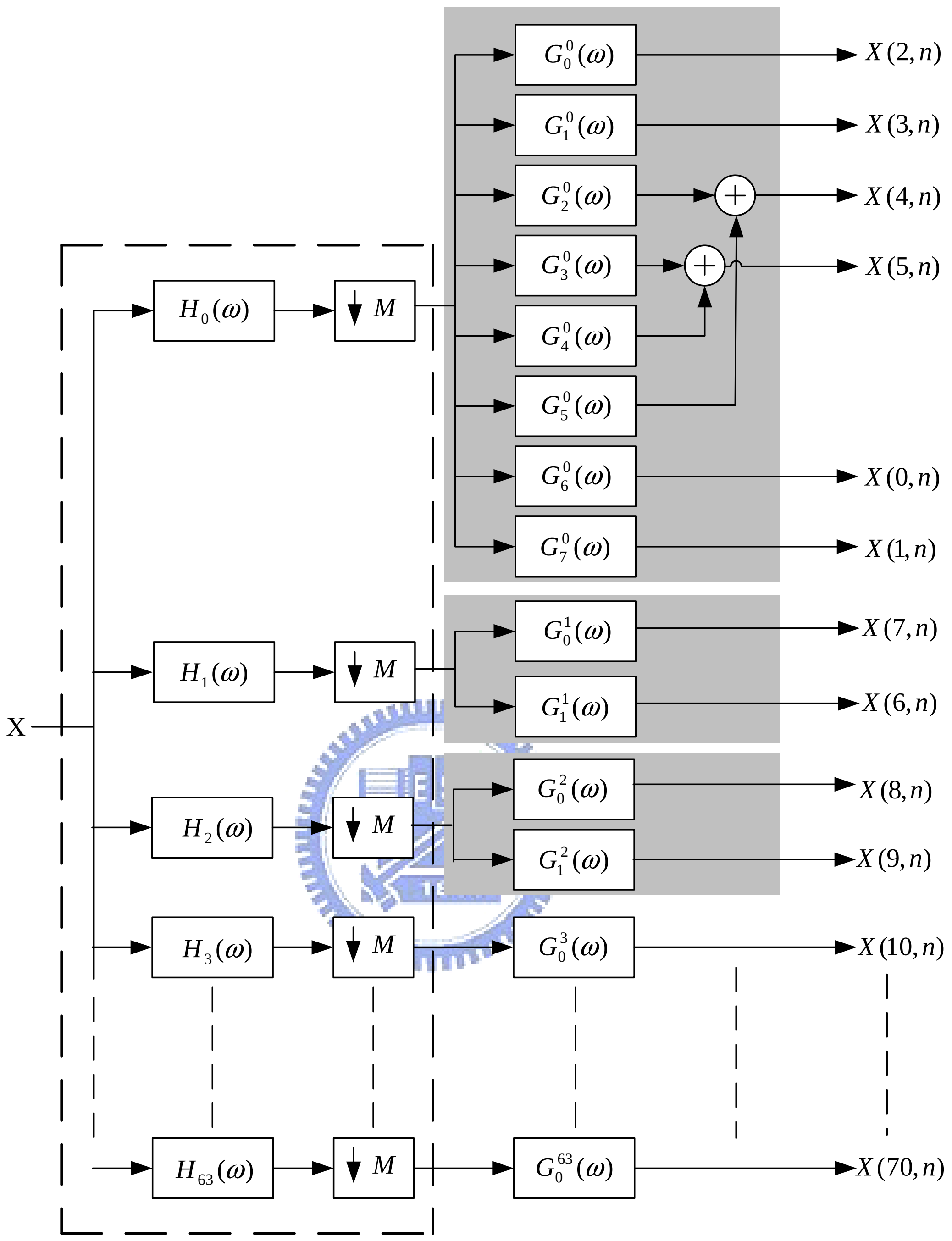


Figure 3.6: Hybrid QMF analysis filter bank providing 71 output bands. The input is fed into a 64-band analysis QMF bank (dashed box). The three lower QMF subbands are further split to increase low frequency resolution (see shadowed box).

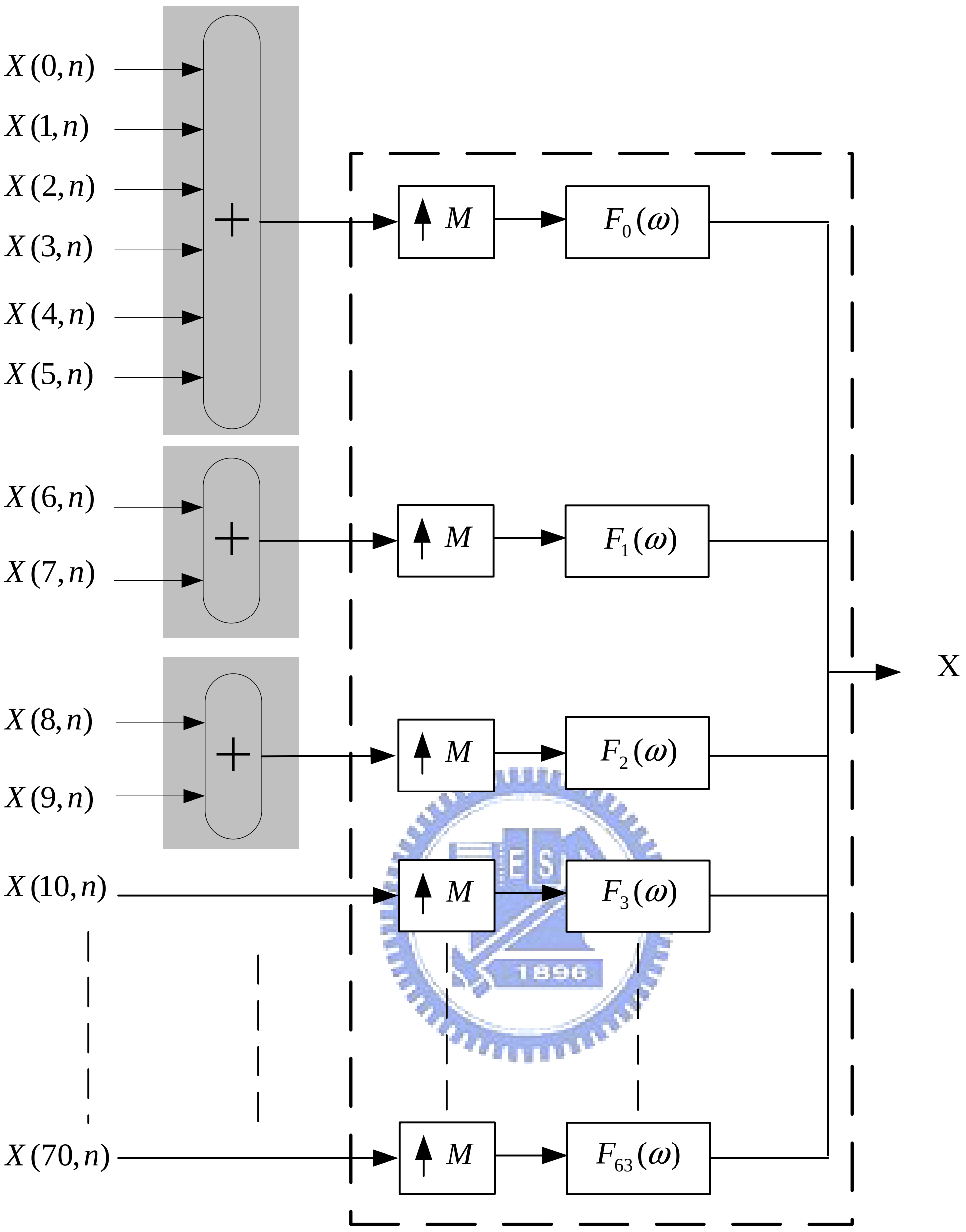


Figure 3.7: Hybrid QMF synthesis filter bank using 71 input bands. The low frequency coefficients are simply added (see shadowed box) prior to the synthesis with the QMF.

3.2.4 Analysis Quadrature Mirror Filter (AQMF) Bank

The first filter bank is compatible with the filter bank used in the SBR algorithms [17]. The subband signals are generated by this filter bank are obtained by convolving the input signal with a set of analysis filter impulse response $h_k[n]$ given by:

$$h_k[n] = p[n]\exp\left\{\frac{j\pi}{M}\left(k+\frac{1}{2}\right)\left(n-\frac{1}{2}\right)\right\},$$

where $p[n]$ represents the low-pass prototype filter impulse with 640 filter length, M represents the number of frequency bands (M=64) and k, the subband index (k=0,…,M-1).

The filtered outputs are subsequently down sampled by a factor M resulting in the down-sampled QMF outputs $X_k[n] = (x * h_k)[Mn]$.

The equation above is purely analytical. In practice, the computational complexity can be reduced by using the poly-phase decomposition method as described in the following steps, in which an array **x** consisting of 640 time domain input samples are assumed. Higher indices in the array correspond to older samples. Figure 3.8 shows the QMF analysis window.

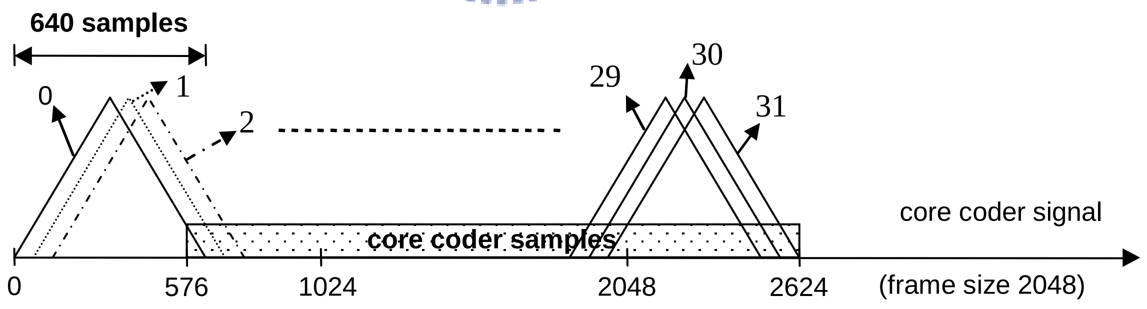


Figure 3.8: QMF analysis windowing [17]. Index 0 to 31 represent 32 windows.

The QMF process is as follows.

1. Shift the samples in the array **x** by 64 positions. The oldest 64 samples are discarded and the 64 new samples are stored in positions 0 to 63.
2. Multiplying the samples of array **x** by window **c** to array **Z** ($Z[n] = x[n]\times c[n]$, for n=0 to

639). The 640 window coefficients **c** are showed in Figure 3.9.

3. Sum the samples according to the formula, $u[n]=\sum_{j=0}^{4} Z[n+128j]$, n=0 to 127, to create the 128-element array **u**.

4. Calculate 64 new subband samples by the matrix operation **X=Mu**, where

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

X (k,j) corresponds to the jth subband sample of the kth QMF subband.

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

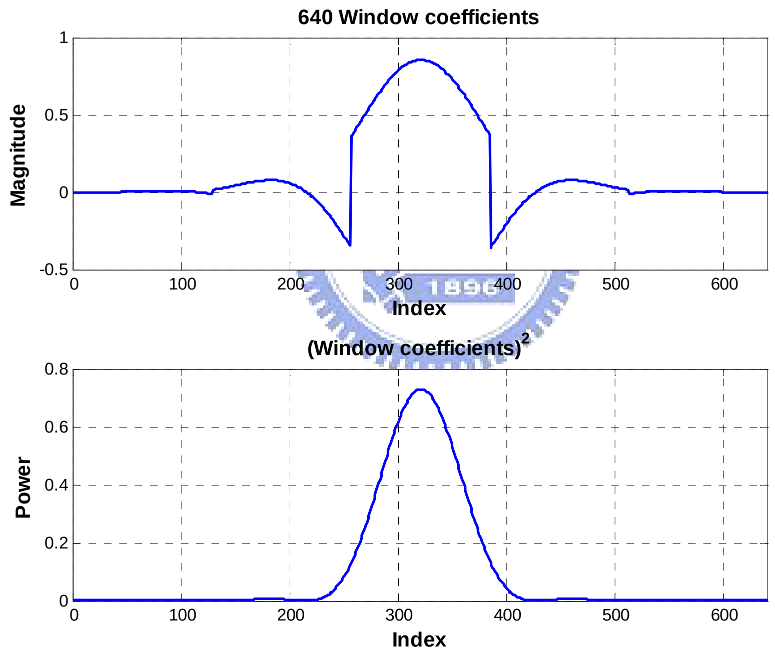


Figure 3.9: Coefficients of the QMF bank window

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

The magnitude responses of the first 4 frequency bands (k=0… 3) of the QMF analysis bank are illustrated by Figure 3.10.

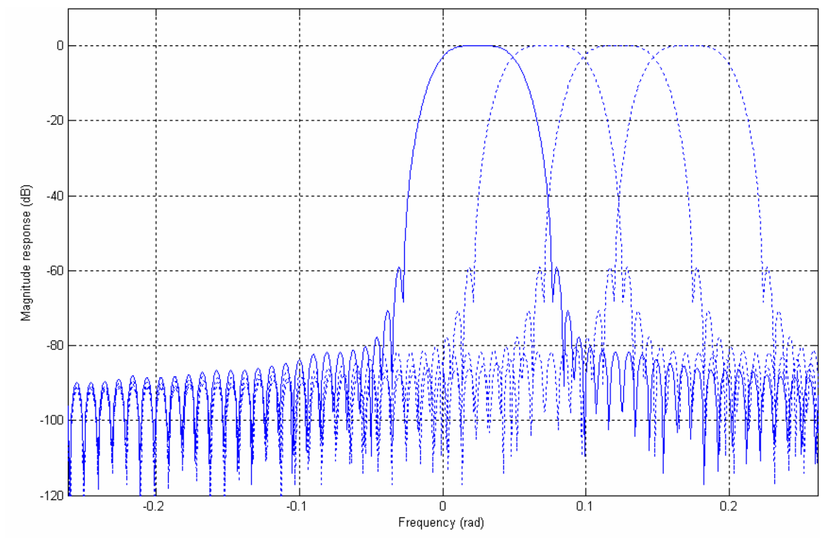


Figure 3.10: Magnitude responses for the first 4 band of the QMF analysis filter bank.

(The magnitude for *k*=0 is highlighted)

3.2.5 Hybrid Filterbank for improved frequency resolution

At a sampling rate of 44.1 kHz, the 64-bands analysis filter bank results in an effective bandwidth of approximately 344 Hz, which is considerably wider than the required spectral resolution at low frequencies. In order to further improve the frequency resolution, the lower QMF subbands are further split with an additional filter bank based on oddly-modulated Q^{th} band filter banks. Depending on the QMF subband, two types of filter have been defined.

Table 3.1: Overview of low frequency split

QMF subband p	Number of bands Q^p	Filter
0	8	Type A
1	2	Type B
2	2	

$$TypeA: G_q^p = g^p[n]\exp(j\frac{2\pi}{Q^p}(q+\frac{1}{2})(n-6)),$$

$$TypeB: G_q^p = g^p[n]\cos(\frac{2\pi}{Q^p}q(n-6)),$$

where g^p represents the 12-order prototype filters in QMF subband *p*, Q^p the total number of sub-subbands in subband *p*, *q* the sub-subband index in QMF channel *p* and *n* the time index.

According to Table 3.1, the first three QMF bands perform sub-subband filtering with Q^0=8, Q^1=2, and Q^2=2. The remaining QMF subbands that are not filtered are delay compensated. This delay amounts to 6 QMF subband samples (i.e. $G_0^k(z) = z^{-6}$ for *k*=[3…63]). Besides, in order to further reduce the complexity of this configuration, some of the filter bank outputs have been summed and resulting total 71 output bands. As an example, the magnitude response of the 8-band sub-filter bank is given in Figure 3.11.

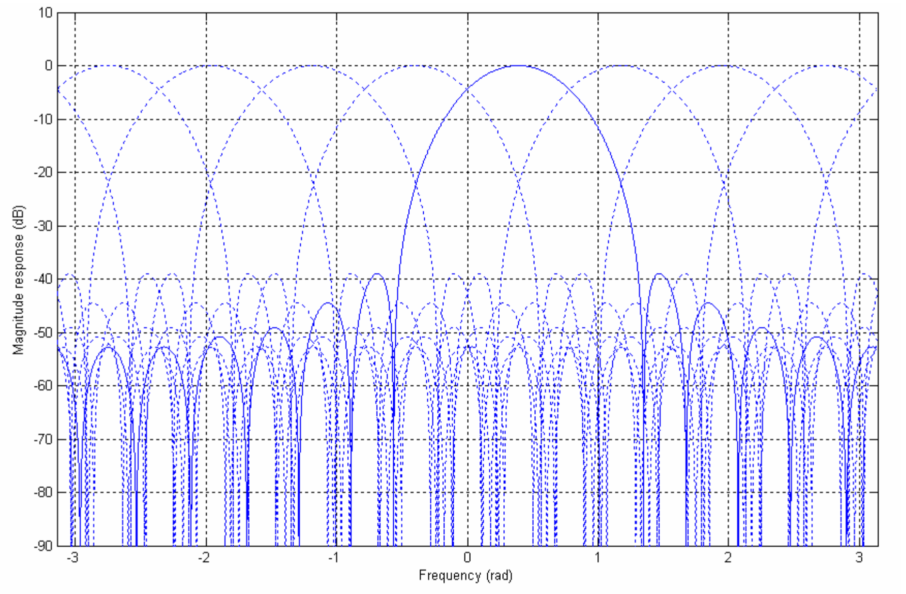


Figure 3.11: Magnitude response of the 8-band sub-filter bank. (subband *q*=0 is highlighted)

3.2.6 Subband Partition

To reduce the complexity and the bitrate of spatial parameters, 71 subband signals are grouped into fewer parameter bands. Since the spatial parameters vary over time and frequency, the psychoacoustic data indicate that a Bark or Equivalent Rectangular Bandwidth (ERB) like frequency scale is appropriate for spatial parameters in the following equation:

$ERB(f) = 24.7(0.00437f + 1)$,

where *f* is the center-frequency in Hz. Hence, the subband coefficients are non-uniformly divided into 20 individual parameter bands according to such a perceptual frequency scale. In each parameter band one set of spatial parameters will be separately estimated and transmitted to the decoder.

3.2.7 Spatial Audio Parameters

The Spatial Audio Coding systems employs two conceptual modules with which it is possible to describe any arbitrary mapping from M to N channels and back, with $N<M$. The structure of the system divides the input channels into pairs of channels that are coded with modules which take two input channels, and produces one output channel (a Reverse One-To-Two module, R-OTT). However, there are also modules that take three input channels and produce two output channels are available (a Reverse Two-To-Three module, R-TTT).

3.2.7.1 R-OTT module

The purpose of the R-OTT module is to create a mono downmix from a stereo input and extract the relevant spatial parameters. For each frequency band, two parameters are computed (assuming input signals x_1 and x_2).

- Channel Level Differences (CLD) – They represent the power ratio of corresponding time/frequency tiles of the input signals, given by:

$$CLD = 10\log 10\left(\frac{\sum_n\sum_m x_1^{n,m}x_1^{n,m*}}{\sum_n\sum_m x_2^{n,m}x_2^{n,m*}}\right).$$

- Inter-channel coherence/cross-correlation (ICC) – They represent the similarity measure of the corresponding time/frequency tiles of the input signals, given by:

$$ICC = \mathrm{Re}\left\{\frac{\sum_n\sum_m x_1^{n,m}x_2^{n,m*}}{\sqrt{\sum_n\sum_m x_1^{n,m}x_1^{n,m*}\sum_n\sum_m x_2^{n,m}x_2^{n,m*}}}\right\}.$$

3.2.7.2 R-TTT module

The R-TTT module performs a downmix from three (l, c, r) to two (l_0, r_0) downmix channels, combined with the generation of the associated spatial parameters. It is appropriate for modeling the symmetrically downmixed center from a stereo downmix pair. The TTT module has two modes of operation.

TTT Energy Mode

The first method comprises an energy-based parameterization, using channel level difference (CLD) parameters.

$$CLD_1 = 10\log 10\left(\frac{\sum_n\sum_m l_1^{n,m} l_1^{n,m*} + \sum_n\sum_m r_1^{n,m} r_1^{n,m*}}{\sum_n\sum_m c_2^{n,m} c_2^{n,m*}}\right).$$

$$CLD_2 = 10\log 10\left(\frac{\sum_n\sum_m l_1^{n,m} l_1^{n,m*}}{\sum_n\sum_m r_2^{n,m} r_2^{n,m*}}\right).$$

This method aims at providing the desired output energy ratios of the signals l, r, and c represented by the energy ratios CLD_1 and CLD_2.

TTT Prediction Mode

A second mode of operation for the R-TTT module is based on transmission of up-mix matrix directly. It makes use of the following parameters.

- Channel Prediction Coefficients (CPC) – The general purpose of the TTT module at the decoder side is to generate three signals from the transmitted stereo downmix signal pair. If the up-mix process from the stereo pair (l_0, r_0) to the signal triplet (l', r', c') is written in matrix notation, the up-mix matrix M_{CPC} is given by:

$$\begin{bmatrix} l' \\ r' \\ c' \end{bmatrix} = M_{CPC}\begin{bmatrix} l_0 \\ r_0 \end{bmatrix}$$

$$M_{CPC} = \frac{1}{2}\begin{bmatrix} c_1+2 & c_2-1 \\ c_1-1 & c_2+2 \\ 1-c_1 & 1-c_2 \end{bmatrix}$$

Here, c_1 and c_2 represent the transmitted CPC parameters. The CPC parameters aim at an optimum reconstruction of the spatial parameters of the signals l', r', and c' (compared to the spatial parameters of the corresponding signals at the encoder side). In other words, given the up-mix matrix M_{CPC}, the parameters c_1 and c_2 can be optimized according to several optimization criteria. However, the recovered signal will in general consist of only partially correlated signals. Therefore, there will be a prediction loss.

- Inter-channel coherence/cross-correlation (ICC) – Unlike the R-OTT module, the ICC parameter here describes the prediction loss cause by the CPC parameters. It is the power ratio between the R-TTT module input and reconstructed signals:

$$ICC = \frac{\langle l \cdot l^* \rangle + \langle r \cdot r^* \rangle + \langle c \cdot c^* \rangle}{\langle l' \cdot l'^* \rangle + \langle r' \cdot r'^* \rangle + \langle c' \cdot c'^* \rangle}$$

Here, <.>denotes the expected value operator, and * denotes complex conjugation.

3.2.7.3 Hierarchical Configuration

To encode 5.1 surround sounds into two-channel stereo in particularly attractive in view of its backward compatibility with the existing stereo consumer devices. Figure 3.12(a) shows a block diagram of a 5.1-to-2 encoder for such a typical system consisting of three R-OTT and an R-TTT module. The signals L, Ls, C, LFE, R and Rs denote the left front, left back, center, LFE, right front and right back channels, respectively. Another example is illustrated in Figure 3.12(b), which shows how the R-OTT modules can be connected in a tree structure, forming a 5.1-to-1 encoder.

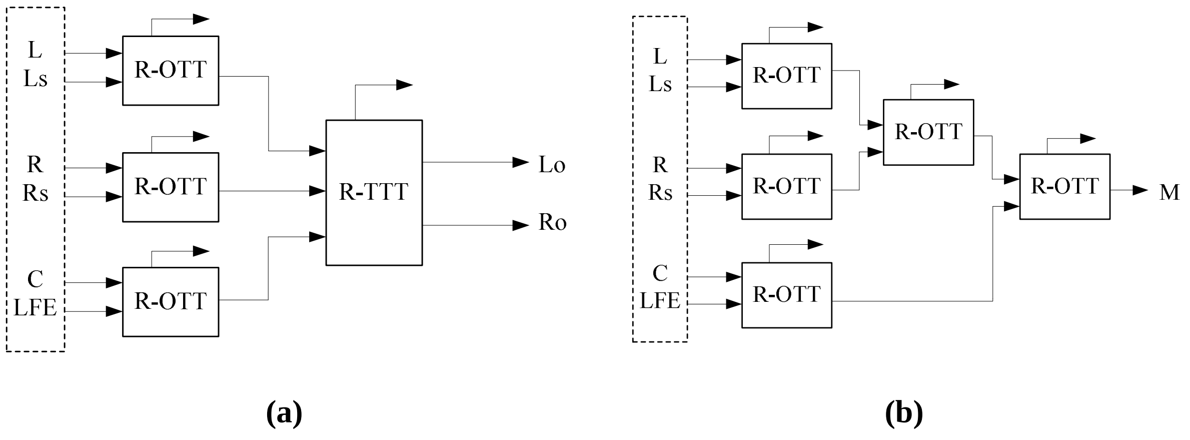


Figure 3.12: (a) 5.1-to-2 MPEG Surround encoder (b) 5.1-to-1 MPEG Surround encoder

3.3 Combination of MPEG Surround and HE-AAC

As mention previously, the MPEG Surround coder can be connected to a state-of-the-art coder. Since the MPEG Surround coder interfaces to the downmix channels by means of a QMF-domain representation, identical to that standardized in the SBR tool of MPEG-4 HEAAC. In the case that the spatial coder is combined with HEAAC, this QMF representation is directly available as an intermediate signal in the HEAAC coder. Figure 3.13 shows the block diagram of MPEG Surround-HEAAC encoder. The output signals of the hybrid synthesis filterbank are directly fed to the SBR tool.

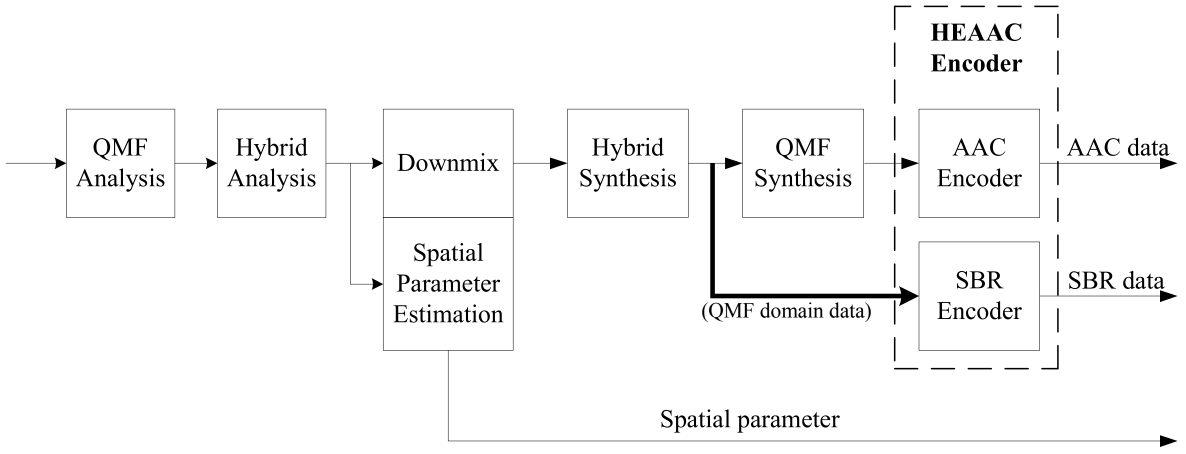


Figure 3.13: Interconnection of MPEG Surround and HEAAC encoder

Chapter 4

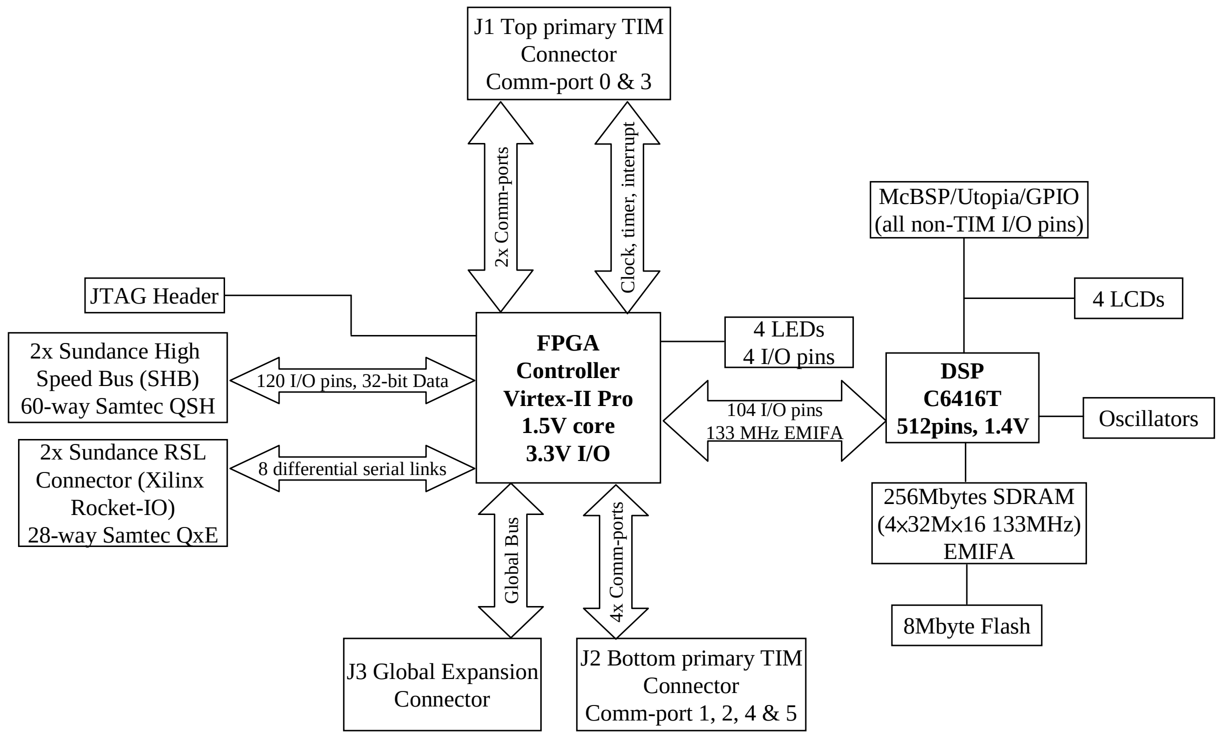
DSP Implementation Environment

We select the TI DSP platform to implement the MPEG Surround encoder. The DSP baseboard (SMT395) is made by Sundance which houses Texas Instruments' TMS320C6416T DSP chip and Xilinx Virtex-II Pro FPGA. Because our implementation is mainly in software, the discussions in this chapter focus on the DSP system environment, DSP chip and its features. Then, the software development tool, Code Composer Studio (CCS), is introduced. At the end, some important acceleration techniques and features which can reduce stalls or hazards on DSP system are also included.

4.1 DSP Baseboard (SMT395)

The block diagram of the Sundance DSP baseboard system (SMT395) is shown in Figure 4.1 [18]. SMT395 utilizes the signal processing technology to provide extreme processing flexibility and high performance. Some important features of SMT395 are listed as follows.

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



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

Figure 4.1: The block diagram of the Sundance DSP Baseboard system [18]

4.2 DSP chip

The TMS320C6416T DSP is using the VelociTI.2 architecture [19]. The VelociTI.2 is a high performance, advanced very long instruction word (VLIW) architecture, making it an excellent choice for multi-channel, multi-functional, and performance-driven applications. VLIW architecture can achieve high performance through increased instruction-level parallelism, and perform multiple instructions during a single cycle. Because parallelism takes the DSP well beyond the performance capabilities of traditional superscalar systems, it is the key to high performance.

The TMS320C6416T DSP chip is the highest-performance fixed-point DSP generation in the TMS320C64x series. According to [19], TMS320C64x series is also a member of the TMS320C6000 (C6x) family. The block diagram of the C6000 family is shown in Figure 4.2. The detailed features of the C6x family devices include:

- Advanced VLIW DSP core
- Eight independent functional units, including two multipliers and six arithmetic units (ALU).
- 64 32-bit general-purpose registers
- Instruction packing to reduce code size, program fetches, and power consumption.
- Conditional execution of all instructions.
- Non-aligned Load and Store architecture
- Byte-addressable (8/16/32/64-bit data), providing efficient memory support for a variety applications.
- 8 bit overflow protection

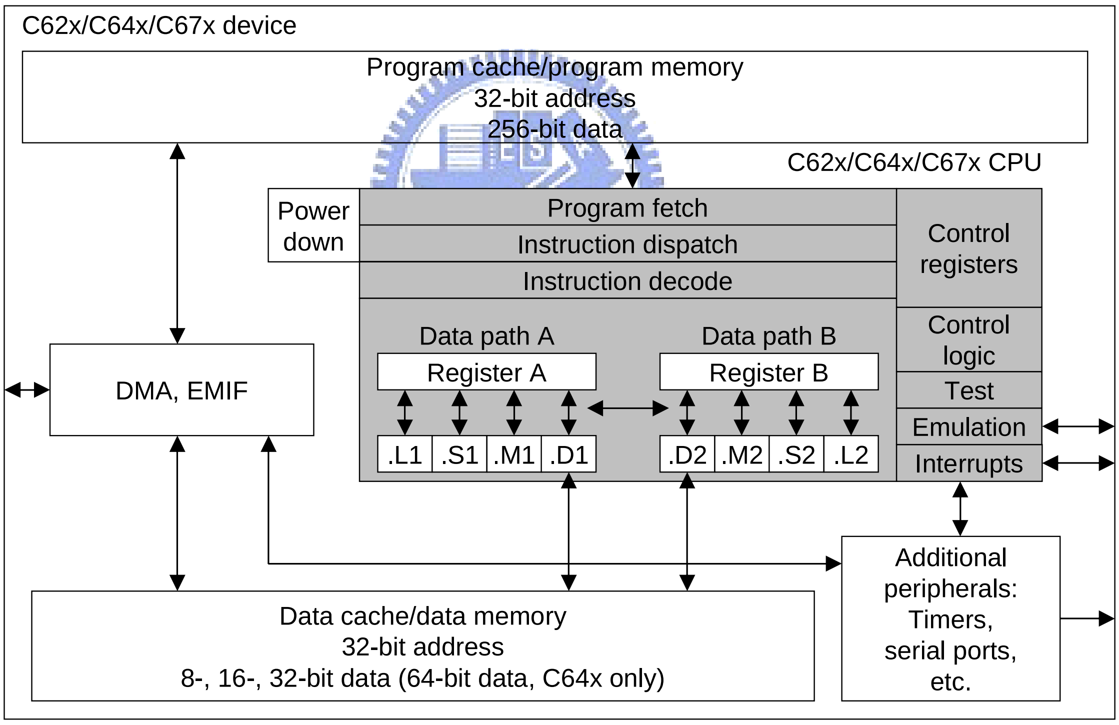


Figure 4.2: Block diagram of TMS320C6x DSP [23]

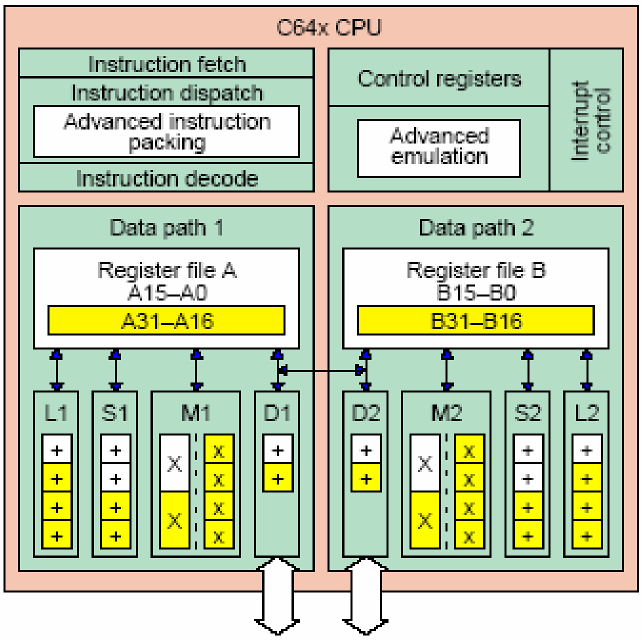
Peripherals such as enhanced direct memory access (EDMA) controller, power-down logic, and two external memory interfaces (EMIFs) usually come with the CPU, while peripherals such as serial ports and host ports are on only certain devices. In the following sections, three major parts of C64x DSP chip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

4.2.1 Central Processing Unit (CPU)

Besides eight independent functional units and sixty-four general purpose registers, the C64x CPU consists of the program fetch unit, instruction dispatch unit, instruction decode unit, two data path, interrupt logic, several control registers and two register files. The DSP chip architecture is illustrated in Figure 4.3.

The instruction dispatch and decode units could decode and arrange the eight instructions to eight function units respectively. The eight function units in the C64x architecture could be further divided into two data paths, A and B. Each path consist four functional units, including one for multiplication operations (.M), another one for logical and arithmetic operations (.L), another one for branch, bit manipulation, and arithmetic operations (.S), and another one for loading/storing, address calculation and arithmetic operation (.D).

C64x CPU



Dual 64 bits load/store paths
Figure 4.3: C64x DSP chip architecture [23]

Two cross-paths (1x and 2x) allow functional units from one data path to access and 32-bit operand to the register file on the other side. There can be a maximum of two cross-path source reads per cycle. There are 32 general purpose registers in each data path, and some of them are reserved for specific addressing or used for conditional instructions. Each functional unit has its own 32-bit bus for writing into a general purpose register file.

4.2.2 Memory and Peripherals

The C64x uses two-level cache-based memory architecture [21] and has a powerful set of peripherals. Level 1 cache is split into level 1 program (L1P) cache and level 1 data (L1D) cache. The size of each L1 cache is 16kB. The level 2 memory/cache (L2) consists of a 1MB memory space and can be optionally split into cache (up to 256 KB) and L2 SRAM (addressable on-chip memory). Besides, it also has one external memory which is a 256 MB SDRAM operated at 133 MHz.

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

The EDMA controller transfers data between regions in the memory map without the intervention by CPU. It could move the data from internal memory to external memory or from internal peripherals to external devices.

The HPI used for communication between the host PC and the target DSP. It is a 16/32-bit wide parallel port through which a host processor could directly access the CPU’s memory space. The host can direct access to memory-mapped peripherals and has ease of access.

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

4.3 TI DSP Code Development Environment

In this section, we will give a briefly introduction about the coding development environment in this project. The Code Composer Studio (CCS) and the coding development flow are illustrated.

4.3.1 Code Composer Studio (CCS)

The Code Composer Studio (CCS) is a software integrated development environment (IDE) for building and debugging programs. We briefly describe some of its features related to our implementation below. The details can be found in [20].

- Real time analysis
- Compile codes and generate Common Object File Format (COFF) output file.
- Provide debug options such as step over, step in, step out, run free, and so on.
- Watches any memory sections when the DSP halts.
- Chip support libraries (CSL) to simplify device configuration.
- Provide debug options such as step over, step in, step out, run free, and so on.
- Support optimized DSP functions such as FFT, filtering, convolution.
- Count the instruction cycles between successive profile-points.
- Arrange code/data to different memory space by linker command file
- Probes a PC file stream into or from the target memory location

We mainly use the CCS tool for debugging, refining, optimizing, and implementing our C codes on DSP. The profiling function helps us to evaluate if our changes to the codes are better or not.

4.3.2 Code Development Flow

The DSP code development can be divided into three steps.

- Step1: Develop the C code without any regard to the particular structure of the C64x. Then, use the debugger to profile the code to identify any inefficient sections in the code. If the performance is not satisfactory, go to step2.
- Step2: Use DSP intrinsic, shell options, and coding techniques for code generation to improve the C codes. Refine the C code procedures such as compiler options, intrinsics, statement, data type modifiers, and code transformations. If the code efficiency is still not sufficient, proceed to step3.
- Step3: Extract the most time-critical areas and rewrite the code in linear assembly. Then, use assembly optimizer to optimize the code.

In general, we do not go to setp3 because the linear assembly is too detail and it takes much more time than in step2. Figure 4.4 shows the steps of the software development flow [20].

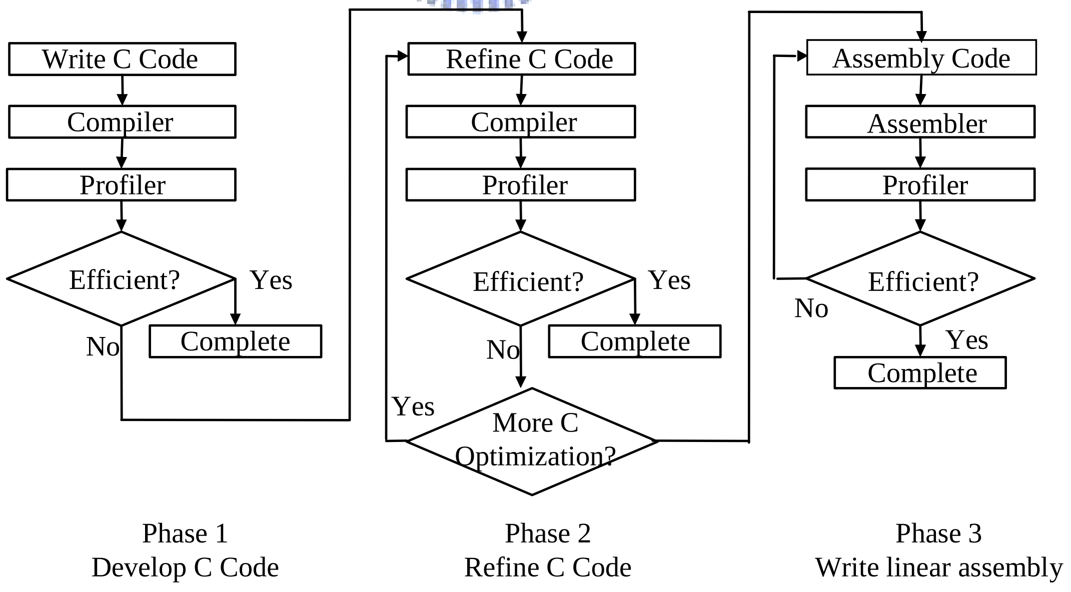


Figure 4.4: Code development flow [20]

4.4 DSP Code Acceleration Methods

In this section, we will describe several methods that can accelerate our code and reduce the execution time on the C64x DSP. Some of these methods are supported by the features of C64x DSP system.

4.4.1 Compiler Option Setting

The CCS compiler translates the source program in more efficient assembly code, and it supports several options to optimize the code. The C/C++ complier is able to perform various optimizations to reduce code size and improve execution performance. The easiest way to invoke optimization is to use the cl6x shell program, specifying the -o*n* option on the cl6x command line, where *n* denotes the level of optimization (0, 1, 2, 3) which controls the type and degree of optimization:

Table 4.1: Options that control optimization [22]

Optimization Level	**Description**
-o0 **(Register)**	■ Performs control-flow-graph simplification ■ Allocates variables to registers ■ Performs loop rotation ■ Eliminates unused code ■ Simplifies expressions and statements ■ Expands calls to functions declared inline
-o1 **(Local)**	Performs all -o0 optimization, and: ■ Performs local copy/constant propagation ■ Removes unused assignments ■ Eliminates local common expressions
-o2 **(Function)**	Performs all -01 optimizations, and: ■ Performs software pipelining ■ Performs loop optimizations ■ Eliminates global common sub-expressions ■ Eliminates global unused assignments ■ Converts array references in loops to incremented pointer form

	■ Performs loop unrolling
-o3 **(File)**	Performs all -o2 optimizations, and: ■ Removes all functions that are never called ■ Simplifies functions with return values that are never used ■ Inline calls to small functions ■ Reorders function declarations so that the attributes of called functions are known when the caller is optimized ■ Propagates arguments into function bodies when all calls pass the same value in the same argument position ■ Identifies file-level variable characteristics

4.4.2 Fixed-point Coding

The C64x DSP is a fixed-point processor, so it can perform fixed-point operations only. Although C64x DSP can simulate floating-point processing, it takes a lot of extra clock cycle to perform the same operation. Table 4.2 is the test results of C64x DSP processing time of instructions "add" and "mul" for different datatypes. The "char", "short", "int", and "long" are fixed-point data types, and the "float" and "double" are floating-point data types. We can see clearly that the floating-point operations need more than 10 times execution cycles comparing to the fixed-point operations. In order to accelerate our program on the C64x DSP, it is necessary to convert the datatypes from floating-point to fixed-point. However, this modification may cause the data losing some accuracy.

Table 4.2: Processing time on the C64x DSP for different data types

Assembly Instruction	char 8-bit	short 16-bit	int 32-bit	long 40-bit	float 32-bit	double 64-bit
add	1	1	1	2	77	146
mul	2	2	6	8	54	69

4.4.3 Loop Unrolling

Loop unrolling expands the loops so that all iterations of the loop appear in the code. It often increases the number of instructions available to execute in parallel. If the codes have conditional instructions, sometimes the compiler does not sure that the branch will be happen or not. It takes more clock cycles to wait for the decision of branch operation. Thus, we can unroll the loop to avoid some of the overhead for branching. Example 4.1 is the loop unrolling and Table 4.3 shows the cycles and code size.

```
(a)
/*Before unrolling*/

int i,a=0,b=0;
for (i=0;i<8;i++)
{
a+=i;
b+=i;
}
```

```
(b)
/*After unrolling*/

int i=0,a=0,b=0;
a+=i; b+=i; i++;
a+=i; b+=i; i++;
a+=i; b+=i; i++;
a+=i; b+=i; i++;
a+=i; b+=i; i++;
a+=i; b+=i; i++;
a+=i; b+=i; i++;
a+=i; b+=i; i++;
```

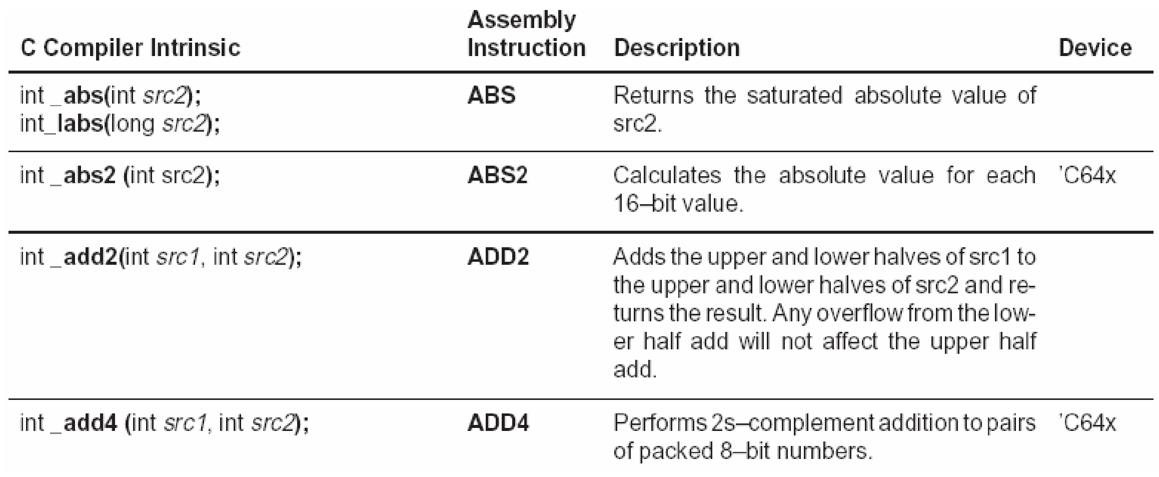
Example 4.1 Loop unrolling.

Table 4.3: Comparison between Rolling and Unrolling

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

4.4.4 Intrinsics and Packet Data Processing

The TI C6000 compiler provides many special functions that map C codes directly to inlined C64x instructions, to increase C code efficiently. These special functions are called intrinsics. Figure 4.5 shows some examples of the intrinsic functions for the C6000 DSP. The entire list of intrinsics for the C6000 DSP can be found in [20].



C Compiler Intrinsic	Assembly Instruction	Description	Device

int _**abs**(int *src2*);
int_**labs**(long *src2*);
	ABS	Returns the saturated absolute value of src2.	

int _**abs2** (int src2);	**ABS2**	Calculates the absolute value for each 16–bit value.	'C64x
int _**add2**(int *src1*, int *src2*);	**ADD2**	Adds the upper and lower halves of src1 to the upper and lower halves of src2 and returns the result. Any overflow from the lower half add will not affect the upper half add.	
int _**add4** (int *src1*, int *src2*);	**ADD4**	Performs 2s–complement addition to pairs of packed 8–bit numbers.	'C64x

Figure 4.5: Intrinsic functions of the TI C6000 series DSP (partial list) [20]

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

4.4.5 Register and Memory Arrangement

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

4.4.6 Using Marcos

Because the software-pipelined loop can not contain function calls, it takes more clock cycles to complete the function call. Hence, we can change the functions to the “define” macros under some conditions. In addition, replacing the function with the macro can cut down the code for initial function definition and reduce the number of branches. However, macros are expanded each time they are called if the function has a number of instructions, so it is not efficient in memory usage.

4.4.7 Linear Assembly

When we are not satisfied with the efficiency of assembly codes which are generated by the TI CCS compiler, we can replace some codes by the assembly codes and then optimize the assembly directly. But this process generally is too detail and very time consumption in practice. Hence, we will do this process at the last step if we have very strict constrains in the processing performance and we have no other algorithms choice.

Chapter 5

Acceleration of MPEG Surround Encoder on DSP Platform

In this chapter, we present several acceleration methods for the MPEG Surround encoder. We will first introduce the MPEG Surround reference software. By analyzing the complexity of the reference encoder, we determine which parts are required to speed up. Then, we propose several methods to reduce the computing time. After proposing the algorithms, we also implement a MPEG Surround-HEAAC encoder on the DSP platform. Finally, we will show the acceleration results.

5.1 Software Environment

5.1.1 MPEG Surround RM0 Reference Software

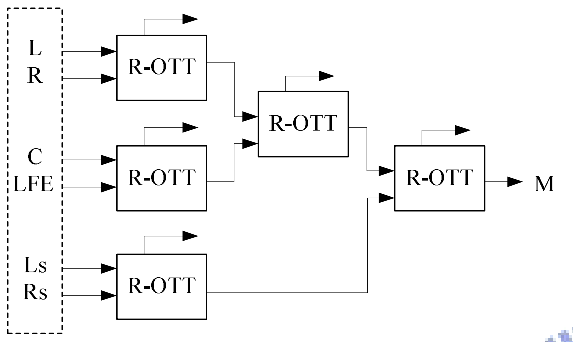
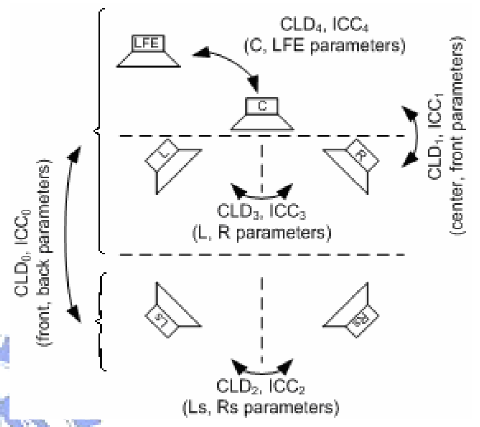
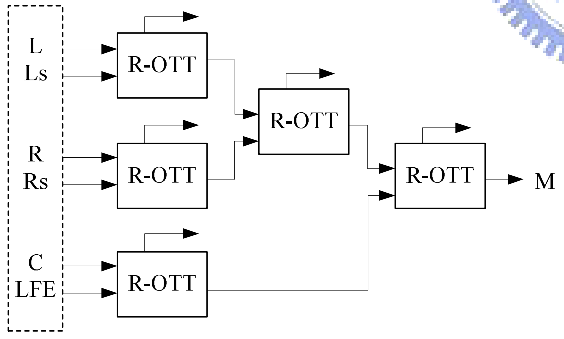
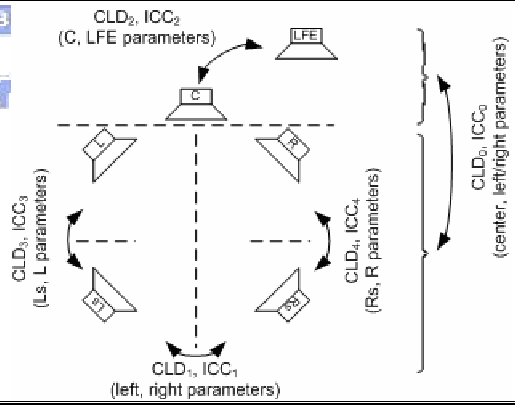
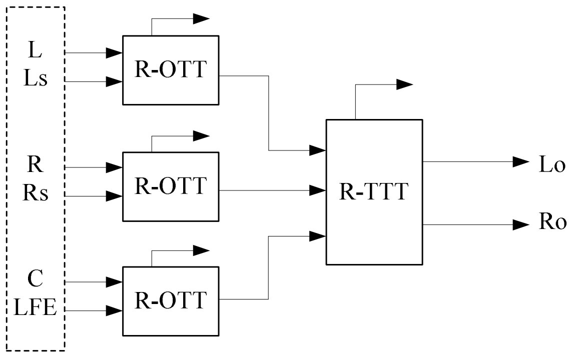
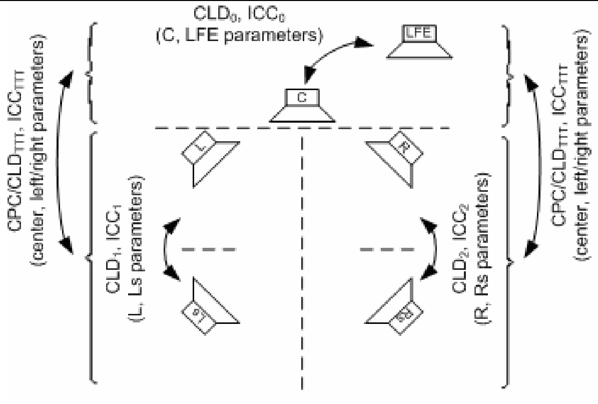
The MPEG Surround group of the MPEG standard committee provides a piece of reference software [25]. It is written in the C programming language for the codec specified by ISO/IEC 23003-1. It was originally developed by Agere Systems, Coding Technologies, Fraunhofer IIS, and Philips. We will introduce the MPEG Surround reference encoder in the next section.

5.1.2 Command Line Switches

In the reference software, there are two command line arguments that control the encoder configuration. The first argument is *Tree-Config*, which defines the tree structure configuration according to Table 5.1. The 5151 and 5152 configurations make use of five

R-OTT modules to produce a mono downmixed output. The encoder with the 525 configuration consists of three R-OTT modules and an R-TTT module, and produces a stereo output.

Table 5.1: Tree structure of the MPEG Surround reference software encoder [10]

5151 Tree Structure	
 L R C LFE Ls Rs R-OTT R-OTT R-OTT R-OTT R-OTT M	 CLD$_4$, ICC$_4$ (C, LFE parameters) LFE C L R CLD$_1$, ICC$_1$ (center, front parameters) CLD$_3$, ICC$_3$ (L, R parameters) CLD$_0$, ICC$_0$ (front, back parameters) Ls Rs CLD$_2$, ICC$_2$ (Ls, Rs parameters)
5152 Tree Structure	
 L Ls R Rs C LFE R-OTT R-OTT R-OTT R-OTT R-OTT M	 CLD$_2$, ICC$_2$ (C, LFE parameters) LFE C L R CLD$_0$, ICC$_0$ (center, left/right parameters) CLD$_3$, ICC$_3$ (Ls, L parameters) CLD$_4$, ICC$_4$ (Rs, R parameters) Ls Rs CLD$_1$, ICC$_1$ (left, right parameters)
525 Tree Structure	
 L Ls R Rs C LFE R-OTT R-OTT R-OTT R-TTT Lo Ro	 CLD$_0$, ICC$_0$ (C, LFE parameters) LFE C L R CPC/CLD$_{TTT}$, ICC$_{TTT}$ (center, left/right parameters) CLD$_1$, ICC$_1$ (L, Ls parameters) CLD$_2$, ICC$_2$ (R, Rs parameters) CPC/CLD$_{TTT}$, ICC$_{TTT}$ (center, left/right parameters) Ls Rs

The reference encoder provides two different time resolutions to extract the spatial parameters. In a spatial frame, there are 32 time slots. The other argument, *ParamSlot*, defines the time slot to which each parameter set applies. The value of *ParamSlot* could be set to 16 or 32. In other words, there could be one or two sets of parameter extracted in a spatial frame. Table 5.2 shows the average spatial parameter bit-rate of different argument settings.

Table 5.2: Bitrates of different argument setting

	Tree Strcture		
	5151	5152	525
ParamSlot =16	20.6 Kbps	20.7 Kbps	16.4 Kbps
ParamSlot =32	9.8 Kbps	9.8 Kbps	7.9 Kbps

5.2 MPEG Surround Encoder Complexity Analysis

We profile the MPEG Surround encoder to find which part takes the most computation time. There are two methods in taking the profiles. One is using the C64xx simulator and the other is using the C6416 simulator. The optimization level is set to -o3 (file level). The profiling results using two methods are shown in Figures 5.1 and 5.2, respectively. The test audio sequence is “choir.wav”, which is a 5.1 channel sequence with a sampling rate at 44.1k Hz. The *Tree-Config* setting is 5151 and the *ParamSlot* is set to 32.

The C64xx simulator simulates the execution cycles of the C64xx core processor with a flat memory system. That is, it does not simulate the cycles needed to access the peripherals and cache system. In contrast, the C6416 simulator supports L1D, L1P, L2 cache, EDMA, Timer, SDRAM and Generic sync RAM Memory models. Thus, the profiling result of the C6416 simulator is closer to the actual cycles on the DSP hardware.

Table 5.3: Cycles on different simulators

	C64xx Simulator		C6416 Simulator	
Total cycles	**6,652,171,801**	**100%**	**109,392,624,928**	**100%**
Initialization	546,109,415	8.2%	10,892,715,255	10.0%
QMF Analysis	4,044,561,270	60.8%	59,290,420,712	54.2%
Hybrid Analysis	1,105,261,974	16.6%	20,850,080,358	19.1%
Ottbox	65,800,045	1.0%	516,609,513	0.5%
Hybrid Synthesis	323,692	0.004%	4,749,476	0.004%
QMF Synthesis	871,205,627	13.1%	17,506,903,336	16.0%
others	18,909,778	0.3%	331,146,278	0.3%

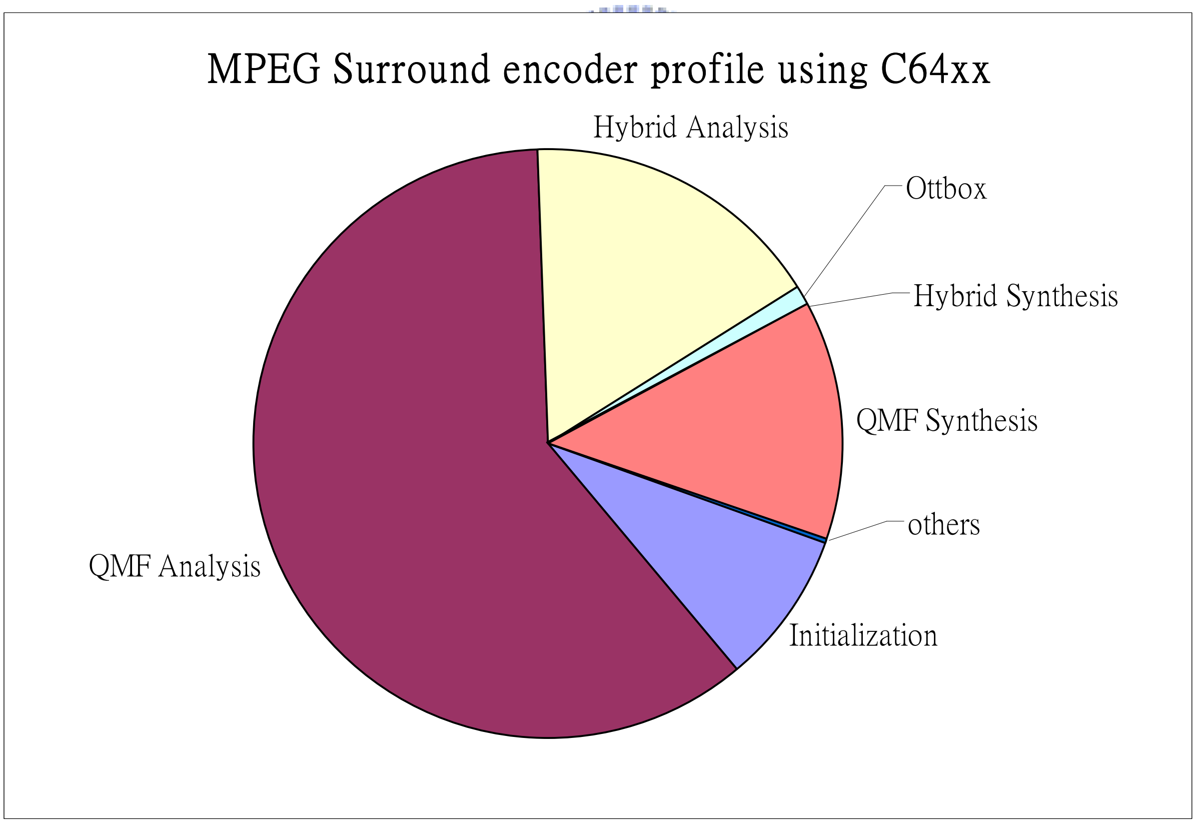


Figure 5.1: Profiling of the MPEG Surround encoder on the C64xx simulator

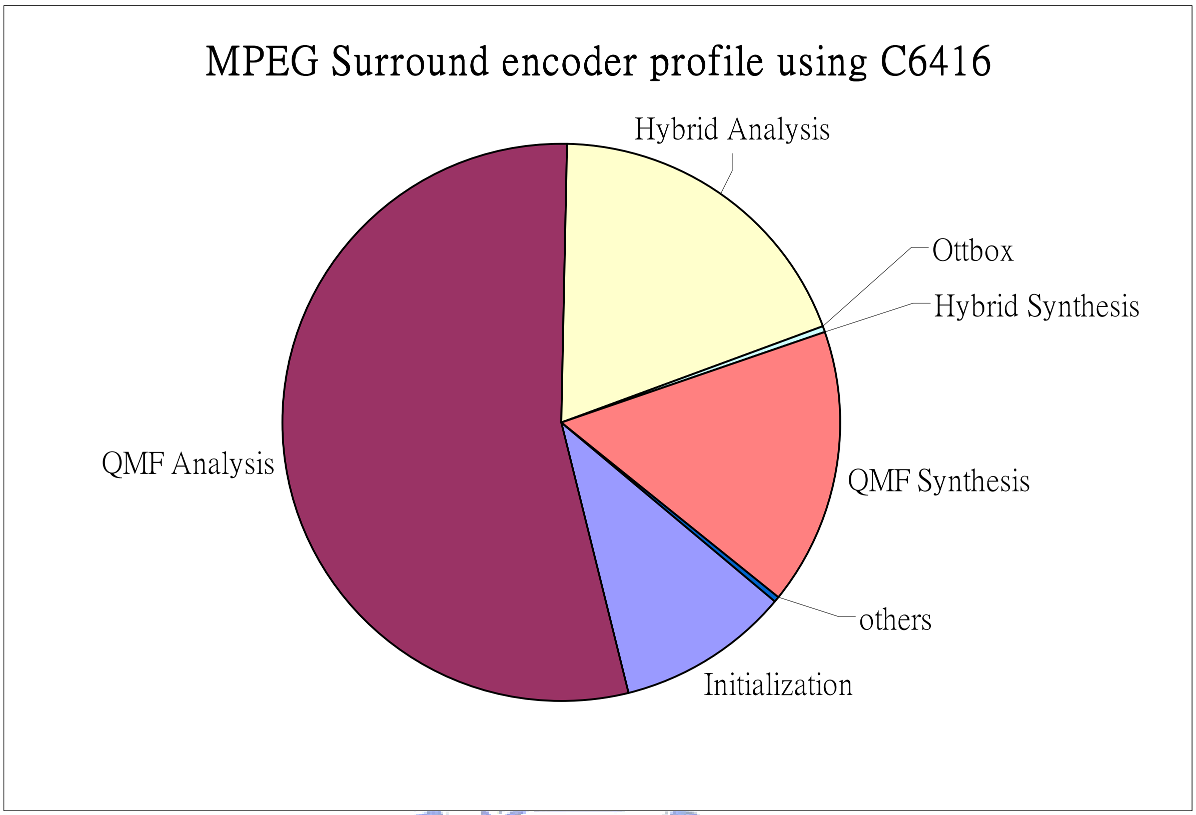


Figure 5.2: Profiling of the MPEG Surround encoder on the C6416 simulator

5.3 Memory System

From the profiling result, the total cycles of the C6416 simulator are approximately sixteen times of that of the C64xx simulator, because the C64xx simulator ignores the memory access of the instructions and data. In other words, about 94% of the total cycles are wasted for memory stalls. It means that the system spends a lot time on transferring data. If we can use the memory system more efficiently, the stall cycles can be decreased.

We can collect the cache information as it is generated by the C6416 simulator, and the result is shown on Table 5.4. The core cycles are only 6% of the total cycles, and the data cache hit rate is less than 1%. In the memory hierarchy of our DSP platform, the L1D cache is too small so that the data cache miss frequently occurs. When data cache miss occurs, it requires additional stall cycles to access data in the external memory. However, as mention in section 4.2.2, the architecture of TI C6000 family provides two-level of cache memory. After enabling the L2 cache, it brings in a great improvement as shown on Table 5.4. The data cache hit rate increases to 99%, so the stall cycles decrease significantly. The percentage of the core cycles also arises to 94%. In this project, the two-level cache configuration gives the most benefit.

Table 5.4: The effect of using L2 cache memory

C6416 Simulator	Original		L2 Cache	
Event	Cycles	Percentage	Cycles	Percentage
Total Cycles	109,392,624,928	N/A	7,046,624,226	N/A
Core cycles(excl. stalls)	6,650,964,730	6.06	6,650,933,220	94.1
NOP cycles	1,466,138,665	22.04	1,466,131,719	22.04
Stall Cycles	102,761,643,954	93.94	415,647,210	5.9
Cross Path Stalls	20,708,884	0.02	20,708,819	0.29
L1P Stall Cycles	66,534,794	0.06	107,803,638	1.53
L1D Stall Cycles	102,675,148,392	93.86	286,779,377	4.07
Instruction cache hits	1,784,119,231	98.31	1,774,843,175	97.8
Instruction cache misses	30,710,512	1.69	39,996,118	2.2
Data cache references	1,365,575,405	N/A	1,365,580,304	N/A
Data cache reads	675,741,828	49.48	675,744,284	49.48
Data cache writes	689,833,562	50.52	689,836,072	50.52
Data cache hits	6,448,798	0.47	1,359,374,412	99.55
Data cache read hits	6,238,724	0.92	671,482,721	99.37
Data cache write hits	210,074	0.03	687,891,691	99.72
Data cache misses	1,359,126,607	99.53	6,205,892	0.45
Data cache read misses	669,503,116	99.08	4,261,532	0.63
Data cache write misses	689,623,491	99.97	1,944,360	0.28

5.4 Floating-point to Fixed-point Conversion

The MPEG Surround reference software is written in C code with floating-point datatypes. However, it is inefficient to implement a floating-point program on a fixed-point DSP such as C64x. Hence, in order to reduce the execution clock cycles, it is necessary to convert the floating-point code to the fixed-point codes.

5.4.1 Word-length Determination

During the conversion, we collect the dynamic range information in the execution functions. Based on the collected information, we choose an appropriate word-length to each variable. The word-lengths must be carefully designed to avoid overflow. Figure 5.3 shows the converted fixed-point data word-lengths between the modules in the MPEG Surround encoder.

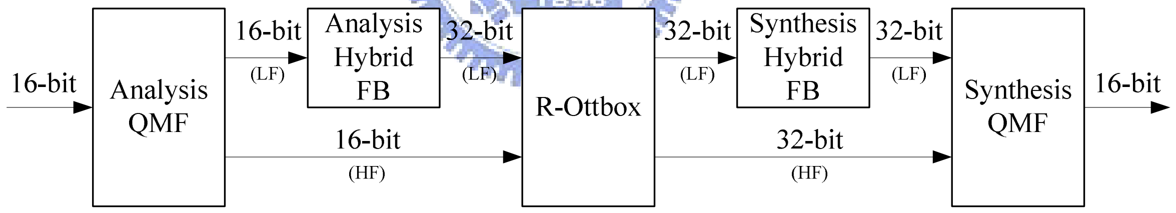


Figure 5.3: Data word-lengths of the fixed-point encoder

The input and output waveforms are stored in 16-bit PCM files. After passing through the analysis QMF filters, the input data are transformed to the sub-band domain with a 16-bit representation. The low frequency QMF coefficients are fed into a second-stage filterbank. The output coefficients of the analysis hybrid filterbank are stored in a 32-bit datatype to maintain their accuracy. Then, the R-Ottboxes perform downmix process and the word-length of the downmixed signal (in the sub-band domain) is 32 bits.

5.4.2 Simulation Results on DSP

Table 5.5 shows the acceleration result of the fixed-point codes. The result is based on the C6416 simulator with L2 cache enabled. We notice that the execution cycles are significantly reduced.

Table 5.5: Performance of fixed-point codes

	Floating-point (Ori.)	Fixed-point	Reduction ratio (%)
Total cycles	**7,046,624,226**	**47,155,001**	**99.33**
Initialization	553,634,278	256,399	99.95
QMF Analysis	4,315,174,856	20,966,993	99.51
Hybrid Analysis	1,137,277,617	14,947,777	98.69
Ottbox	66,845,629	1,352,341	97.98
Hybrid Synthesis	525,237	294,197	43.99
QMF Synthesis	947,689,855	6,136,115	99.35
others	25,476,754	3,201,179	87.43

The fixed-point conversion procedure introduces distortion due to quantization error, so we compare the converted codes with the original floating-point ones and measure the data precision loss in signal-to-noise ratio (SNR).

Table 5.6: SNR due to fixed-point conversion

	SNR
Analysis Filterbank	65.25dB
Synthesis Filterbank	65.04dB
Overall	50.99dB

Now again, we profile the fixed-point version of the MPEG Surround reference encoder. The result is shown in Figure 5.4. We notice the QMF bank and Hybrid analysis filterbank are the major parts of the MPEG Surround encoder. We propose several complexity reduction techniques in the following sections. Our goal is to reduce the codec complexity while not to degrade its output audio quality.

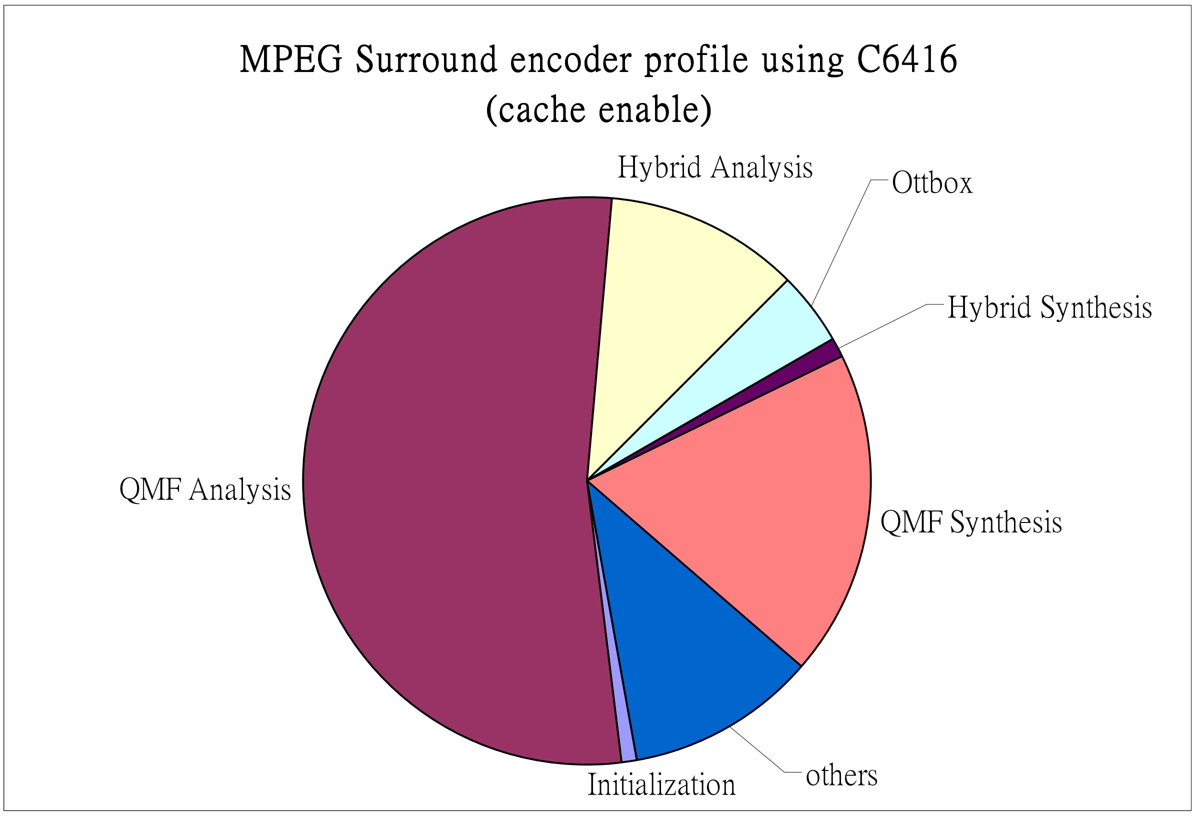


Figure 5.4: Profiling of the MPEG Surround encoder on the C6416 simulator (cache enabled)

5.5 Fast QMF Bank Algorithms

The QMF analysis and synthesis procedures spend the most part of the DSP processing time (about 73%) in the MPEG Surround encoder, so we want to speed up this part to improve total system performance.

5.5.1 Problem Definition

As we mention previously in section 3.2.4, the matrix operation in the analysis QMF is defined as:

$$X(k) = \sum_{n=0}^{127} u(n) \exp\left\{\frac{i\pi(k+0.5)(2n-1)}{128}\right\}, 0 \le k < 64 \qquad (5.1)$$

On the other hand, the matrix operation in the synthesis QMF is:

$$v(n) = \frac{1}{64}\sum_{k=0}^{63} X(k) \exp\left\{\frac{i\pi(k+0.5)(2n-255)}{128}\right\}, 0 \le n < 128 \qquad (5.2)$$

From the equations above, it requires 8192 multiplications and 8064 additions in performing a 64-channel analysis QMF, and 8192 multiplications and 8128 additions in performing a 64-channel synthesis QMF. The computational complexity is huge, so we need fast algorithms for QMF to reduce the complexity.

5.5.2 Analysis Quadrature Mirror (AQMF) Bank

First, we review the type-IV Discrete Cosine Transform (DCT) and Discrete Sine Transform (DST):

DCT-IV: $$X(k)=\sum_{n=0}^{N-1}x(n)\cos\left\{\frac{\pi(k+0.5)(2n+1)}{2N}\right\},0\le k<N \tag{5.3}$$

DST-IV: $$X(k)=\sum_{n=0}^{N-1}x(n)\sin\left\{\frac{\pi(k+0.5)(2n+1)}{2N}\right\},0\le k<N \tag{5.4}$$

Second, we separate (5.1) into real part and imagery part, shown as (5.5).

$$X(k)=X_r(k)+iX_i(k) \tag{5.5}$$

where

$$X_r(k)=\sum_{n=0}^{127}u(n)\cos\left\{\frac{\pi(k+0.5)(2n-1)}{128}\right\},0\le k<64 \tag{5.6}$$

$$X_i(k)=\sum_{n=0}^{127}u(n)\sin\left\{\frac{\pi(k+0.5)(2n-1)}{128}\right\},0\le k<64 \tag{5.7}$$

Consider the real part only

$$\begin{aligned}X_r(k)\ &=u(0)\cos\left\{\frac{\pi(k+0.5)(-1)}{128}\right\}+u(1)\cos\left\{\frac{\pi(k+0.5)(1)}{128}\right\}\\&+\sum_{n=2}^{64}u(n)\cos\left\{\frac{\pi(k+0.5)(2n-1)}{128}\right\}+\sum_{n=65}^{127}u(n)\cos\left\{\frac{\pi(k+0.5)(2n-1)}{128}\right\}\end{aligned} \tag{5.8}$$

In (5.8), there exist symmetrical relationships between the cosine terms:

$$\cos\left\{\frac{\pi(k+0.5)(-1)}{128}\right\}=\cos\left\{\frac{\pi(k+0.5)(1)}{128}\right\}, \tag{5.9}$$

and

$$\sum_{n=65}^{127}u(n)\cos\left\{\frac{\pi(k+0.5)(2n-1)}{128}\right\}=-\sum_{n=2}^{64}u(129-n)\cos\left\{\frac{\pi(k+0.5)(2n-1)}{128}\right\} \tag{5.10}$$

By applying (5.9) and (5.10) to (5.8), yield:

$$\begin{aligned} X_r(k) &= \left(u(0)+u(1)\right)\cos\left\{\frac{\pi(k+0.5)(1)}{128}\right\} + \sum_{n=2}^{64}\left(u(n)-u(129-n)\right)\cos\left\{\frac{\pi(k+0.5)(2n-1)}{128}\right\} \\ &= \left(u(0)+u(1)\right)\cos\left\{\frac{\pi(k+0.5)(1)}{128}\right\} + \sum_{n=1}^{63}\left(u(n+1)-u(128-n)\right)\cos\left\{\frac{\pi(k+0.5)(2n+1)}{128}\right\} \end{aligned} \tag{5.11}$$

Compare (5.11) with (5.3), by denoting

$$\begin{aligned} &u_r(0) = u(0)+u(1), \\ &u_r(n) = u(n+1)-u(128-n), 1 \le n < 64 \end{aligned} \tag{5.12}$$

(5.11) can be modified as:

$$X_r(k) = \sum_{n=0}^{63} u_r(n)\cos\left\{\frac{\pi(k+0.5)(2n+1)}{128}\right\}, 0 \le k < 64 \tag{5.13}$$

Thus, the real part of the QMF coefficient can be computed by applying a 64-point DCT-IV to $u_r(n)$.

Similarly, by denoting

$$\begin{aligned} &u_i(0) = -u(0)+u(1), \\ &u_i(n) = u(n+1)+u(128-n), 1 \le n < 64 \end{aligned} \tag{5.14}$$

The imagery part of the QMF coefficient can be computed by applying DST-IV:

$$X_i(k) = \sum_{n=0}^{63} u_i(n)\sin\left\{\frac{\pi(k+0.5)(2n+1)}{128}\right\}, 0 \le k < 64 \tag{5.15}$$

In summary, the analysis QMF procedure can be decomposed into a 64-point DCT-IV of $u_r(n)$ and a 64-point DST-IV of $u_i(n)$ with a few additional permutations. The signal flow graphs of the decomposition are shown in Figures 5.5 and 5.6, respectively.

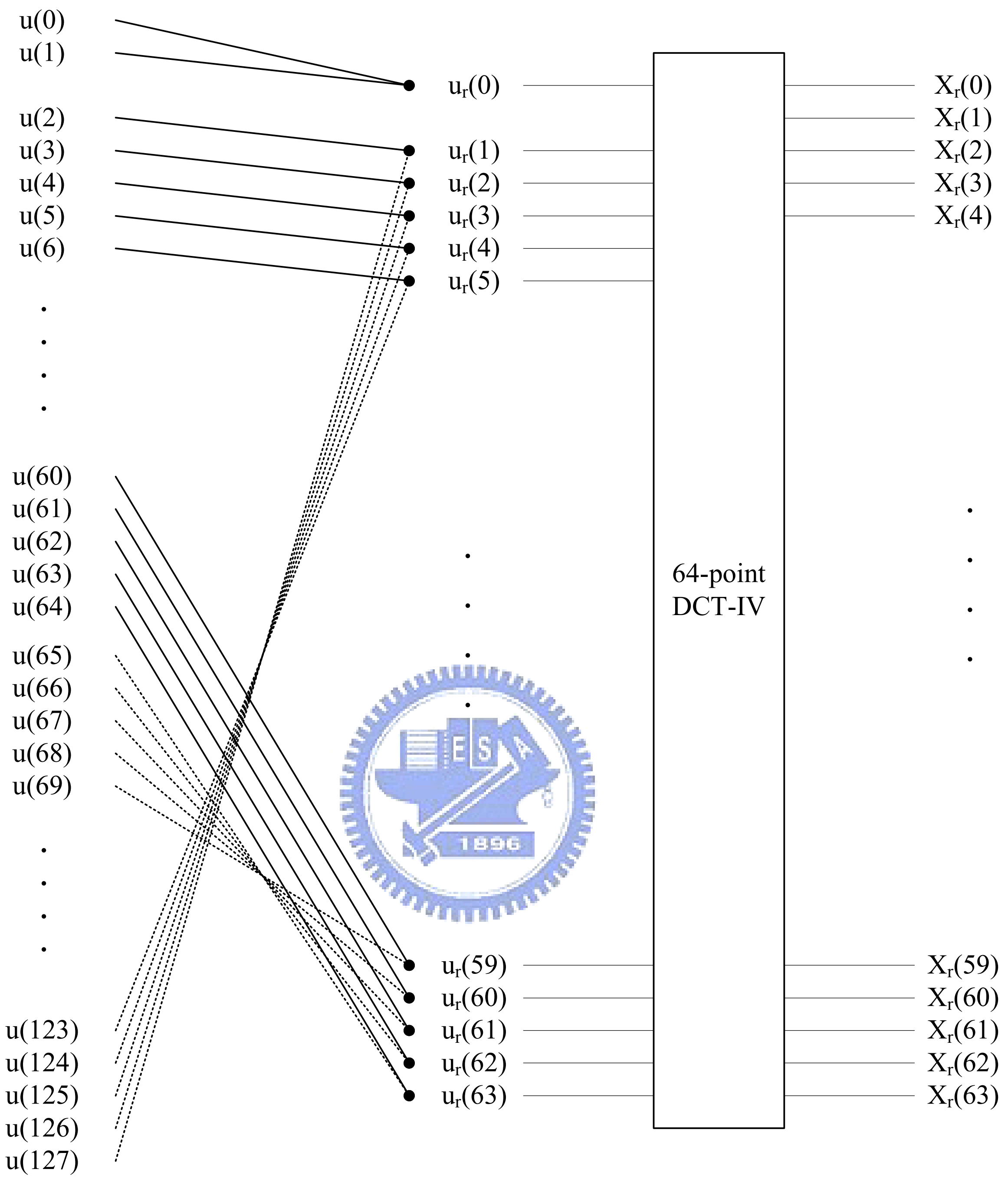


Figure 5.5: Signal flow graph of fast analysis QMF (real part), where the dotted lines denote subtract operations.

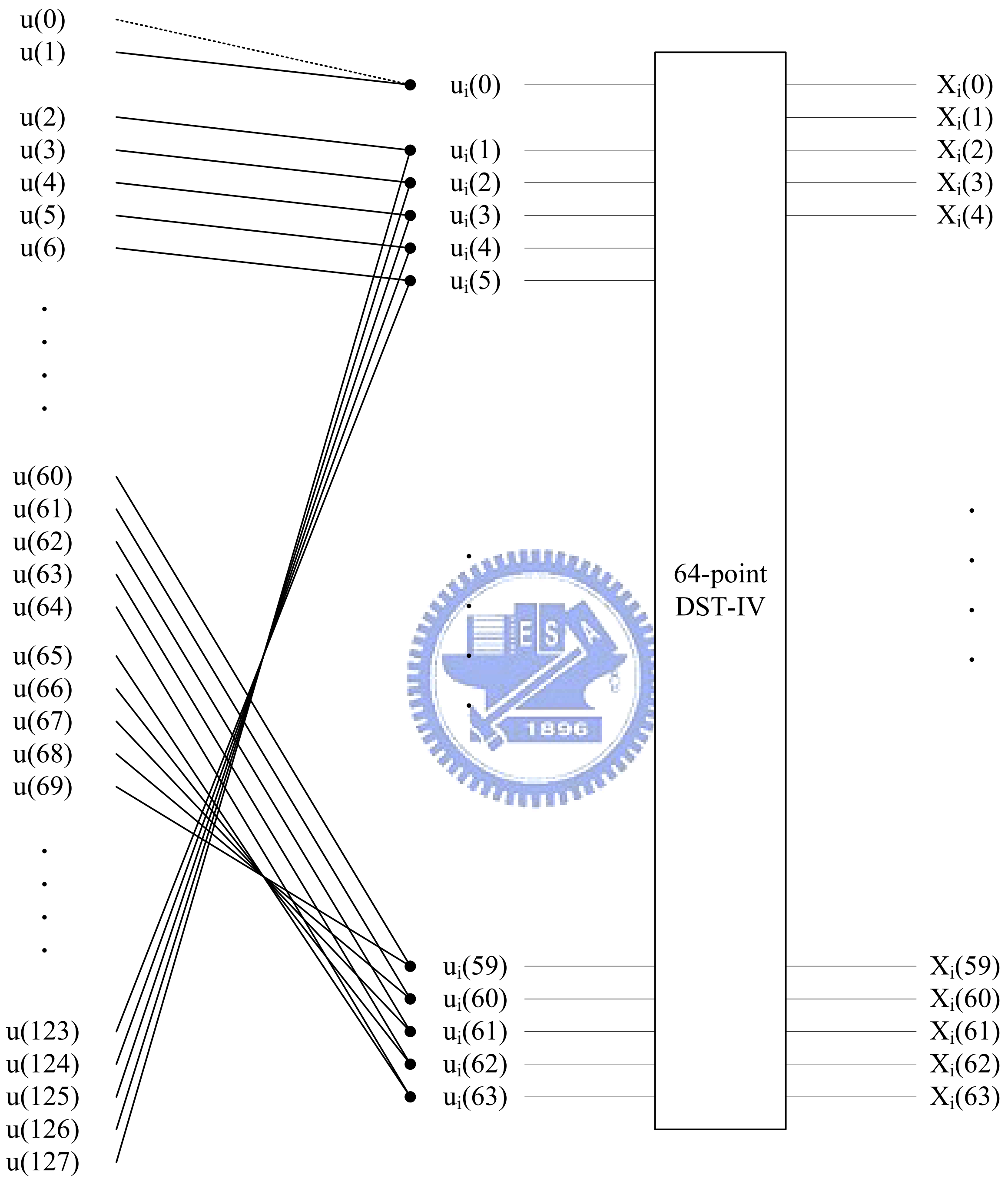


Figure 5.6: Signal flow graph of fast analysis QMF (imagery part), where the dotted lines denote subtract operations.

5.5.3 N/2-point FFT Algorithm for DCT-IV and DST-IV

There are many algorithms published for calculating the DCT-IV and DST-IV to reduce the total number of multiplications or of all arithmetic operations required [26][27]. However, these proposals may not implement well on a very restricted regular structure such as DSP. In [28], R. Gluth proposes a realization for these transforms based on a standard FFT. Since the optimized FFT programs for TI c6x DSP are available, we adopt the efficient FFT algorithm to improve the performance of DCV-IV and DST-IV.

We will describe the forward operation steps of DCT-IV and DST-IV here, and the derivation of this algorithm can be found in Appendix A. The computational steps for DCT-IV are as follows.

1. Compute $u_r = (x_{2r} + jx_{N-1-2r})$
2. Multiply the pre-twiddle: $u_r^{'} = u_r e^{-j\frac{\pi}{N}(r+\frac{1}{8})}$
3. Perform N/2-point complex FFT: $U_k^{'} = FFT\{u_r^{'}\}$
4. Multiply the post-twiddle: $U_k = U_k^{'} e^{-j\frac{\pi}{N}(k+\frac{1}{8})}$
5. Finally, the coefficients of the DCT-IV output are derived from U_k as:

$$C_{2k}^{IV} = \frac{1}{2}\mathrm{Re}\{U_k\} \;\; ; \;\; C_{N-1-2k}^{IV} = \frac{1}{2}\mathrm{Im}\{U_k\}$$

Similarly, the sequence u_r for calculation of the DST-IV is

$u_r = (x_{2r} - jx_{N-1-2r})$,

and the output of DST-IV are assigned to the U_k using the rules

$$S_{2k}^{IV} = \frac{1}{2}\mathrm{Im}\{U_k\} \;\; ; \;\; S_{N-1-2k}^{IV} = \frac{1}{2}\mathrm{Re}\{U_k\}$$

We summarize the FFT-based DCT-IV and DST-IV algorithms by the flow diagram shown in Figure 5.7.

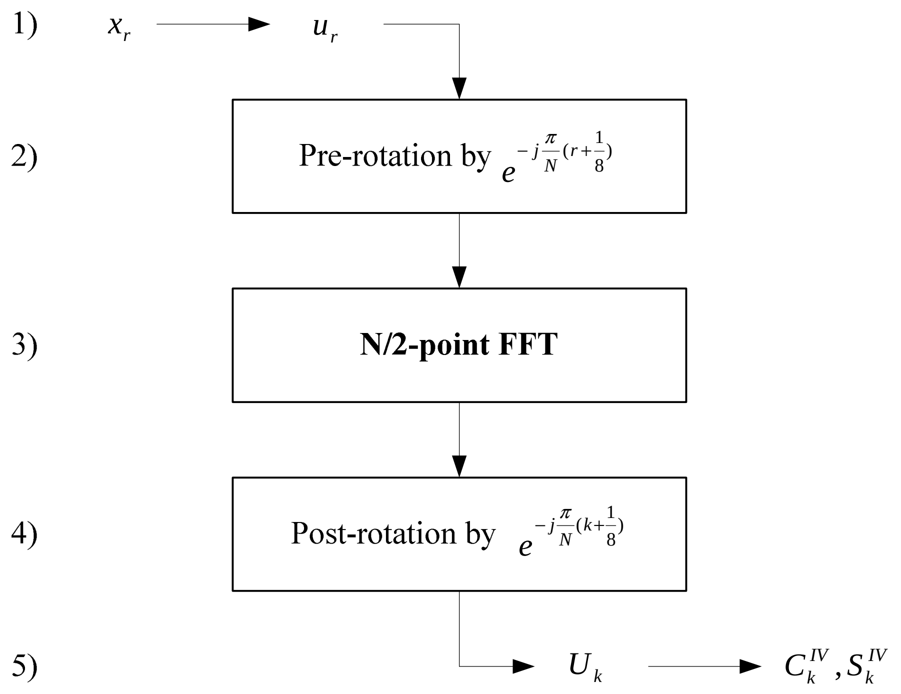

1)
x_r
u_r
2)
Pre-rotation by $e^{-j\frac{\pi}{N}(r+\frac{1}{8})}$
3)
N/2-point FFT
4)
Post-rotation by $e^{-j\frac{\pi}{N}(k+\frac{1}{8})}$
5)
U_k
C_k^{IV}, S_k^{IV}

Figure 5.7: Block diagram of the fast DCT-IV and DST-IV [28]

5.5.4 Using DSP Library

The TI C64x Digital Signal Processor Library (DSPLIB) is an optimized DSP Function Library for C programmers using TMS320C64x devices [30]. It includes many C-callable, assembly-optimized, general-purpose signal-processing routines. By using these routines, the execution speeds are considerably faster than the equivalent codes written in C language.

There are several FFT/IFFT functions available in the DSPLIB. They can be used to replace the C-coded FFT functions in the fast DCT/DST algorithm. The steps of installing the DSPLIB under the Code Composer Studio (CCS) are described as follows.

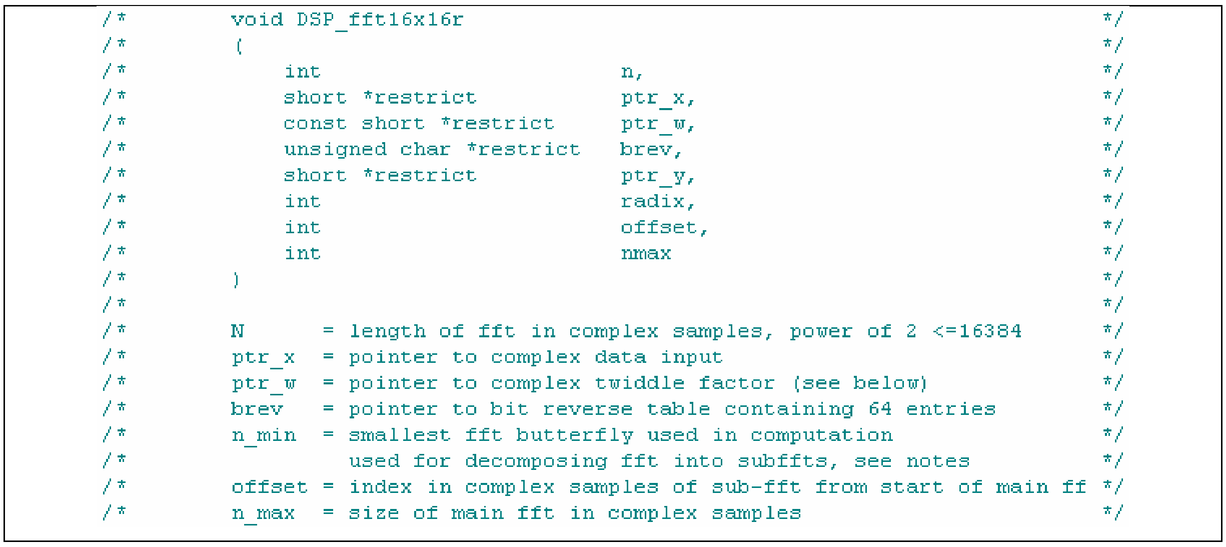
- Link the DSP Library to the application

  Ex: \CCStudio_v3.1\c6400\dsplib\lib\dsp64x.lib

- Include the appropriate header file

  Ex: dsp_fft16x16r.h

- Follow the DSP Library API to use the kernel



```
/*        void DSP_fft16x16r                                                  */
/*        (                                                                   */
/*            int                          n,                                 */
/*            short *restrict              ptr_x,                             */
/*            const short *restrict        ptr_w,                             */
/*            unsigned char *restrict      brev,                              */
/*            short *restrict              ptr_y,                             */
/*            int                          radix,                             */
/*            int                          offset,                            */
/*            int                          nmax                               */
/*        )                                                                   */
/*                                                                            */
/*        N       = length of fft in complex samples, power of 2 <=16384      */
/*        ptr_x   = pointer to complex data input                             */
/*        ptr_w   = pointer to complex twiddle factor (see below)             */
/*        brev    = pointer to bit reverse table containing 64 entries        */
/*        n_min   = smallest fft butterfly used in computation                */
/*                  used for decomposing fft into subffts, see notes          */
/*        offset  = index in complex samples of sub-fft from start of main ff */
/*        n_max   = size of main fft in complex samples                       */
```

Figure 5.8: TI Complex FFT library API [30]

We compare the 32-point complex FFT in the DSPLIB with the C-complied FFT function. Table 5.7 shows the results. The speed of DSPLIB FFT is about 3.8 times faster than that of the C-complied FFT.

Table 5.7: Comparison of C-complied FFT and DSPLIB FFT

	Clock Cycle	Code Size (bytes)
C-complied FFT	740	1804
DSPLIB FFT	196	856

5.5.5 QMF Bank Acceleration Results

We have modified and accelerated the AQMF bank in the fixed-point version of the MPEG Surround encoder. In addition, the synthesis QMF (SQMF) bank can be accelerated in a similar way. Tables 5.8 and 5.9 show the final results. We notice that the reduction ratios of execution cycles are 78% and 73% in the AQMF bank and the SQMF bank, respectively.

Table 5.8: The acceleration result of AQMF bank

Test Sequence	Original AQMF bank (cycles)	Accelerated AQMF bank (cycles)	Reduction ratio (%)
Choir	20,966,993	4,343,490	79.28
ts30	20,952,076	4,421,292	78.90
approaching_tunnel	20,972,479	4,442,427	78.82
carneval	21,002,679	4,472,220	78.71
fountain_music	20,989,067	4,438,179	78.85

Table 5.9: The Acceleration result of the SQMF bank

Test Sequence	Original SQMF bank (cycles)	Accelerated SQMF bank (cycles)	Reduction ratio (%)
Choir	6,136,115	1,665,333	72.86
ts30	6,134,446	1,628,339	73.46
approaching_tunnel	6,134,567	1,628,460	73.45
carneval	6,131,875	1,625,768	73.49
fountain_music	6,130,129	1,624,012	73.51

5.6 Fast Hybrid Filterbank Algorithms

The lowest three QMF subbands are fed through a second complex-modulated filterbank to further enhance the low frequency resolution. The sub-filterbank has a filter length of L =12, and the analysis filter for QMF sub-band p is given by:

$$S_q^p[n] = s^p[n] * g_q^p[n] \qquad (5.16)$$

where

$$g_q^0[n] = g^0[n]\exp\left\{ j\frac{2\pi}{Q^0}(q+\frac{1}{2})(n-6) \right\}$$

$$g_q^{1,2}[n] = g^{1,2}[n]\cos\left\{ \frac{2\pi}{Q^{1,2}} q(n-6) \right\}$$

where $g^p[m]$ is the prototype filter, Q^p, the number of frequency bands, and $q = 0,1,...,Q^p - 1$,the frequency index of the resulting sub-subband signals $S_q^p[n]$.

The sub-filter is time consuming and complex because the sub-filterbank outputs do not down-sampled. However, it can be also implemented by using the DFT algorithms.

5.6.1 Fast Analysis Hybrid Filterbank

The sub-filterbank splits the 0th QMF band coefficients into 8 sub-subbands and it splits the 1st and 2nd QMF band coefficients into 2 sub-subbands, respectively. We derive the fast algorithm for band 0 only because the others are considerably simple. For simplicity, the QMF band index p is omitted, and (5.16) yields:

$$S_q[n] = \sum_{m=0}^{12} s[n-m]g[m]\exp\left\{ j\frac{2\pi}{8}(q+\frac{1}{2})(m-6)\right\} \tag{5.17}$$

Change the index expression

$$m = 8m_1 + m_2, \qquad \begin{matrix} m_1 = 0,1 \\ m_2 = 0,...,7 \end{matrix} \tag{5.18}$$

With (5.17), this yields

$$S_q[n] = \sum_{m_2=0}^{7} a_{m_2}[n]\exp\left\{ j\frac{2\pi}{8}(q+\frac{1}{2})(m_2-6)\right\} \tag{5.19}$$

where

$$a_{m_2}[n] = \sum_{m_1=0}^{1} (-1)^{m_1} g[8m_1+m_2]s[n-8m_1-m_2] \tag{5.20}$$

Equation (5.20) can be viewed as a polyphase decomposition of $g[n]$

By expending the exponential term in (5.19):

$$S_q[n] = \sum_{m_2=0}^{7} (a_{m_2}[n]e^{j\frac{\pi}{8}(m_2-6)})\exp\left\{ j\frac{2\pi}{8}q(m_2-6)\right\} \tag{5.21}$$

Compare (5.21) with the 8-point complex FFT, the above equations show that the hybrid analysis filterbank can be calculated by the following steps:

1. Perform the polyphase filtering to compute a_{m_2} by equation (5.20).
2. Circularly shift the indices of a_{m_2} by 6: $a'_{m_2} = a_{(m_2+6)\%8}$
3. Multiply the twiddle factor: $b_{m_2} = a'_{m_2} e^{j\frac{m_2\pi}{8}}$
4. Perform the 8-point complex FFT: $S_q = FFT\{b_{m_2}\}$

The flow diagram of the analysis hybrid filterbank is shown in Figure 5.9. Again, the clock cycles of FFT can be further reduced by using the DSPLIB.

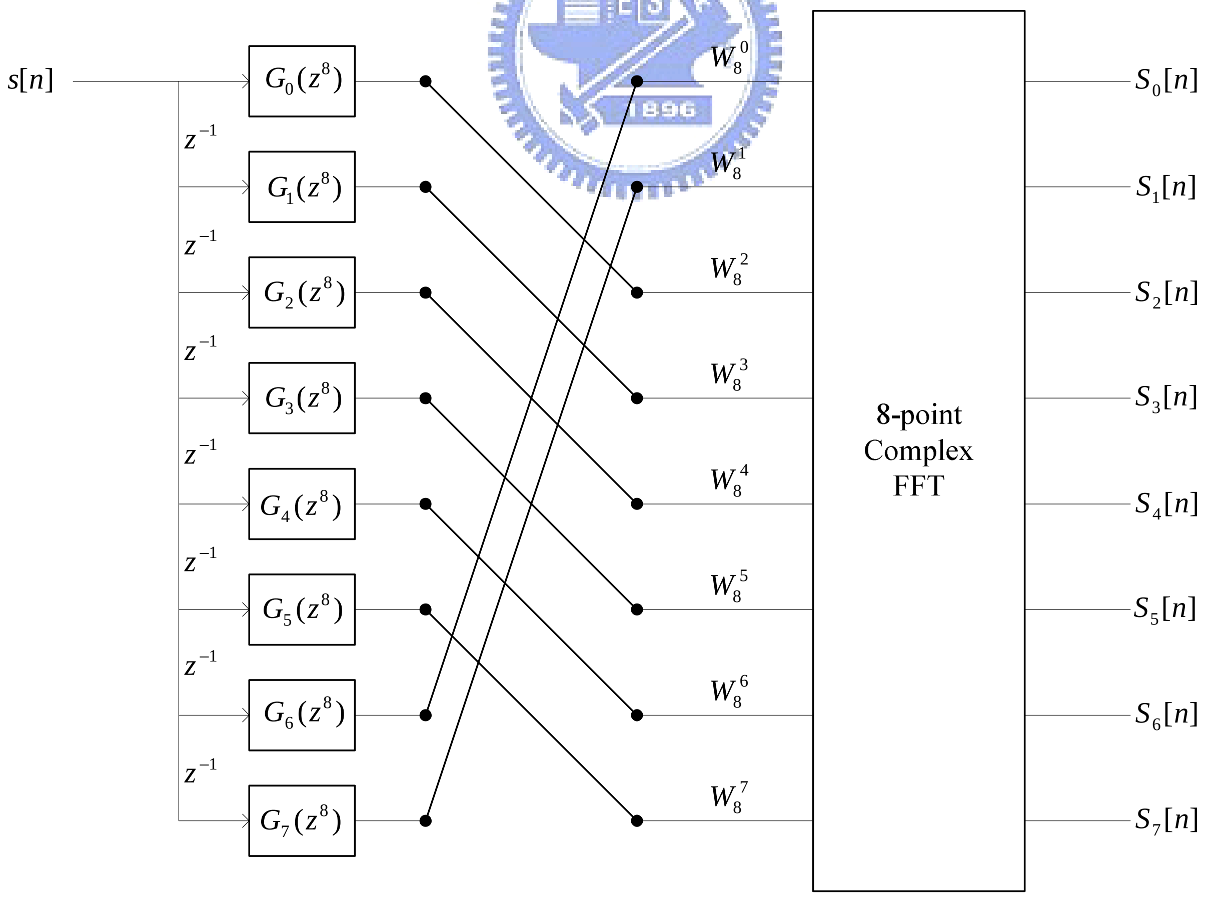


Figure 5.9: Signal flow graph of the fast analysis hybrid filterbank

5.6.2 Analysis Hybrid Filterbank Implementation Results

Table 5.10 shows the implementation results of the Analysis Hybrid filterbank. By using fast algorithms, the reduction ratio of the execution cycles is about 77% in the analysis hybrid filterbank.

Table 5.10: The acceleration result of analysis hybrid filterbank

Test Sequence	Original Analysis Hybrid FB (cycles)	Accelerated Analysis Hybrid FB (cycles)	Reduction ratio (%)
Choir	14,947,777	3,404,386	77.22
ts30	14,940,437	3,405,809	77.20
approaching_tunnel	14,946,461	3,402,930	77.23
carneval	14,940,967	3,397,464	77.26
fountain_music	14,948,398	3,405,428	77.22

5.7 LFE Channel Acceleration

In a standard 5.1-channel audio playback system, the '.1' channel means a band-limited Low Frequency Effects (LFE) channel. In contrast to the main channels, the LFE channel delivers bass-only information and has no direct effect on the perceived directionality of the reproduced soundtrack. Figures 5.10 and 5.11 show the spectrums of the front-left channel and the LFE channel, respectively. We notice that the signal of the front-left channel almost covers the full bandwidth while that of the LFE channel only has a frequency range below 500 Hz. Hence, in order to accelerate the encoder, we simplify the calculations associated with the LFE channel.

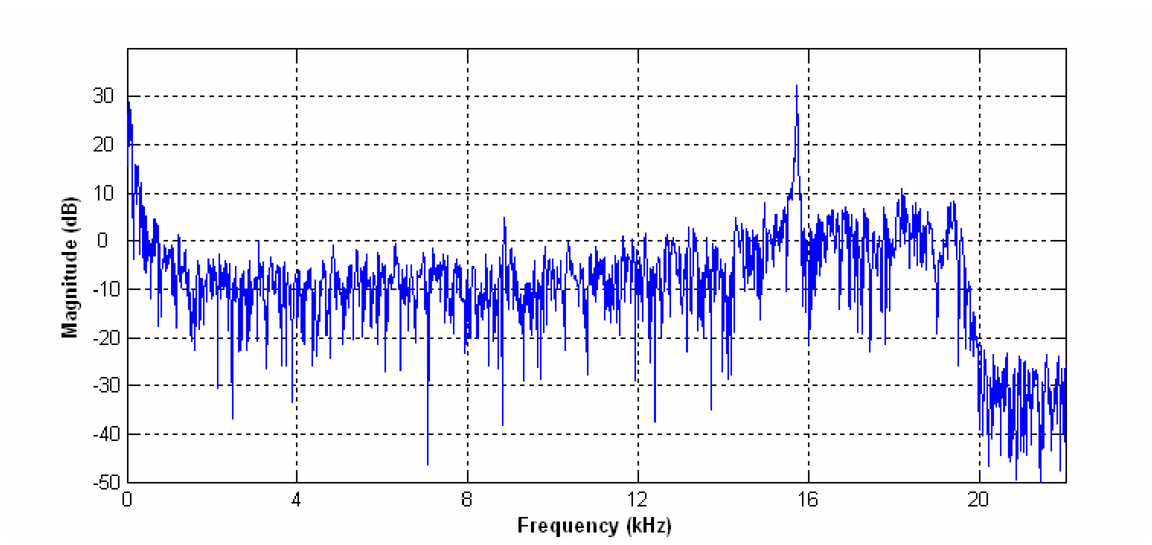


Figure 5.10: The spectrum of the front-left channel signal

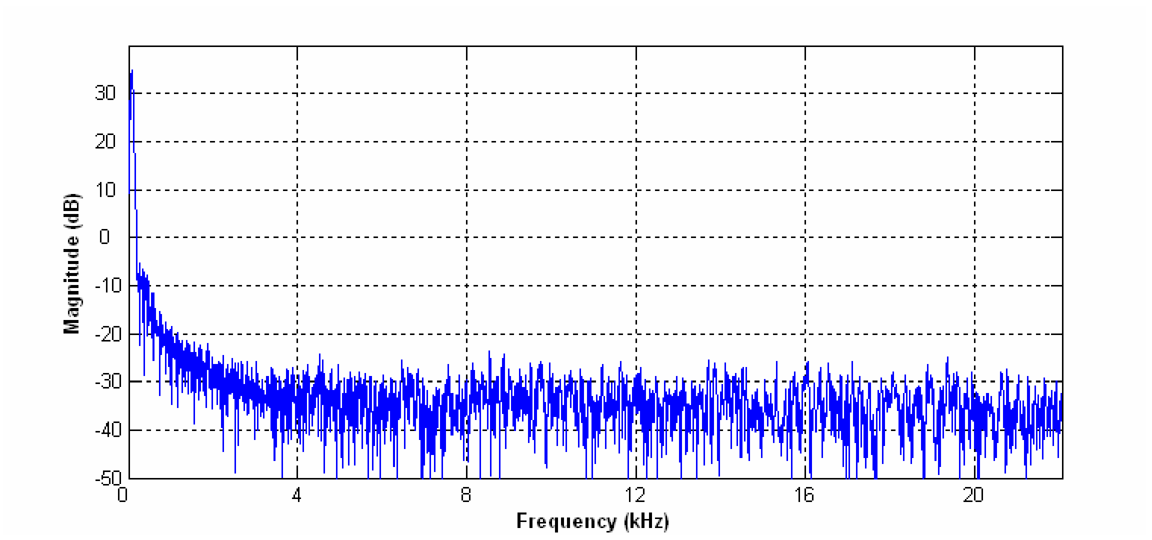


Figure 5.11: The spectrum of the LFE channel signal

We analyze the output signal of the analysis QMF bank, and find that the 0th QMF subband contains most of energy in the LFE channel. So we adopt a simplified QMF bank for the LFE channel, in which only the output signals in the lowest band will be computed. The outputs in the 0th band are calculated by simply applying a FIR filter with the QMF coefficients. As for the other subbands (1st to 63rd bands), the output signals are directly assigned to zero.

In the analysis hybrid filterbank, the frequency resolution of the lowest three QMF bands is increased. For the LFE channel, only the 0th band contains information, so the sub-filters of the 1st and 2nd bands can be removed. Figure 5.12 shows the structure of the simplified analysis filterbanks for the LFE channel. Compared to the main channels, it only calculates the output coefficients corresponding to the QMF band 0.

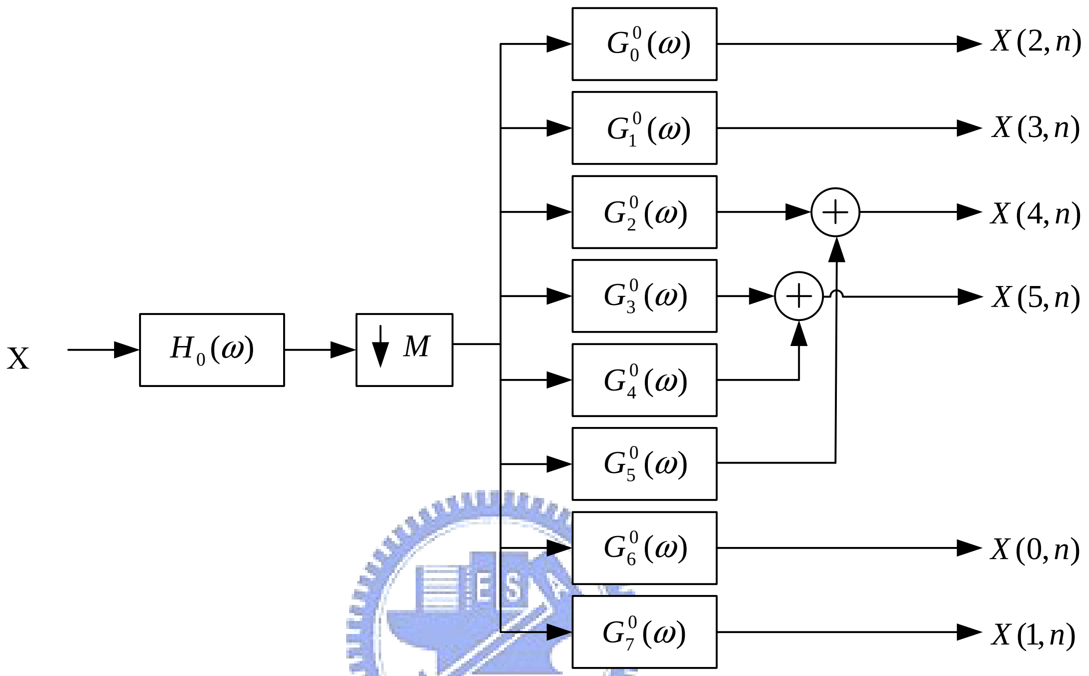


Figure 5.12: Simplified analysis filterbanks for the LFE channel

Table 5.11 shows the acceleration results for the LFE channel. The execution cycles in the filterbanks of the LFE channel are about 38% reduced.

Table 5.11: Acceleration result of LFE channel

Test Sequence	Original LFE (cycles)	Accelerated LFE (cycles)	Reduction ratio (%)
Choir	1,291,313	804,800	37.68
ts30	1,290,539	817,511	36.65
approaching_tunnel	1,292,189	799,473	38.13
carneval	1,296,264	799,474	38.32
fountain_music	1,295,359	799,475	38.28

5.8 Implementation of MPEG Surround-HEAAC Encoder

In the previous work, Huang has implemented the HEAAC encoder on the C64x DSP platform. The source code of the HEAAC is originally provided by 3GPP [31]. He adopted several acceleration methods to speed up the calculations on the DSP [32]. We like to extend Huang’s work by including the MPEG Surround encoder together with the HEAAC encoder. The combined MPEG Surround-HEAAC encoder is able to compress a 5.1-channel audio at low bitrates.

Figure 5.13 shows the structure of the interconnection of the MPEG Surround encoder and the HEAAC encoder. The AQMF bank, downsampling filter, AAC, and SBR tools are the modules related to HEAAC encoder (in the dotted box), and the others are related to MPEG Surround encoder. In the MPEG Surround encoder, the downmixed signal is transformed back to the time domain and fed in the HEAAC encoder. We will give a brief introduction for the HEAAC encoder.

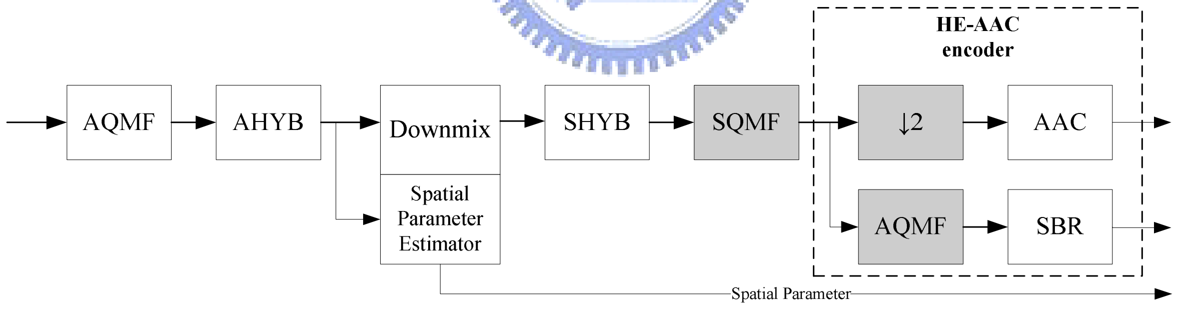


Figure 5.13: Interconnection of the MPEG Surround-HEAAC encoder

5.8.1 HEAAC Encoder

MPEG-4 HEAAC is a combination of Spectral Band Replication (SBR) tool and MPEG-2 AAC LC profile. The SBR system is used as a duel-rate system. The AAC encoder processes only the low frequency part of audio signals. It uses a downsampling filter to obtain the low frequency part of audio signals. The SBR encoder operates at the original sampling rate. It is devised based on the fact that there are usually high correlations between the lower and higher frequency parts of audio signals, so those correlations are coded as side-information and carried in the bitstream. The extraction of SBR side-information is processed in the 64-channel QMF domain, which is same as for the MPEG Surround. The details of MPEG-2 AAC and SBR can be found in [16] and [17], respectively.

5.8.2 Complexity Analysis

First, we connect the accelerated MPEG Surround encoder with the HEAAC encoder and profile the main modules for the MPEG Surround-HEAAC encoder on the C6416 simulator with L2 cache enabled. The result is shown on Table 5.12. The execution cycles of the MPEG Surround have about 12% of the overall system.

Table 5.12: Profiling of the MPEG Surround-HEAAC encoder

		Cycles	Percentage
Total		**91,943,427**	**100%**
MPEG Surround Encoder (Accelerated)	QMF Analysis	4,012,967	4.36%
	Hybrid Analysis	3,173,968	3.45%
	Ottbox	1,357,891	1.48%
	Hybrid Synthesis	302,233	0.33%
	QMF Synthesis	1,604,333	1.74%
HEAAC Encoder	Down-sampling Filter	20,513,525	22.31%
	QMF Analysis	19,915,569	21.66%
	SBR	19,741,829	21.47%
	AAC	17,745,833	19.30%
	others	3,575,279	3.89%

5.8.3 Simplified MPEG Surround-HEAAC Structure

We simplify the structure of the original MPEG Surround-HEAAC encoder. The structure is shown in Figure 5.14. As we know that the SBR is applied in the QMF domain, so it is unnecessary to transform the downmixed signal back to the time domain. Instead, we remove the SQMF and AQMF procedures before the SBR module. Furthermore, we combine the SQMF bank and downsampling filter to a downsampled SQMF bank, so the computational complexity can be further reduced. We will describe the downsampled SQMF bank below.

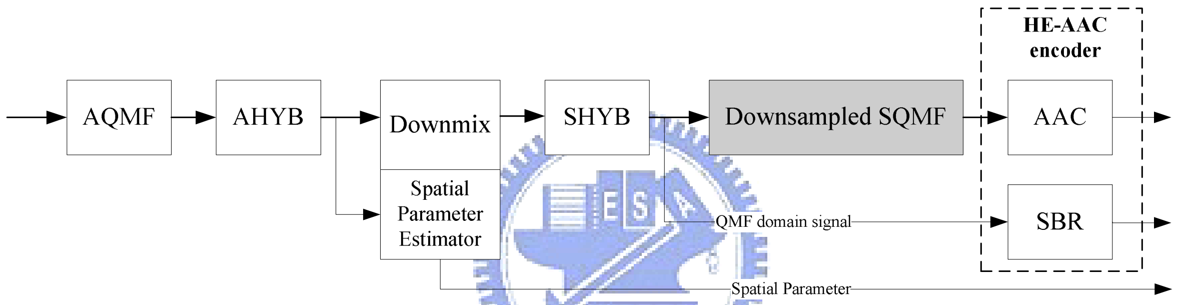


Figure 5.14: The structure of the simplified MPEG Surround-HEAAC encoder

The downsampled synthesis filtering is achieved using a 32-channel QMF bank. The output from the filterbank is real-valued time domain samples. The process comprises the following steps, where an array **v** consisting of 640 samples is assumed.

1. Shift the samples in the array **v** by 64 positions. The oldest 64 samples are discarded.
2. The 32 new complex-valued subband samples are multiplied by the matrix **N**

$$N(k,n) = \frac{1}{64}\exp\left(\frac{i\pi(k+0.5)(2n-127)}{64}\right), \begin{cases} 0 \le k < 32, \\ 0 \le n < 64. \end{cases}$$

   In the equation, exp() denotes the complex exponential function and i is the imaginary unit. The real part of the output from this operation is stored in the positions 0 to 63 of array **v**.

3. Extract samples from **v** according to

$$\left.\begin{array}{r} g[64\cdot n+k]=v[128\cdot n+k] \\ g[64\cdot n+32+k]=v[128\cdot n+96+k] \end{array}\right., \begin{cases} 0\le k<32 \\ 0\le n<5 \end{cases},$$

to create the 320-element array **g**.

4. Multiply the samples of array **g** by every other coefficient of the window **c** to produce array **w**. ($w[n]=g[n]\times c[2n]$, for n=0 to 319). The window coefficients of **c** are the same as for the analysis filterbank.
5. Calculate 32 new output samples by summing up the samples from array **w** according to the formula, $nextOutputAudioSample[k]=\sum_{n=0}^{9} w[32\cdot n+k]$, k=0 to 31.

We also notice the matrix operation in the step 2 is very similar to that in the AQMF bank. Hence, we adopt the same decomposition methods based on DCT-IV and DST-IV. The result is shown on Table 5.13.

Table 5.13: Reduction ratio of using downsampled SQMF bank

	Original (cycles)	Downsampled SQMF (cycles)	Reduction ratio (%)
SQMF	1,604,333	518,828	98.77
Downsampling	20,513,525		
AQMF(HEAAC)	19,915,569		

5.9 Experiments and Acceleration Results

Several experiments verifying the above acceleration methods are presented in the following. The proposed MPEG Surround encoder and MPEG Surround-HEAAC encoder are implemented on the 32-bit fixed-point C6416T DSP processor.

5.9.1 MPEG Surround Encoder (without HEAAC)

After accelerating the codes and modifying the algorithms, we have significantly reduced the computation load of the MPEG Surround encoder on DSP. We simulate with the C6416 DSP simulator to profile the cycles before and after acceleration of the MPEG Surround encoder. The optimization level is set to -o3 (file level) and the L2 cache is enabled. Table 5.14 shows the final results. We can see that after the fixed-point conversion, the overall speed is about 150 times faster. And after algorithms modification, the final result achieves 511 times faster than the original one. Table 5.15 shows the profile of the proposed MPEG Surround encoder. Compare to the original encoder, the execution cycles of the analysis QMF bank are reduced from 54% to 28% of the total execution time.

Table 5.14: The final acceleration result of the MPEG Surround encoder

		Total Execution Cycles	
Choir	Original	7,046,624,226	N/A
	Fixed-point	47,155,001	149.4x
	Final	13,781,607	511.3x
ts30	Original	7,046,531,873	N/A
	Fixed-point	46,865,520	150.4x
	Final	13,635,426	516.8x
approaching _tunnel	Original	7,046,783,840	N/A
	Fixed-point	47,153,413	149.4x
	Final	13,720,465	513.6x
carneval	Original	7,049,507,045	N/A
	Fixed-point	47,169,205	149.5x
	Final	13,733,887	513.3x
fountain _music	Original	7,049,503,069	N/A
	Fixed-point	47,167,059	149.5x
	Final	13,732,140	513.4x

Table 5.15: Profile of the proposed MPEG Surround encoder on C6416 simulator

	Original (float)		Fixed-point		Final	
Function	Cycles	Percent	Cycles	Percent	Cycles	Percent
Total cycles	**7,046,624,226**	**100%**	**47,155,001**	**100%**	**13,781,607**	**100%**
Initialization	553,634,278	7.86%	256,399	0.54%	283,557	2.06%
QMF Analysis	4,315,174,856	61.24%	20,966,993	44.46%	3,900,636	28.30%
Hybrid Analysis	1,137,277,617	16.14%	14,947,777	31.70%	3,170,837	23.01%
Ottbox	66,845,629	0.95%	1,352,341	2.87%	1,342,935	9.74%
Hybrid Synthesis	525,237	0.01%	294,197	0.62%	299,537	2.17%
QMF Synthesis	947,689,855	13.45%	6,136,115	13.01%	1,604,106	11.64%
others	25,476,754	0.36%	3,201,179	6.79%	3,179,999	23.07%

5.9.2 MPEG Surround-HEAAC Encoder

We also implement the combination of the accelerated MPEG Surround encoder and the HEAAC encoder on the TI DSP platform, and propose a simplified structure to speed up. Table 5.16 shows the comparison between the original and the simplified structures. We can notice that the speed of the simplified encoder is about 1.83 times faster than the original one. Table 5.17 shows the profile of the simplified MPEG Surround-HEAAC encoder. The MPEG Surround part occupies about 18 percent in the final system.

Table 5.16: Acceleration result of the MPEG Surround-HEAAC encoder

		Total Execution Cycles	
Choir	Original	91,943,427	N/A
	Simplified	50,313,874	1.83x
ts30	Original	90,976,562	N/A
	Final	50,213,164	1.81x
approaching _tunnel	Original	92,840,027	N/A
	Simplified	51,832,299	1.79x
carneval	Original	91,117,098	N/A
	Simplified	49,994,663	1.82x

fountain _music	Original	90,798,366	N/A
	Simplified	49,043,238	1.85x

Table 5.17: Profiling of the proposed MPEG Surround-HEAAC encoder on C6416 simulator

	Original SAC-HEAAC		Simplified SAC-HEAAC	
Function	Cycles	Percent	Cycles	Percent
Total cycles	**91,943,427**	**100%**	**50,313,874**	**100%**
QMF Analysis	4,012,967	4.36%	4,052,749	8.05%
Hybrid Analysis	3,173,968	3.45%	3,009,096	5.98%
Ottbox	1,357,891	1.48%	1,338,743	2.66%
Hybrid Synthesis	302,233	0.33%	236,327	0.46%
QMF Synthesis	1,604,333	1.74%	Down-sampled QMF 518,828	1.03%
Down-sampling Filter	20,513,525	22.31%		
QMF Analysis	19,915,569	21.66%		
SBR	19,741,829	21.47%	20,000,178	39.75%
AAC	17,745,833	19.30%	17,971,943	35.71%
others	3,575,279	3.89%	3,186,010	6.33%

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The main goal to this thesis is to implement and accelerate the MPEG Surround encoder on the TI C6416T DSP processor. We have efficiently reduced the complexities of the MPEG Surround encoder. First, we overcame the memory bottleneck by setting up properly the L2 cache on the DSP platform. We also converted the floating-point codes to the fixed-point, which match the DSP hardware architecture.

We adopt several acceleration methods to speed up the reference encoder. These methods are fast QMF bank algorithms, fast hybrid filterbank algorithms, and LFE channel acceleration. In QMF bank, we decomposed the matrix operation in the QMF procedure into DCT-IV/DST-IV with a small amount of additional permutations. And the computation of N-point DCT-IV and DST-IV can be further reduced by using the N/2-point complex FFT. For the hybrid filterbank, we proposed a fast algorithm based on FFT algorithm. In addition, we used the optimized FFT functions in DSPLIB, which is faster than the C-coded FFT. For the LFE channel, we observed that the 0th QMF subband contains most of energy. Hence, we only calculated the output signal in the 0th QMF band to reduce the computations. By employing these techniques, the proposed MPEG Surround encoder is 513 times faster than the original version.

Furthermore, we connected the MPEG Surround encoder together with the HEAAC encoder, and simplified the structure by using the downsampled SQMF bank. The speed of the simplified encoder is about 1.82 times faster than the previously optimized structure.

6.2 Future Work

Our thesis mainly focuses on the speed of the overall system. However, the MPEG Surround reference encoder released by ISO/IEC do not optimize for the audio quality. For example, it uses very simple algorithms to perform the downmix procedure and extract the spatial parameters. Besides, we notice that some tools in the MPEG Surround standard, such as residual coding, guided envelope shaping (GES), and 3D stereo, are not implemented in the reference encoder. Therefore, proposing new algorithms for enhancing the audio quality of the MPEG Surround codec can be a useful and challenging task.

Also, the TI DSP board contains an FPGA. We can integrate the FPGA implementation together with DSP to accelerate the overall system. But the data transfer between DSP and FPGA is quite complex, and we are unable to use it yet.

References

[1] J. Blauert, *Spatial Hearing: The Psychophysics of Human Sound Localization*, revised edition, Cambridge, 1997.

[2] R. I. Chernyak and N. A. Dubrovsky, "Pattern of the noise images and the binaural summation of loudness for the different interaural correlation of noise", *Proceedings of the 6th International Congress on Acoustic*, Tokyo, pp.A53-A56, 1968.

[3] J. Herre and et al., "Intensity stereo coding", *96th Audio Engineering Society (AES) Convention*, Amsterdam, 1994.

[4] M. Bosi and et al., "ISO/IEC MPEG-2 advanced audio coding", Journal of the AES, vol.45, no. 10, pp. 789-814, Oct. 1997.

[5] J. Breebaart and et al., "Parametric coding of stereo audio", *EURASIP Journal on Applied Signal Proc.*, pp. 1305-1322, Sep. 2005.

[6] H. Purnhagen, "Low complexity parametric stereo coding in MPEG-4", *Proc. of the 7th International Conference on Audio Effects (DAFx'04)*, Italy, Oct. 2004.

[7] F. Baumgarte and C. Faller, "Binaural cue coding - part I: psychoacoustic fundamentals and design principles, *IEEE Transactions on Speech and Audio Proc.*, vol. 11, no. 6, pp. 509-519, Nov. 2003.

[8] C. Faller and F. Baumgarte, "Binaural cue coding – part II: schemes and application", *IEEE Transactions on Speech and Audio Proc.*, vol. 11, no. 6, pp. 520-531, Nov. 2003.

[9] ISO/IEC JTC1/SC29/WG11 (MPEG), Document N6455, "Call for proposals on spatial audio coding", Munich, March 2004.

[10] ISO/IEC JTC1/SC29/WG11 (MPEG), Document N8324, "Text of ISO/IEC FDIS 23003-1, MPEG Surround", Klagenfurt, 2006.

[11] J. Breebaart and et al., "High-quality parametric spatial audio coding at low bit rates", *116th Audio Engineering Society (AES) Convention*, Germany, May 2004.

[12] J. Herre and et al., "The reference model architecture for MPEG spatial audio coding", *118th Audio Engineering Society (AES) Convention*, Spain, May 2005.

[13] J. Breebaart and et al., "MPEG spatial audio coding / MPEG Surround: overview and current status", *119th Audio Engineering Society (AES) Convention*, New York USA, Oct. 2005.

[14] L. Villemoes and et al., "MPEG Surround: the forthcoming ISO standard for spatial audio coding", *AES 28th International Conference*, Sweden, 2006.

[15] ISO/IEC JTC1/SC29/WG11 (MPEG), Document N7390, "Tutorial on MPEG Surround audio coding", Poland, July 2005.

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

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

[18] Sundance DSP System, "SMT 395 user manual", May 2001.

[19] Texas Instruments, "TMS320C6414T, TMS320C6415T, TMS320C6416T fixed-point digital signal processors", Literature number SPRU226H, Nov. 2003.

[20] Texas Instruments, "TMS320C6000 programmer's guide", Literature number SPRU198F, Feb. 2001.

[21] Texas Instruments, "TMS320C64x DSP two-level internal memory reference guide", Literature number SPRU610B, Aug. 2004.

[22] Texas Instruments, "TMS320C6000 optimizing compiler v6.0 beta user's guide", Literature number SPRU187N, July 2005.

[23] Texas Instruments, "TMS320C6000 CPU and instruction set reference gudie", Literature number SPRU189F, Jan. 2000.

[24] Texas Instruments, "TMS320C64x technical overview", Literature number SPRU395B, Jan. 2001.

[25] ISO/IEC JTC1/SC29/WG11 (MPEG), Document N8636, "ISO/IEC 23003-1:2006/PDAM2, MPEG Surround reference software", Oct. 2006.

[26] W. H. Chen, C. H. Smith, S. C. Fralick, "A fast computational algorithm for the discrete cosine transform", *IEEE Trans. Commun.*, vol. 25, no. 9, pp. 1004-1008, Sep. 1977.

[27] B. G. Lee, "A new algorithm to compute the discrete cosine transform", *IEEE Transaction on Acoustic, Speech, and Signal Proc.*, vol. 32, no. 6, pp. 1243-1245, Dec. 1984.

[28] R. Gluth, "Regular FFT-related transform kernels for DCT/DST-based polyphase filter banks", *IEEE ICASSP*, 1991.

[29] G. Bonnerot and M. Bellanger, "Odd-time odd-frequency discrete Fourier transform for symmetric real-valued series", *IEEE Proceedings*, vol. 64, pp. 392-393, March 1976.

[30] Texas Instruments, "TMS320C64x DSP library programmer's reference", Literature number SPRU565B, Oct. 2003.

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

[32] Y. -C. Huang, "MPEG-4 High Efficient AAC codec acceleration and implementation on TI DSP", M.S. thesis, Department of Electrical and Computer Engineering, National Chiao Tung University, HsinChu, Taiwan, ROC, 2006.

Appendix A

N/2-point FFT Algorithm for DCT-IV and DST-IV

We will describe the N/2-point FFT in detail in this appendix. We will show the mathematical derivation of the algorithm. The details can be found in [28].

The N-point DCT-IV and DST-IV are defined as

$$DCT_{[N]}^{IV} : C_k^{IV} = \sum_{r=0}^{N-1} x_r \cos\left(\frac{\pi}{4N}(2k+1)(2r+1) \right) \tag{A.1}$$

$$DST_{[N]}^{IV} : S_k^{IV} = \sum_{r=0}^{N-1} x_r \sin\left(\frac{\pi}{4N}(2k+1)(2r+1) \right) \tag{A.2}$$

A.1 Relationship to O^{2}DFT

In order to find their relationship to Fourier Transform, we introduce the odd-time odd-frequency discrete Fourier transform (O^{2}DFT) of length K, and which is defined as

$$O^2DFT_{[K]k}\{\underline{u}\} = U_k = \sum_{r=0}^{K-1} u_r \exp\left(-j\frac{\pi}{2K}(2k+1)(2r+1) \right) \tag{A.3}$$

Using the 2N-point zero-padded vector $\underline{x}'$ given by

$$x_r' = \begin{cases} x_r & for \quad 0 \le r \le N-1 \\ 0 & for \quad N \le r \le 2N-1 \end{cases} \tag{A.4}$$

The DCT-IV and the DST-IV of $\underline{x}$ are real and imaginary part of the $O^2DFT_{[2N]}$ of $\underline{x}'$ according to

$$C_k^{IV} = \mathrm{Re}\left\{ O^2DFT_{[2N]k}\{\underline{x}'\} \right\}, \tag{A.5}$$

$$S_k^{IV} = \mathrm{Im}\left\{ O^2DFT_{[2N]k}\{\underline{x}'\} \right\}, \quad k \in \{0,\ldots,N-1\} \tag{A.6}$$

Rewriting the zero-padded vector $\underline{x}'$ as the sum of the odd extension, $\underline{w}$, and the even extension, $\underline{v}$, of $\underline{x}$ according to

$$\underline{x}' = \tfrac{1}{2}(\underline{w} + \underline{v})$$

with

$$\begin{array}{lll} 0 \le r \le N-1: & v_r = x_r; & w_r = x_r \\ N \le r \le 2N-1: & v_r = x_{2N-1-r}; & w_r = -x_{2N-1-r} \end{array} \tag{A.7}$$

By applying the symmetry properties of the O^{2}DFT, equations (A.5) and (A.6) are simplified

$$C_k^{IV} = \tfrac{1}{2} W_k = \tfrac{1}{2} O^2 DFT_{[2N]k}\{\underline{w}\} \tag{A.8}$$

$$S_k^{IV} = -j \tfrac{1}{2} V_k = \tfrac{1}{2} \mathrm{Im}\{O^2 DFT_{[2N]k}\{\underline{v}\}\} \tag{A.9}$$

This provides the solution for an efficient realization of equation DCT-IV and DST-IV, because there is a fast algorithm for the calculation of the O^{2}DFT of real symmetric sequences of even length. We will introduce this method below [29].

A.2 Fast Algorithm for Symmetric O^{2}DFT

The equation for a fast O^{2}DFT of length K exploiting these symmetries is:

$$\begin{aligned} P_k &= U_{2k} + jU_{K/2+2k} \\ &= 2 \sum_{r=0}^{K/4-1} (u_{2r} - ju_{K/2+2r}) e^{-j\frac{2\pi}{K}(2k+\frac{1}{2})(2r+\frac{1}{2})} \\ &= 2e^{-j\frac{2\pi}{K}(k+\frac{1}{8})} \underbrace{\sum_{r=0}^{K/4-1} \left\{ \{u_r' e^{-j\frac{2\pi}{K}(r+\frac{1}{8})}\} e^{-j\frac{2\pi}{K/4}rk} \right\}}_{K/4-\text{point } FFT} \end{aligned} \tag{A.10}$$

where

$$\begin{aligned} u_r' &= (u_{2r} - ju_{K/2+2r}) \\ r,k &= 0,1,\ldots,\tfrac{K}{4}-1 \end{aligned} \tag{A.11}$$

(A.10) is in the form of a modified complex FFT.

$\underline{W}$, the transform of real odd vectors is retained from $\underline{P}$ according to

$$W_{2k} = \mathrm{Re}\{P_k\}; \quad W_{K/2+2k} = \mathrm{Im}\{P_k\} \qquad \text{(A.12)}$$

and the separation to get $\underline{V}$ corresponding to an even input is given by

$$V_{2k} = j\,\mathrm{Im}\{P_k\}; \quad V_{K/2+2k} = -j\,\mathrm{Re}\{P_k\} \qquad \text{(A.13)}$$

$\underline{P}$ contains all information to retrieve all coefficients using the symmetry properties.

In summery, the N-point DCT-IV and DST-IV can be computed by applying 2N-point O^{2}DFT to real symmetric sequences, and it can be obtained by using a standard N/2-point complex DFT.

自傳

韓志岡，西元 1983 年 5 月 13 日出生於台北市。2005 年畢業於國立交通大學電信工程學系。同年進入交通大學電機資訊學院電子工程研究所電路與系統組碩士班，於 2007 年取得碩士學位，論文題目為「環繞 MPEG 編解碼器之增速及其在 TI DSP 平台上的實現」。研究範圍與趣包括：數位信號處理、多媒體通訊、音訊壓縮。