A MODIFIED LOW-BIT-RATE ACELP SPEECH CODER AND ITS IMPLEMENTATION

JIAN GUANG ZHOU

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements for the Degree of Master of Applied Science at Concordia University
Montreal, Quebec, Canada

November, 2003

Your file Votre référence
ISBN: 0-612-91162-4
Our file Notre référence
ISBN: 0-612-91162-4

ABSTRACT

A MODIFIED LOW-BIT-RATE ACELP SPEECH CODER AND ITS IMPLEMENTATION

Jian Guang Zhou

In modern communication systems, low-bit-rate speech coder is widely employed to increase the bandwidth efficiency of the network. The ITU-T G.729 is one such speech coder extensively used in packet voice communication systems. Since the reduction in the complexity has been a major issue in designing and implementing low-bit-rate speech coder, the focus of this research is on reducing the complexity of the existing G.729 standard. Based on the fact that the fixed codebook search accounts for much of the computation in the coding process, in this thesis, the existing codebook search algorithm of G.729A, a reduced complexity version of G.729, is modified and incorporated into this standard. The test results on a general PC indicate that the modified search scheme reduces the computation load by about 40% with only a slight degradation in the perceptual quality of the speech.

In the second part of this project, the modified speech coder is implemented on the TMS320C5416 DSK DSP board. First, the implementation scheme is described and the initial results for both the G.729A and the modified coder are presented. Then, several optimization procedures are developed to effectively reduce the run-time of the algorithm. With the optimized implementation, the run-time of the speech coder is significantly reduced to meet the real time requirement to process a single frame within 10 ms. As for the savings on the computational load, the run-time of the modified fixed codebook search algorithm is about 30% smaller than that of the original search scheme used in G.729A.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis supervisors, Dr. M. N. S. Swamy and Dr. M. O. Ahmad, for their guidance and support throughout the course of this research work.

I am grateful to the colleagues of our research group, particularly, Mr. Wei Chu, Dr. Ke Li, and Mr. Jing Liang, for their continuous support, suggestions and inspirations during this research.

I would like to thank Nortel Networks and Micronet, a National Network of Centers of Excellence, Canada, for their financial support given to my supervisors.

Last but not least, many thanks are due to my family, especially to my wife, Wei Kuang, for her inspiration and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACELP	algebraic code-excited-linear-prediction
ALU	arithmetic logic unit
API	application program interface
AWGN	additive white Gaussian noise
CELP	code-excited-linear-prediction
COFF	common object file format
CSSU	compare, select, store unit
DARAM	dual access RAM
DSK	DSP starter's kit
ETSI	Europe Telecommunication Standard Institution
G.729A	ITU-T G.729 Annex A
IDATA	internal data
IPROG	internal program
ISPP	interleaved single pulse permutation
LD-CELP	low-delay CELP
LPC	linear prediction coding
LSP	line spectral pair
MELP	mix-excited-linear-prediction
MIPS	million instruction per second
MMSE	minimum mean-squared error
MOS	mean opinion score

OVLY	overlay
PCM	pulse code modulation
PESQ	perceptual evaluation of speech quality
SEGSNR	segment SNR
STS	statistics
SWI	software interrupt
TIA	telecommunication industry association
VAD	voice activity detection
WMOPS	weighted million operations per second

Chapter 1

Introduction

1.1 General

In modern digital systems, speech signal is represented in digital format and it is often desirable to represent the signal using as few bits as possible. The purpose of speech coding is to reduce the bit rate of digital speech signals. For storage applications, a lower bit usage means less memory. For transmission applications, lower bit rate means less bandwidth, power and memory. It is, therefore, cost-effective to use an efficient speech compression algorithm in a digital speech storage or transmission system. Speech coding is a technique to provide such a compression. Although larger bandwidth has become available in wired communications because of the rapid development in optical transmission, there still is a growing demand for bandwidth conservation, particularly in wireless and satellite communication systems. Speech coding plays a vital role in wireless and Internet transmission. A speech coder consists of an encoder and a decoder. The encoder takes the original digital speech signal and produces a low-rate bit-stream. This bit-stream is the input to the decoder, which recovers an approximation of the original signal. Figure 1.1 shows the structure of the speech coder in a communication system. There are many attributes associated with a speech coder [1]. Bit rate, speech quality, delay, complexity and channel error sensitivity are the fundamental characteristics of a speech coder. The criteria for selecting the appropriate bit rate for transmission depend on the cost of transmission, the cost of coding, and the speech quality requirements. Some twenty years ago, speech coding was not commonly used because of the high cost of coding and the low speech quality. However, now because of a dramatic increase in the efficiency of digital speech processing hardware and the recent advances in speech coding research, this situation has significantly changed, and speech coding is employed in a large number of applications.

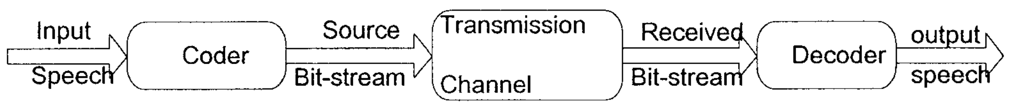


Figure 1.1 Block diagram of digital speech transmission

This chapter is organized as follows. Section 1.2 gives an overview of the basic attributes of a speech coder. In Section 1.3, we introduce some basic properties of speech signals for a better understanding of the speech coders. Section 1.4 gives a brief review of speech coding standards. Section 1.5 provides the details of the thesis organization.

1.2 Attributes of the Speech Coder

There are many factors that should be considered when evaluating the performance of a speech coder. Some major attributes of a speech coder are now considered.

(i) Bit-rate

An essential motivation for speech coding is to reduce the bit-rate of the speech coder. The bit-rate of a speech coder can be lower than 1 Kb/s for a vocoder and can be as high as 64 Kb/s for a waveform coder. Depending on application, there are fixed-rate or variable-rate speech coders. Most of the existing speech coders are fixed-rate coders, since they are simpler to design. Coders used for satellite and cellular telephony range from 3.3 to 13 Kb/s, whereas those used for general telephone networks have bit rates of more than 16 Kb/s. For a secure telephone communication, the bit rates are between 0.8 to 4.8 Kb/s [2]. There are also many applications for variable-rate coders. The ITU-T G.723.1 standard is an example of a variable rate coder [3].

(ii) Speech Quality

The quality of the reconstructed speech signal is an important attribute of a speech coder. The "mean opinion score" (MOS) is the most commonly used measure for the subjective quality of the coded speech. The MOS is extracted from the results of a category test performed by 20 to 60 untrained listeners. The listeners give a score for each of a set of utterances on a scale from 1 (for unacceptable quality) to 5 (for excellent quality). Because of a wide variation among listeners, the MOS test requires a large number of speech data, speakers and listeners to get an accurate rating of a speech coder. A MOS score of 4.0 or higher defines toll quality, and the reconstructed speech signal is nearly indistinguishable from the original signal. A MOS score between 3.5 and 4.0 defines communication quality, which is sufficient for natural telephone communication. At a score of 3.0 or lower, the reconstructed speech may be intelligible, but often lacks naturalness and speaker recognizability. It is a difficult problem to have an objective evaluation of speech quality for a variety of speech coders and input signals. This is because the quality of reconstructed signal of coders becomes more and more dependent on the characteristics of the input signal, making it difficult to anticipate the behavior of a coder in real-world applications. However, recently a standard objective method has been developed to estimate the speech quality [4], and this objective criterion is used in our project to estimate the quality of the decoded speech signal using our modified codebook search algorithm. Another objective measurement for the speech quality is segment SNR, this method is most suitable for waveform coders and it is not particularly suitable for the evaluation of vocoders.

(iii) Delay

Delay is an issue that is of importance for a two-way communication. A delay of 150 ms can be considered as impairment for highly interactive conversational tasks. However, if the echo canceller equipment is not used, the network echoes can be objectionable even when the two-way delay is less than 100 ms. Coder delay includes four components: algorithmic delay, computational delay, multiplexing delay and transmission delay. Speech coders often operate on a block-by-block basis, each block being called a frame. One frame of data must be collected before processing begins. Often the coder requires some additional look-ahead beyond the frame that is to be encoded. The algorithm delay is the sum of the frame length and the look-ahead. Computational delay is the actual processing time required for the coder; in most of the implementations, the processing delay is equal to a frame length or slightly less. In many of the transmission systems, a block of bits corresponding to a frame is first assembled by the coder before it is transmitted, while at the receiver, the block of bits associated with a frame is assembled before decoding begins. This assembling delay, whether it is at the transmitter or the receiver end, is called the multiplexing delay. The transmission delay is usually less than a frame size. Often we assume that multiplexing delay and transmission delay adds up to one frame length. So roughly speaking, the minimum delay of the system is usually between 3 and 4 times the frame size.

(iv) Complexity

The computational complexity of a speech coder determines the cost and power consumption of the hardware used for it's implementation. Most speech coders are implemented on DSP chips. Speed and random access memory (RAM) usage are the two most important contributors to complexity. The faster the chip or the larger the chip size, the greater the cost. These same attributes also influence the power consumption, which is also a critical attribute for many handheld applications. Thus, complexity is a determining factor for both the cost and the power consumption. Speed is most commonly measured as the number of millions of instructions per second (MIPS) necessary for the real-time implementation of the speech coding algorithm. Sixteen-bit fixed point DSPs are most commonly used for low cost implementations. Hence, the complexity is often specified in terms of the fixed-point MIPS and the number of 16-bit words of RAM needed for an implementation. In our project, the speech coder is implemented using a 16-bit fixed-point DSP chip and evaluate the reduction of the complexity using the modified codebook search algorithm. Reducing the complexity of the speech coder for the real-time application is a critical concern for most network operators. Thus, it is also a major motivation for us to focus the research on this aspect.

(v) Channel Error Sensitivity

In wireless applications, the bit stream received at the other end of channel is always corrupted by channel errors. Thus, there should be a packet loss recovery mechanism in the decoder. There are two types of errors in the channel: random error and burst error. For the random error, each transmitted bit has the same probability of error; and the overall error rate is usually between 1% and 5%. Burst error is typical in the wireless environment. To counter random errors, a sufficient signal-to-noise ratio must be achieved. In many existing communication systems, the speech coder is separate from the channel coder. In such systems, the transmitted bits are usually classified into groups with different sensitivities to channel errors, and these groups are protected at different levels. In ITU-T G.729 [5], a parity bit is inserted in the encoded parameter for error detection, and in the decoder, there is an error concealment mechanism for the recovery of frame erasures. However, in our work we assume transparent transmission through the channels. Table 1.1 lists some of the basic attributes of different speech coders [1].

TABLE 1.1 SOME OF THE EXISTING SPEECH CODERS

Standard	Coding type	Bit Rate	MOS	Complexity	Delay (ms)
G.711	PCM	64	4.3	1	0.125
G.726	ADPCM	32	4.0	10	0.125
G.728	LD-CELP	16	4.0	50	0.625
GSM	RPE-LTP	13	3.7	5	20
G.729	CS-ACELP	8	4.0	22	15
G.723.1	ACELP	5.3	3.8	25	37.5
	MP-MLQ	6.3			
FS1015	LPC-10	2.4	Synthetic	10	22.5

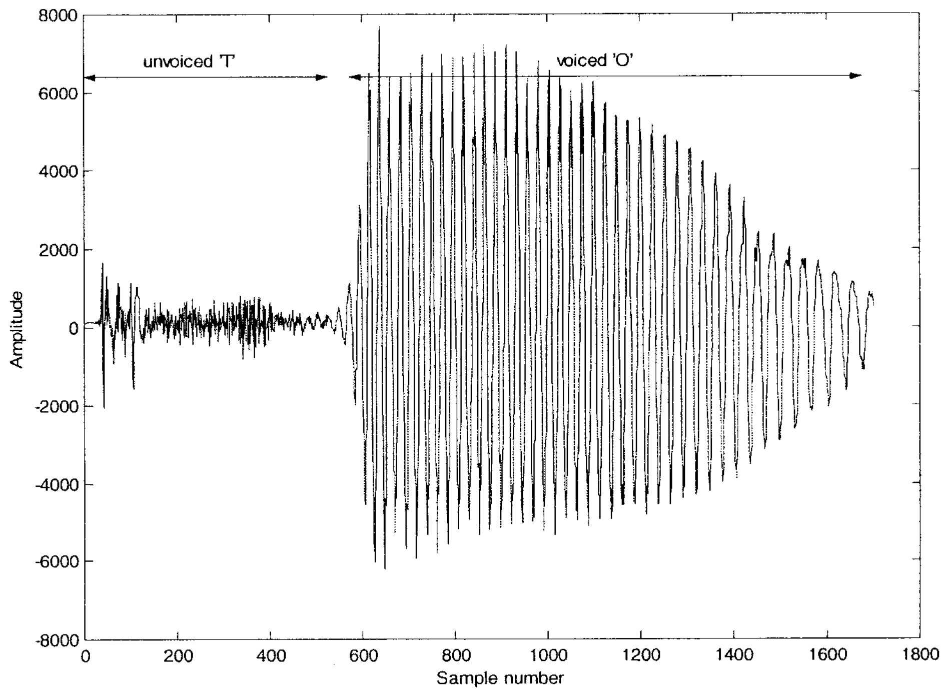
1.3 Properties of Speech Signals

In order to build an effective speech coder, an understanding of the properties of a speech signal is needed. Such an understanding leads to models that remove redundancy from a speech signal and transmit only the perceptually relevant information. The most prominent property of a speech signal is that it is band limited with a bandwidth of about 3.4 KHz. Hence it can be sampled at the Nyquist rate of 8 KHz, quantized and companded to either 8- or 16-bit pulse code modulation (PCM). Hence, an uncoded speech can be represented using a bit rate of 128 Kb/s, which is an extremely high bit rate. Specifically, for a signal with a bandwidth of 3.4 kHz and a SNR of 30 dB, assuming an additive white Gaussian noise (AWGN) channel, the bit rate C, based on Shannon's channel capacity theorem, is given by [2]

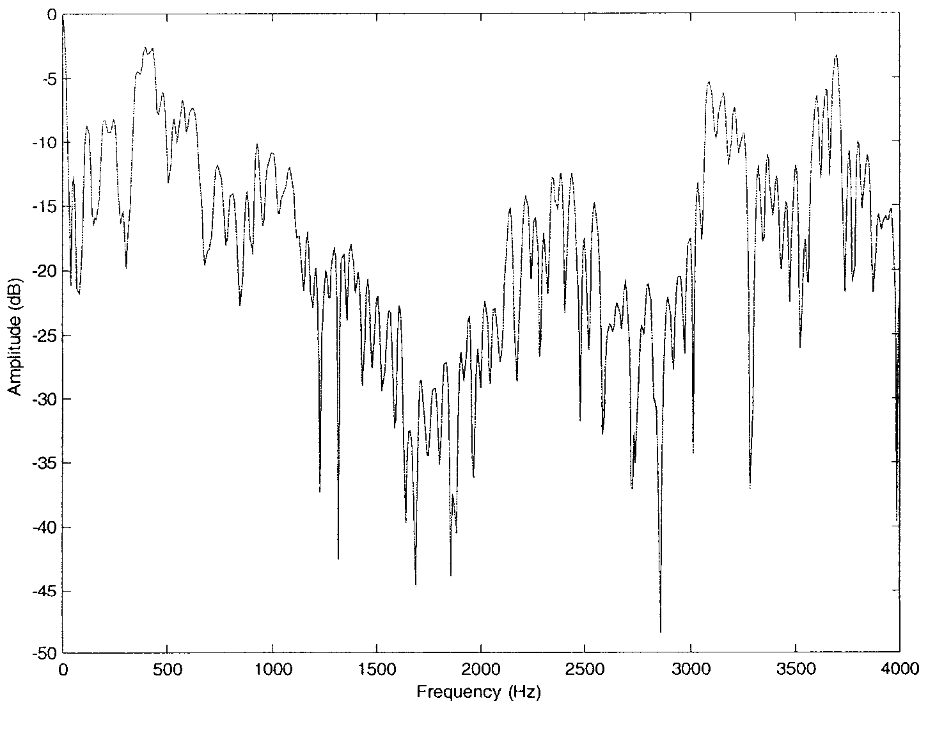
$$C = WLog_2[1 + P/G]$$

where W is the bandwidth and P/G is the SNR. The equation above shows that the bit rate required to represent speech with a small error is about 34 Kb/s. Although it is a reduction by a factor of 4, it is still too high for modern telecommunication systems.

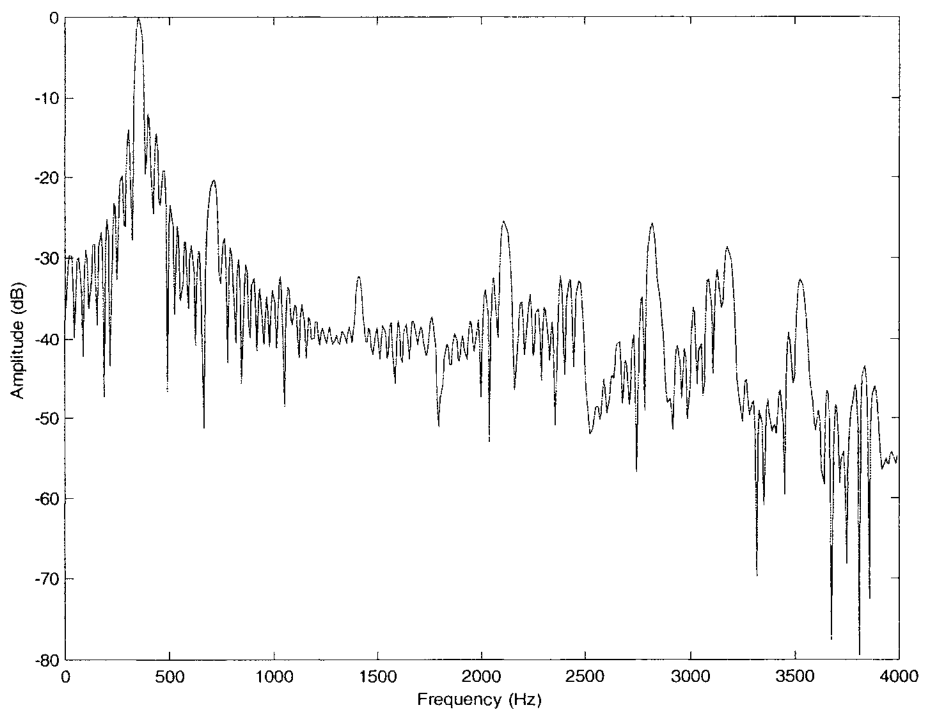
The Shannon limit of 34 Kb/s is an upper bound, since it does not take into account the redundancies in speech signals. Another important property of the speech signal is that there exists a correlation between adjacent samples; this is known as short-term correlation. Further, it is quasi-periodic, called long-term correlation [2]. By using techniques such as linear prediction and pitch prediction, the short-term and long-term redundancies can be removed to give an even lower bit rate. The time domain characteristics of the speech signal can be classified into unvoiced and voiced segments. Unvoiced segments are aperiodic and have a noise-like appearance. Figure 1.2 (a) demonstrates an example of unvoiced sound 'T' and voiced sound 'O'. The fact that the unvoiced segments are noise-like in their appearance suggests that we can replace it with a Gaussian noise source and the human ear will not be able to perceive the difference. Voiced sounds, on the other hand, are found to be periodic and have short and long term correlations. An example of the voiced speech is the sound of the vowels (/a/, /e/, /i/, /o/, /u/). The frequency domain characteristics of the speech signal lie in regions called formants. Formants correspond to the resonant frequencies of the vocal tract and usually it is below 4 KHz. It is also the region where most of the speech energy is concentrated. Figure 1.2 (b) illustrates an example of an unvoiced segment in the frequency domain, while Figure 1.2 (c) that of a voiced segment.


unvoiced 'T'
voiced 'O'
Amplitude
Sample number
8000
6000
4000
2000
0
-2000
-4000
-6000
-8000
0
200
400
600
800
1000
1200
1400
1600
1800

(a)


Amplitude (dB)
Frequency (Hz)
0
-5
-10
-15
-20
-25
-30
-35
-40
-45
-50
0
500
1000
1500
2000
2500
3000
3500
4000

(b)



(c)

Figure 1.2 Speech signal in time and frequency domain

(a) Voiced and unvoiced signal in the time domain; (b) Unvoiced signal "T" in the frequency domain; and (c) Voiced signal "O" in the frequency domain

The final important characteristic of a speech signal has to do with human perception. Exploiting the perceptual properties of the ear could lead to significant improvement in the performance of a speech coder. This is particularly true as we pursue lower and lower bit rate speech coders while avoiding major degradation in speech quality. One of the well-known properties of the auditory system is the auditory masking, which has a strong effect on the perceptibility of one signal in the presence of another [6]. Noise is less likely to be heard at frequencies of strong speech energy (e.g., speech formants) and more likely to be heard at frequencies of low speech energy (e.g., speech valleys). Spectral masking is a popular technique that takes advantage of this perceptual limitation by concentrating most of the noise in high-energy spectral regions where it is least audible. Humans perceive voiced and unvoiced sounds differently. In spectral domain, the amplitudes and the locations of the first three formants and the spacing between the harmonics are important [7]. For unvoiced signals, it has been shown in [8] that the unvoiced speech segments can be replaced by a noise-like signal with a similar spectral envelope without a drop in the perceived quality of the speech signal. In both the voiced and unvoiced cases, the time envelope of the speech signal contributes to intelligibility and naturalness [9,10].

1.4 Speech Coding Standards

In the past decade, more speech coding standards have been created than in all the previous years. The reasons for this are the maturity of speech coding technology and the need to satisfy a growing demand for new speech communication techniques. Standards exist because there are strong needs to have a common means for communication. Manufacturers, service providers, and customers all realize that it is in everyone's interest to have a common standard. The International Telecommunication Union (ITU) is responsible for setting global telecommunication standards. The European Telecommunications Standards Institute (ETSI) is responsible for setting standards for the digital cellular system in Europe. The Telecommunication Industry Association (TIA) makes standards for cellular telephony and other telecom applications in U.S. In 1994, ITU adopted the Low-Delay Code-Excited Linear Predictive (LD-CELP) algorithm [11] for the toll quality coding of speech at 16 Kb/s, and is known as ITU G.728. Shortly after this standard was adopted, another CELP-based speech coding running at 8 Kb/s was developed by the University of Sherbrooke [5]. It is toll quality as well and has a performance comparable to that of ITU-T G.726 at 32 Kb/s. Later, G.729A, a reduced-complexity version of G.729, was developed [12]. During the same period, ITU-T adopted G.723.1 as a standard speech coder to be used in the visual telephony as part of the overall H.324 family. In 1996, U.S. Department of Defense (DoD) standardized a new 2.4 Kb/s vocoder with communications quality to replace both the FS1015 and FS1016. There were seven candidates involved in this standardization and the winner was the Mixed-Excitation Linear Predictive Vocoder (MELP) developed by Texas Instruments [13]. It has been reported that its speech quality is even better than that of FS1016 4.8 Kb/s vocoder, a vocoder with twice the bit rate of MELP. It is also computationally efficient and robust in difficult background environments such as those encountered in commercial and military communication systems. Recently, ITU has set a demanding goal of further reducing the existing toll quality rate by a factor of two, down to the regions of 4 Kb/s with a quality equivalent to the existing 8 Kb/s standard (G.729). It is expected that this standardization will be finalized soon. There are numerous intended applications for this standardization such as visual telephony, multimedia applications in personal communication environments and Internet telephony. A worldwide effort is currently underway to prepare for this standardization. Table 1.1 lists some of the existing speech coder standards [1].

1.5 Scope and Organization of the Thesis

The main concern of this thesis is to reduce the complexity of the existing standard speech coder G.729A. In this thesis, a modified fixed codebook search scheme is proposed and incorporated into G.729A, which will be henceforth referred to as the modified coder. In order to determine the reduction in the complexity for the new search scheme, tests are carried out first on a general PC. Then, the scheme is implemented on TMS320C5416 DSK DSP board, followed by some optimization procedures employed to reduce the run-time of the speech coder. The test results both on the general PC and on the DSP board indicate that the proposed fixed codebook search scheme has savings of about 30% in computation complexity.

The thesis is organized as follows. Chapter 2 gives an overview of the linear prediction technique, which is essential for a discussion of the CELP speech coder. An overview of the basic structure of G.729A is also included in this chapter. Chapter 3 first describes in detail the codebook search algorithms used in G.729 and G.729A. Then, a new scheme for codebook search that reduces the computational complexity is proposed. In Chapter 4, implementations of the speech coder G.729A and of the modified coder are carried out on TMS320C5416 DSK board. Further, the implementation results of the two coders are analyzed and compared. Chapter 5 summarizes the main contributions of the thesis and gives future research directions.

Chapter 2

Introduction to LPC and the G.729 and the G.729A Speech Coders

Since linear predictive coding (LPC) analysis is an essential part in most speech coding algorithms, in this chapter, we will examine the algorithm in detail and see how LPC can remove the short-term correlation (redundancy) in a speech signal. Also, we will introduce the ITU-T G.729 and G.729A standards in this chapter.

2.1 Linear Predictive Speech Coding

LPC is the most common technique for low-bit-rate speech coding and is a very important tool in speech analysis. The popularity of LPC derives from its precise representation of the speech spectral magnitude, as well as its relatively simple computation. In speech coding, the LPC is used primarily to provide a small set of speech parameters that represent the configuration of the vocal tract. LPC estimates each speech sample based on a linear combination of its p previous samples, a larger p enables a more accurate model.

The main advantage of LPC analysis is that the speech is decomposed into two highly independent components: the vocal tract parameters (LP coefficients) and the glottal excitation (LP excitation). These two components have very different quantization requirements. As a result, different analysis schemes have been applied to enhance the coding efficiency.

We first consider the underlying principles of the short term LPC analysis and discuss how to calculate the LP coefficients. Short-term correlation can be effectively removed from the speech signals using a time varying linear analysis filter A(z) given by [6]

$$A(z) = 1 - \sum_{i=1}^{p} a_i z^{-i} \tag{2.1}$$

where $a_i's$ are linear prediction coefficients, and p is the prediction order. When the speech signal *s(n)* is processed by this filter, a residual signal *r(n)* is produced at the output. The inverse filter of *A(z)* is called the synthesis filter. The synthesis filter models the effect of the vocal tract imposed on the glottal excitation and thus, the frequency response of the synthesis filter corresponds to the spectral envelope (short term correlations) of the input speech signal. In other words, the center frequencies of the resonance of the filter should closely match the formant locations of the speech signal. As a result, the order p of the filter should be chosen such that there is a pair of poles allocated for each formant [6]. For practical speech signals sampled at 8 kHz, normally there are four formants; thus, it is sufficient to set p = 10. Figure 2.1 illustrates the analysis filter, Figure 2.2 shows an example of the speech signal and its residual signal, and Figure 2.3 the synthesis filter.

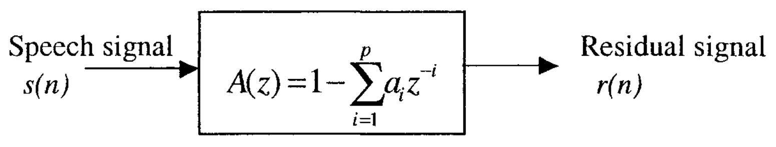


Figure 2.1 The LP analysis filter

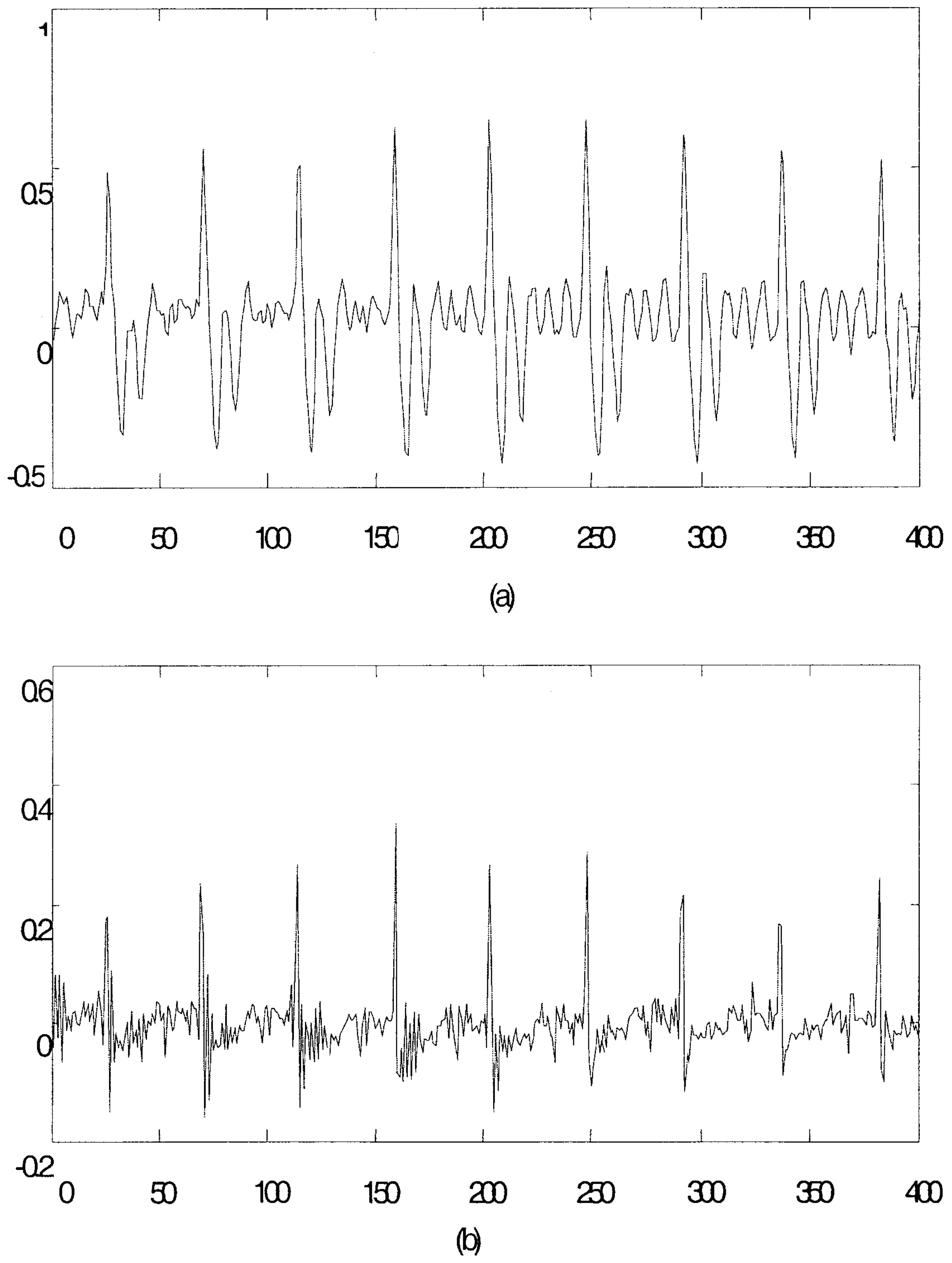


Figure 2.2 An example of speech signal and its residual signal
(a) Female speech signal “e”; (b) The residual signal of “e”

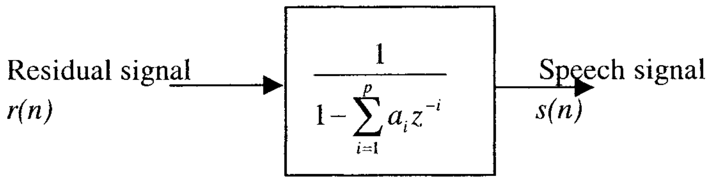


Figure 2.3 The LP synthesis filter

Mathematically, we can express the relationship between *r(n)* and *s(n)* by the following difference equation

$$r(n) = s(n) - \sum_{k=1}^{p} a_k s(n-k) \qquad (2.2)$$

The residual signal *r(n)* is also called the error signal *e(n)*. There are two methods used to estimate the coefficients a_k. One is the autocorrelation method and the other the covariance method. We will use the autocorrelation method. This method uses the least squares technique and chooses a_k such that the mean energy of the resulting error signal is minimized. The energy of the prediction error E is

$$E = \sum_{n=-\infty}^{\infty} r^2(n) = \sum_{n=-\infty}^{\infty} \left[S_w(n) - \sum_{k=1}^{p} a_k S_w(n-k) \right]^2 \qquad (2.3)$$

where $S_w(n)$ is the windowed speech signal. The values of the coefficients a_k are derived by setting

$$\frac{\partial E}{\partial a_k} = 0 \quad \text{for } k = 1, 2, \ldots, p$$

This will yield a system of p linear equations

$$\sum_{n=-\infty}^{\infty} S_w(n)S_w(n-i) = \sum_{k=1}^{p} a_k \sum_{n=-\infty}^{\infty} S_w(n-i)S_w(n-k), \quad i = 1, 2, \ldots, p \tag{2.4}$$

The autocorrelation function of the windowed signal $S_w(n)$ is

$$R(i) = \sum_{n=-\infty}^{\infty} S_w(n)S_w(n-i) = \sum_{n=i}^{L-1} S_w(n)S_w(n-i) \tag{2.5}$$

Hence, (2.4) reduces to

$$R(i) = \sum_{k=1}^{p} a_k R(i-k) \qquad \text{for } i = 1, 2, \ldots, p \tag{2.6}$$

Since the autocorrelation function is an even function, $R(i) = R(-i)$; hence, (2.6) can be expressed in a matrix form as

$$\begin{bmatrix} R(0) & R(1) & \cdots & R(p-1) \\ R(1) & R(0) & \cdots & R(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R(p-1) & R(p-2) & \cdots & R(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} R(1) \\ R(2) \\ \vdots \\ R(p) \end{bmatrix} \tag{2.7}$$

or, $\boldsymbol{RA} = \boldsymbol{r}$.

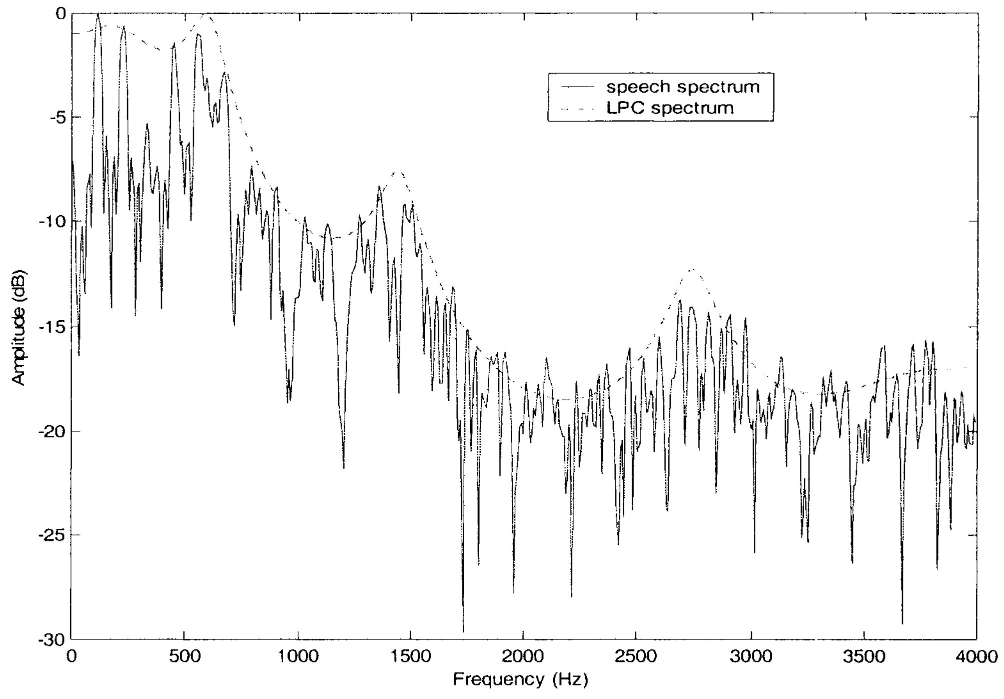
Solution of the above equation requires the inversion of the matrix $\boldsymbol{R}$ and multiplication of the resultant $p \times p$ matrix by the $\boldsymbol{r}$ vector. However, we can exploit the redundancy in the matrix $\boldsymbol{R}$ to simplify the computation. Notice that the matrix $\boldsymbol{R}$ is symmetric and it has a Toeplitz structure, that is, all the elements along a given diagonal are equal. This additional redundancy in the matrix allows the use of the more efficient Levinson-Durbin recursive procedure [14], in which the following set of ordered equations are solved recursively for $m = 1, 2, \ldots, p$.

$$
\begin{aligned}
k_m &= \frac{R(m) - \sum_{k=1}^{m-1} a_{m-1}(k) R(m-k)}{E_{m-1}} \\
a_m(m) &= k_m, \\
a_m(k) &= a_{m-1}(k) - k_m a_{m-1}(m-k) \\
E_m &= (1 - k_m^2) E_{m-1},
\end{aligned}
\tag{2.8}
$$

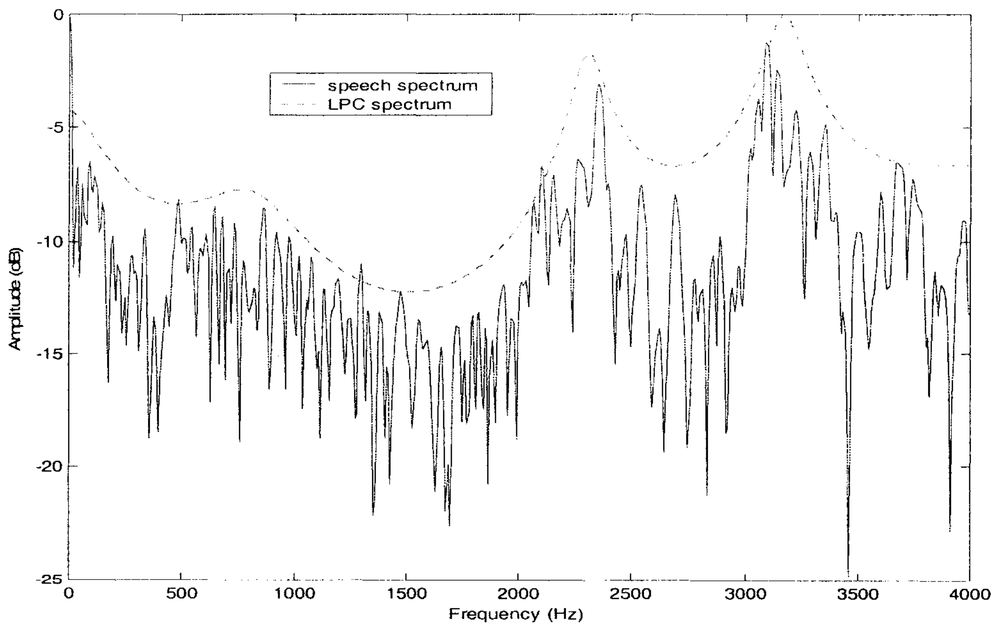
where initially $E_0 = R(0)$ and $a(0) = 1.0$, $1 \le k \le m\text{-}1$. The final solution is given as $a(k) = a_m(k)$. During each cycle m, the coefficients $a_m(k)$ describe the optimal m^{th} order linear predictor, and the minimum error E_m is reduced by the factor ($1 - k_m^2$). Since E_m is never negative, $|k_m| \le 1$. This condition on the reflection coefficients k_m also guarantees a stable LPC synthesis filter, since all the roots of $A(z)$ are then inside the unit circle in the z plane. Figures 2.4 (a) and 2.4 (b) illustrate the LPC analysis results respectively for the voiced and unvoiced speech signals.

2.2 ITU-T G.729 and G.729A Recommendations

This section describes how G.729 and G.729A work as stated in the ITU standard. The ITU-T recommendations contain the descriptions of algorithms for the coding of speech signals at 8 Kb/s using algebraic-code excited linear prediction (ACELP). These coders are designed to operate with a digital signal by first performing telephone bandwidth filtering of the analog signal, then sampling it at 8000 Hz, followed by conversion to 16-bit linear PCM for input to the encoder. The output of the decoder is converted back into analog signal by similar means.



(a)



(b)

Figure 2. 4 LPC spectrums

(a) the voiced signal "a"; (b) the unvoiced signal "t"

2.2.1 General Description of the ACELP Speech Encoder

The coders are based on a code-excited-linear-prediction (CELP) coding model [15]. In this model, the locally decoded signal is compared with the original signal and the coder parameters are selected such that the mean square weighted error between the original and reconstructed signal is minimized. The coder operates on frames of 10 ms, using a 5 ms look-ahead for linear prediction analysis. This results in an overall algorithm delay of 15 ms. The encoding principle is shown in Fig 2.5.

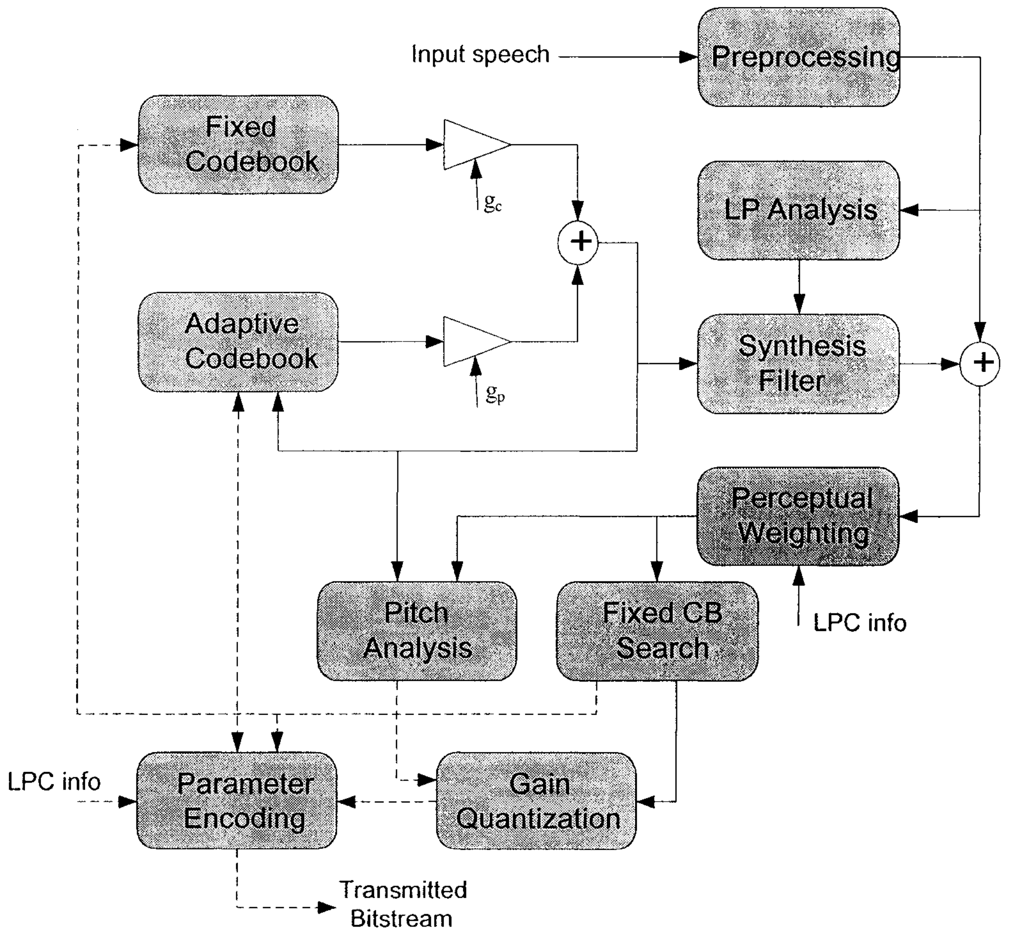


Figure 2.5 Encoding graph of G.729 and G.729A

After processing the 16-bit input samples through a high-pass filter, whose cut-off frequency is 140 Hz, a tenth-order LP analysis is performed, and the LP parameters quantized in the line spectral pair (LSP) domain with 18 bits. The input frame is divided into two sub-frames of 5 ms each. The use of the sub-frames allows for better tracking of the pitch and gain parameters and reduces the complexity of the codebook search. For each sub-frame, the excitation is represented by an adaptive-codebook and a fixed-codebook contribution. The adaptive and fixed-codebook parameters are transmitted every sub-frame.

The adaptive-codebook component represents the periodicity in the excitation signal using a fractional pitch lag with 1/3 sample resolution. The adaptive-codebook is searched using a two-step procedure. An open loop pitch lag is estimated once per frame based on the perceptually weighted speech signal. The adaptive codebook index and gain are found by a closed-loop search around the open-loop pitch-lag. The target signal, which is the signal to be matched, is computed by filtering the LP residual through the weighted synthesis filter. The adaptive codebook index is encoded with 8 bits in the first sub-frame and encoded with 5 bits in the second sub-frame. The target signal is updated by removing the adaptive codebook contribution, and this new target is used in the fixed codebook search. The fixed codebook is a 17-bit algebraic codebook [17]. The gains of the adaptive and fixed codebooks are vector quantized with 7 bits using a conjugate-structure codebook [18]. The bit allocation for a 10 ms frame is shown in Table 2.1.

TABLE 2.1 BIT ALLOCATION OF THE ITU-T 8KB/S SPEECH CODER

Parameter	Codeword	Subframe1	Subframe2	Total
LSP coefficients	L0,L1,L2, L3			18
Adaptive-codebook index	P1,P2	8	5	13
Delay parity bit	P0	1		1
Fixed-codebook index	C1,C2	13	13	26
Fixed-codebook pulse signs	S1,S2	4	4	8
Codebook gains(stage 1)	GA1,GA2	3	3	6
Codebook gains(stage 2)	GB1,GB2	4	4	8
Total				80

(1) Preprocessing

The 16-bit PCM input samples to the speech encoder are filtered with a second-order pole/zero high-pass filter with a cut-off frequency of 140 Hz. The filter prevents the undesired low frequency or DC components. Also, to prevent any overflow in the fixed-point implementation, the input values are divided by two. The high-pass filter is given by [5]

$$H_{hl}(z) = \frac{0.46363718 - 0.92724705z^{-1} + 0.4636718z^{-2}}{1 - 1.9059465z^{-1} + 0.9114024z^{-2}}$$

The input signal filtered through $H_{hl}(z)$ is referred to as $s(n)$, and is used in all the subsequent encoder operations.

(2) LP Analysis

An LP analysis is performed on each speech frame using the autocorrelation method with a 30 ms asymmetric window. Every 80 samples (10 ms), the autocorrelation coefficients of the windowed speech are computed and converted to LP coefficients using the Levinson-Durbin algorithm. The short term synthesis filter is based on a tenth-order LP filter, given by

$$\frac{1}{\hat{A}(z)} = \frac{1}{1+\sum_{i=1}^{10}\hat{a}_i z^{-i}}$$

where $\hat{a}_i$, $i = 1,\ldots,10$ are the quantized LP coefficients.

The LP analysis window consists of two parts: the first part is half of a Hamming window and the second part a quarter of a cosine function cycle. The window is given by

$$w_{lp}(n) = \begin{cases} 0.54-0.46\cos\left(\dfrac{2\pi n}{399}\right), & n=0,\ldots,199, \\ \cos\left(\dfrac{2\pi(n-200)}{159}\right), & n=200,\ldots,239. \end{cases} \tag{2.9}$$

There is a 5 ms look-ahead in the LP analysis, which means that 40 samples are needed from the future speech frame. This translates into an extra algorithmic delay of 5 ms at the encoder. The use of an asymmetric window allows a reduction in the look-ahead without compromising quality [19]. The LP analysis window is applied to 120 samples from the past speech frames, 80 samples from the present speech frame, and 40 samples from the future frame. The use of a 30 ms window was found to provide a smoother evolution of the LP filter, thereby providing a better speech quality. The graph of the window is shown in Figure 2.6.

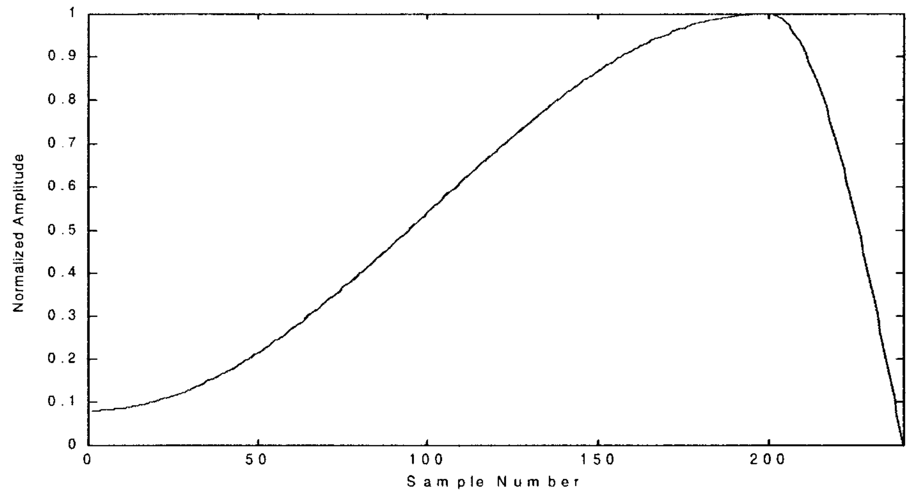


Figure 2.6 The asymmetric window used in G.729 [5].

The windowed speech given by

$$S_w(n) = w_{lp}(n)s(n), \qquad n = 0,\ldots,239 \tag{2.10}$$

is used to compute the autocorrelation coefficients:

$$r(k) = \sum_{n=k}^{239} S_w(n)S_w(n-k), \quad k = 0,\ldots,10 \tag{2.11}$$

A 60 Hz bandwidth expansion [20] is applied; this is achieved by multiplying the autocorrelation coefficients with

$$w_{lag}(k) = \exp[-\frac{1}{2}(\frac{2\pi f_0 k}{f_s})^2], \quad k = 1,\ldots,10 \tag{2.12}$$

where f_0 = 60 Hz is the bandwidth expansion and f_s = 8000 Hz is the sampling frequency. The bandwidth expansion on the autocorrelation coefficients reduces the possibility of ill-conditioning in the Levinson-Durbin algorithm. The modified autocorrelation coefficients are given by:

$$r'(k) = w_{lag}(k)r(k) \qquad k = 1,\dots,10$$

The modified autocorrelation coefficients are used to obtain the LP filter coefficients a_i, $i = 1,\dots 10$, by using the Levinson-Durbin algorithm [14].

(3) Perceptual Weighting

The weighted speech signal *sw(n)* in a sub-frame is obtained by filtering the speech through a perceptual weighting filter *W(z)* [21]. This perceptual weighting filter is based on the LP filter coefficients a_i and is given by

$$W(z) = \frac{A(z/\gamma_1)}{A(z/\gamma_2)} = \frac{1+\sum_{i=1}^{10}\gamma_1^i a_i z^{-i}}{1+\sum_{i=1}^{10}\gamma_2^i a_i z^{-i}} \tag{2.13}$$

The use of the unquantized coefficients gives a weighting filter that matches better the original spectrum, in G.729, γ_1 and γ_2 are made adaptive as functions of the spectral shape of the input signal. However, this adaptive process requires a large computation. In order to reduce the complexity, the filter is modified in G.729A as

$$W(z) = \frac{\hat{A}(z)}{\hat{A}(z/\gamma)} \tag{2.14}$$

This filter is based on the quantized LP filter coefficients $\hat{a}_i$, with a fixed parameter $\gamma = 0.75$. This simplifies the combination of the synthesis and weighting filters to $W(z)/\hat{A}(z) = 1/\hat{A}(z/\gamma)$, which reduces the number of filtering operations while computing the impulse response and the target signal.

(4) Pitch Analysis

To reduce the complexity of the search for the best pitch delay, the pitch search range is limited around a candidate delay T_{op}, which is obtained from an open-loop pitch analysis once every 10 ms using the weighted speech signal *sw(n).* The adaptive codebook approach is used to represent the periodic component in the excitation. The selected adaptive codebook vector is represented by an index, which corresponds to a certain fractional lag value.

For each sub-frame, the target signal *x(n)* and the impulse response *h(n)* of the weighted synthesis filter are computed. The target signal *x(n)* is computed by filtering the LP residual signal *r(n)* using the weighted synthesis filter $1/\hat{A}(z/\gamma)$. The impulse response *h(n)* of the weighted synthesis filter is computed for each sub-frame by filtering a signal consisting of a unit sample extended by zeros through the filter $1/\hat{A}(z/\gamma)$.

The detailed procedure of searching T_{op} is as follows.

Three maximum values, one for each of the following three ranges

$i = 1$: 20,21,...,39

$i = 2$: 40,41,...,79

$i = 3$: 80,81,...,143

are found using the correlation

$$R(k) = \sum_{n=0}^{79} sw(n)sw(n-k) \tag{2.15}$$

These maximum values $R(t_i)$, $i = 1,2,3$, are normalized as

$$R'(t_i) = \frac{R(t_i)}{\sqrt{\sum_n sw^2(n-t_i)}} \tag{2.16}$$

The winner among the three normalized correlations is selected by favoring the delays with the values in the lower range. This is achieved by weighting the normalized correlations corresponding to the longer delays. The best delay T_{op} is determined as follows.

```
$T_{op} = t_3$
$R'(T_{op}) = R'(t_3)$
$if\ R'(t_2) \geq 0.85\ R'(T_{op})$
$R'(T_{op}) = R'(t_2)$
$T_{op} = t_2$
$end$
$if\ R'(t_1) \geq 0.85\ R'(T_{op})$
$R'(T_{op}) = R'(t_1)$
$T_{op} = t_1$
$end$
```

This procedure of dividing the delay range into three sections and favoring the smaller range is used to avoid choosing pitch multiples. In G.729A, the pitch analysis is simplified from that in G.729 by using decimation when computing the correlations of the weighted speech, that is, (2.15) is modified as follows

$$R(k) = \sum_{n=0}^{39} sw(2n)sw(2n-k) \tag{2.17}$$

Based on the informal subjective tests reported in [22], the above simplification of the open loop analysis does not introduce any significant degradation in the performance of the coder.

After T_{op} is decided, a closed loop adaptive codebook search is performed in the first sub-frame around the index corresponding to T_{op}. In this sub-frame, a fractional pitch delay T_l is used with a resolution of 1/3 in the range $\left[19\frac{1}{3},\ 84\frac{2}{3}\right]$ and only integers are used in the range [85, 143]. It has been found that this choice of resolution provides a good trade-off between the performance and the bit rate. For the second sub-frame, a delay T_2 with a resolution of 1/3 is always used in the range $\left[\mathrm{int}(T_1)-5\frac{2}{3},\ \mathrm{int}(T_1)+4\frac{2}{3}\right]$, where int($T_l$) is the integer part of the fractional pitch delay T_l of the first sub-frame. For each sub-frame, the optimal delay is determined using closed loop analysis that minimizes the mean-squared error between the original signal and the reconstructed speech signal. In the first sub-frame, the delay T_l is found by searching a small range (six samples) of delay values around the open loop delay T_{op}. The search boundaries t_{min} and t_{max} are defined as follows.

```
$t_{min} = T_{op} - 3$
if $t_{min} < 20$ then $t_{min} = 20$
$t_{max} = t_{min} + 6$
if $t_{max} > 143$ then $t_{max} = 143$
$t_{min} = t_{max} - 6$
end
```

For the second sub-frame, closed loop pitch analysis is done around the pitch selected in the first sub-frame to find the optimal delay T_2. The search boundaries are between $t_{\min} - \frac{2}{3}$ and $t_{\max} + \frac{2}{3}$, where t_{min} and t_{max} are derived from T_1 as follows.

```
$t_{\min} = \text{int}(T1) - 5$
if $t_{\min} < 20$ then $t_{\min} = 20$
$t_{\max} = t_{\min} + 9$
if $t_{\max} > 143$ then $t_{\max} = 143$
    $t_{\min} = t_{\max} - 9$
end
```

The closed loop pitch search minimizes the mean squared weighted error between the original and reconstructed speech. This is achieved by maximizing the term

$$R(k) = \frac{\sum_{n=0}^{39} x(n) y_k(n)}{\sqrt{\sum_{n=0}^{39} y_k(n) y_k(n)}} \tag{2.18}$$

where $x(n)$ is the target signal and $y_k(n)$ is the past filtered excitation at delay k, which is past excitation convolved with $h(n)$.

The pitch delay T_1 is encoded with 8 bits in the first sub-frame and the relative delay in the second sub-frame is encoded with 5 bits. A fractional delay T is represented by its integer part *int(T)*, and a fractional part *frac*/3, *frac* = -1,0,1. The pitch index is now encoded as

$$P_1 = \begin{cases} 3(\text{int}(T_1) - 19) + frac - 1, & \text{if } T_1 \in [19\frac{1}{3},\ 84\frac{2}{3}] \\ (\text{int}(T_1) - 85) + 197, & \text{if } T_1 \in [85,\ 143] \end{cases} \tag{2.19}$$

The value of the pitch delay T_2 is encoded relative to the value of T_1. The fractional delay T_2 is represented by its integer part int(T_2), and a fractional part *frac/3*, *frac* = -1,0,1. It is encoded as

$$P_2 = 3(\text{int}(T_2) - t_{\min}) + frac + 2 \tag{2.20}$$

where t_{min} is derived from T_1 as described above.

To make the coder more robust against random bit errors, a parity bit *P0* is computed on the delay index *P1* of the first sub-frame. This parity bit is generated through an *XOR* operation on the six most significant bits of *P1*. At the decoder, this parity bit is recomputed and if the recomputed value does not agree with the transmitted value, an error concealment procedure is applied.

Once the adaptive codebook is determined, the adaptive codebook gain g_p is computed as

$$g_p = \frac{\sum_{n=0}^{39} x(n)y(n)}{\sum_{n=0}^{39} y(n)y(n)} \tag{2.21}$$

where *x(n)* is the target signal and *y(n)* is the adaptive codebook vector. The vector *y(n)* is obtained by convolving *v(n)* with *h(n)* as

$$y(n) = \sum_{i=0}^{n} v(i)h(n-i) \qquad n = 0,\ldots,39 \tag{2.22}$$

where $h(n)$ is the impulse response of the weighted synthesis filter $1/\hat{A}(z/\gamma)$, and $v(n)$ is the adaptive codebook contribution.

After the adaptive codebook search for the optimum delay and the gain, the fixed codebook vector is searched. Then the adaptive codebook gain g_p and the fixed codebook gain g_c are jointly quantized using a two-staged conjugate structured codebook [5]. The first codebook is a 3-bit two-dimensional codebook $\mathcal{F}$, and the second codebook is a 4-bit two-dimensional codebook $\mathcal{G}$. The first element in each codebook is $\hat{g}_p$, and the second element is the fixed codebook gain correction factor $\hat{\gamma}$. The term conjugate means that each input vector is quantized as a linear combination of both codebooks. The conjugate structure reduces both computational and memory requirements [27]. After the gain quantization, the final indices of the two codebooks are represented as GA and GB respectively. The fixed codebook search algorithm will be discussed in detail in the next chapter.

2.2.2 Functional Description of the Decoder

The function of the decoder (Figure 2.7) consists of decoding the transmitted parameters (LP parameters, adaptive codebook vector and fixed codebook vector), and performing the synthesis to obtain the reconstructed speech. This is followed by a post-processing stage, which consists of a post-filter and a high-pass filter. For each sub-frame, the LSP coefficients are converted back to the LP filter coefficients a_i, which are used to construct the synthesis filter. The following four steps are used repeatedly for each sub-frame.

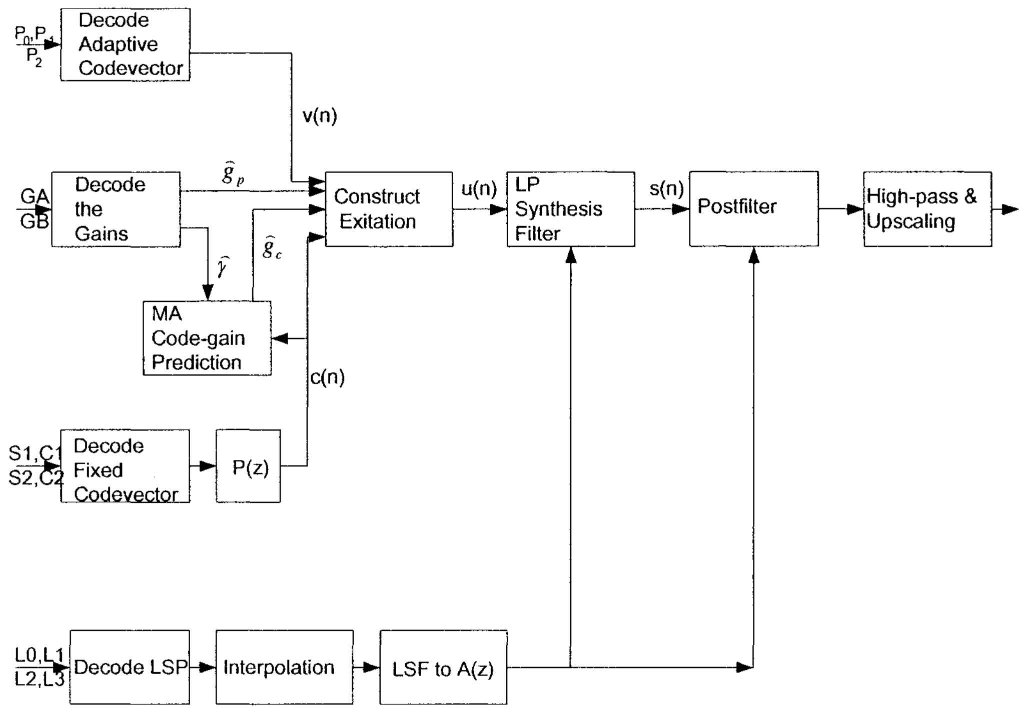


Figure 2.7 Decoding structure for G.729 and G.729A [5]

(1) Decoding of the Adaptive Codebook Vector

The integer and fractional parts of the pitch delay T_1 are derived from the received adaptive codebook index *P1*. It is decoded as follows.

```
if $p_1 < 197$
$\text{int}(T_1) = (P_1 + 2)/3 + 19$
$frac = P_1 - 3\text{int}(T_1) + 58$
else
$\text{int}(T_1) = P_1 - 112$
$frac = 0$
end
```

The integer and fractional part of T_2 are obtained from *P2* and t_{min}, where t_{min} is derived from T_1 as follows:

```
$t_{\min} = \text{int}(T_1) - 5$
if $t_{\min} < 20$ then $t_{\min} = 20$
$t_{\max} = t_{\min} + 9$
if $t_{\max} > 143$ then
$t_{\max} = 143$
$t_{\min} = t_{\max} - 9$
end
```

T_2 is now decoded using

$$\text{int}(T_2) = (P_2 + 2)/3 - 1 + t_{\min}$$
$$frac = P_2 - 2 - 3((P_2 + 2)/3 - 1)$$

After decoding T_1 and T_2, the adaptive codebook vector *v(n)* is found by interpolating the past excitation signal *u(n)* at a given pitch delay.

(2) Decoding of the Fixed Codebook Vector

After receiving the codebook index C and the sign S, the corresponding four pulse positions and pulse signs are determined. Then the fixed codebook vector is constructed using following equation.

$$c(n) = s_0\delta(n - m_0) + s_1\delta(n - m_1) + s_2\delta(n - m_2) + s_3\delta(n - m_3), \quad n = 0,\ldots,39 \qquad (2.23)$$

(3) Decoding of the Adaptive and Fixed Codebook Gains

After receiving the indices GA and GB for codebook $\mathcal{F}$ and $\mathcal{G}$ respectively, the adaptive codebook gain $\hat{g}_p$ is obtained by [5]:

$$\hat{g}_p = \mathcal{F}_1(\text{GA}) + \mathcal{G}_1(\text{GB})$$

and the quantized fixed codebook gain is given by:

$$\hat{g}_c = g'_c \hat{\gamma} = g'_c(\mathcal{F}_2(\mathrm{GA}) + \mathcal{G}_2(\mathrm{GB}))$$

where g'_c is a predicted gain based on previous fixed codebook energies.

(4) Computation of the Reconstructed Speech

As illustrated in Figure 2.6, the speech signal is recovered by filtering the excitation signal *u(n)* through the synthesis filter, and is given by

$$s(n) = u(n) - \sum_{i=1}^{10} a_i s(n-i), \qquad n = 0,1,\ldots,39 \tag{2.24}$$

The reconstructed speech *s(n)* is then processed by the post processing filters, which include a high-pass filter and scaling by a factor of 2.

2.3 Conclusion

A detailed description of the ITU-T G.729 and G.729A speech coders has been given in this chapter. It consists of two parts, namely, the coding and the decoding processes. In the coding part, a few essential signal-processing functions have been presented in detail. These include the LPC and pitch analyses. Some of the major differences between the G.729 and G.729A have been described. Also, the decoding process, which is essentially the same for both G.729 and G.729A, has been briefly presented. In next chapter, a modified fixed codebook search that reduces the complexity of the speech coder G.729A is proposed.

Chapter 3

Proposed Fixed Codebook Search Scheme

3.1 Introduction

Reducing the complexity has been a major concern in designing and implementing CELP speech coders. Recently, a large number of papers have appeared dealing with the reduction of the complexity of CELP coders [23-30]. All these papers have focused on the design of the fixed codebook structure and on efficient search algorithms in finding the optimum codeword in the fixed codebook. As described in Chapter 2, the ITU-T recommendation G.729A is one such speech coder having a much reduced complexity compared to that of G.729. Table 3.1 gives a breakdown of the complexities for the codecs G.729 and G.729A [27]. It is seen from this table that G.729A has about one half of the computation complexity of G.729 (12.418 vs 22.315 MIPS). About 50 percent of the complexity reduction in the coder part (a saving of about 5 MIPS) is due to the algebraic codebook search of G.729A. However, it is seen from this table that the computation of the algebraic codebook search still takes up about thirty percent of the encoding process. Therefore, in this thesis, we focus our attention in reducing the complexity of the fixed codebook search and propose a new codebook search scheme. This chapter is organized as follows. An overview of the theory of codebook search is given in Section 3.2. The search schemes used in G.729 and G.729A are explained in detail in Section 3.3. A new search scheme that reduces the computation complexity of G.729A is proposed in Section 3.4. Section 3.5 contains the simulation results, obtained on a general purpose PC, concerning the

TABLE 3.1 BREAKDOWNS OF THE COMPLEXITIES OF THE CODECS G.729 AND G.729A IN TERMS OF WMOPS[1] AND TMS320C50 MIPS

	WMOPS		C50 MIPS	
Function	G.729	G.729A	G.729	G.729A
Pre-processing	0.20	0.20	0.226	0.226
LP analysis	1.63	1.28	1.957	1.696
LSP quantization & inter.	0.95	0.95	1.390	1.390
LSP to A(z) and weighting	0.30	0.12	0.461	0.173
Open-loop pitch	1.45	0.82	1.563	0.955
Closed-loop pitch	2.83	1.55	3.453	1.778
Algebraic codebook	6.35	1.86	8.406	3.046
Quantization of gains	0.46	0.46	0.643	0.643
Find exe & memory update	0.21	0.08	0.278	0.112
Total(coder)	14.38	7.32	18.377	10.019
Decoder	0.68	0.68	0.133	0.133
Post-filter	2.13	0.73	2.539	1.000
Post-processing	0.22	0.22	0.266	0.266
Total (decoder)	3.03	1.63	3.938	2.399
Total (duplex)	17.41	8.95	22.315	12.418

[1] WMOPS represents weighted million operations per second; see reference [27] for details.

speech quality for G.729, G.729A and the modified codec using the proposed scheme for the codebook search. A comparison of the three schemes in terms of the complexity is also carried out. Conclusions are given in Section 3.6.

3.2 Fixed Codebook Search Criterion

The fixed codebook is based on an algebraic codebook structure using an interleaved single-pulse permutation (ISPP) design. The algebraic codebook is a deterministic codebook rather than the random codebook used in the traditional CELP coders. In this codebook, each codebook vector contains four nonzero pulses. Each pulse can have either an amplitude of +1 or −1, and can assume four different positions. The algebraic codebook structure has advantages in terms of storage, search complexity, and robustness [31-33].

The k^{th} fixed codebook vector $\boldsymbol{c}_k$ is searched by minimizing the mean squared weighted error between the target signal $\hat{\boldsymbol{x}}$ and the synthesized speech signal. The error is defined as [33]

$$E_k = \|\hat{\boldsymbol{x}} - g\boldsymbol{H}\boldsymbol{c}_k\|^2 \tag{3.1}$$

where $\boldsymbol{H}$ is a lower convolution matrix constructed from the impulse response of the weighted synthesis filter[2], g is a gain scaling factor, and $\hat{\boldsymbol{x}}$ is given by

$$\hat{x}(n) = x(n) - g_p y(n) \qquad n = 0,\ldots,39 \tag{3.2}$$

in which $x(n)$ is the target signal for the adaptive codebook search, $\boldsymbol{y}$ is the filtered adaptive codebook vector of (2.22) and g_p is the scaling gain factor given by (2.21).

[2] Please see (2.14)

The principal diagonal elements of $\boldsymbol{H}$ are *h[0]* and the lower diagonals are *h[1], h[2],....., h[L-1]* respectively, where L is sub-frame size. Minimizing E_k, by setting $\partial E/\partial g = 0$, yields

$$g = \frac{\hat{\boldsymbol{x}}^T \boldsymbol{H} \boldsymbol{c}_k}{\boldsymbol{c}_k^T \boldsymbol{H}^T \boldsymbol{H} \boldsymbol{c}_k} \tag{3.3}$$

Substituting (3.3) into (3.1) we get

$$E_k = \hat{\boldsymbol{x}}^T \hat{\boldsymbol{x}} - \frac{(\hat{\boldsymbol{x}}^T \boldsymbol{H} \boldsymbol{c}_k)^2}{\boldsymbol{c}_k^T \boldsymbol{H}^T \boldsymbol{H} \boldsymbol{c}_k}$$

$$= \hat{\boldsymbol{x}}^T \hat{\boldsymbol{x}} - \frac{(\boldsymbol{d}^T \boldsymbol{c}_k)^2}{\boldsymbol{c}_k^T \boldsymbol{\Phi} \boldsymbol{c}_k} \tag{3.4}$$

where $\boldsymbol{d} = \boldsymbol{H}^T \hat{\boldsymbol{x}}$ is a vector containing the correlation between the target vector and the impulse response *h(n)* and $\boldsymbol{\Phi} = \boldsymbol{H}^T \boldsymbol{H}$ is the covariance matrix of the impulse response. The quantities $\boldsymbol{\Phi}$ and $\boldsymbol{d}$ are pre-computed before the codebook search using following formulas:

$$d(m) = \sum_{i=m}^{L-1} \hat{x}(i) h(i-m), \qquad m = 0,1,\ldots, L\text{-}1 \tag{3.5}$$

$$\phi(i,j) = \sum_{m=j}^{L-1} h(m-i) h(m-j), \; i = 0,1,\ldots.,L\text{-}1 \quad j = i,\ldots.,L\text{-}1 \tag{3.6}$$

Since the first term in (3.4) is fixed in searching the codeword $\boldsymbol{c}_k$, only the second term in (3.4) needs to be calculated. Hence, the optimum codeword is obtained by maximizing

$$\tau_k = \frac{(\boldsymbol{d}^T \boldsymbol{c}_k)^2}{\boldsymbol{c}_k^T \boldsymbol{\Phi} \boldsymbol{c}_k} = \frac{C^2}{\varepsilon} \tag{3.7}$$

in which $C^2 = (\boldsymbol{d}^T \boldsymbol{c}_k)^2$, $\varepsilon = \boldsymbol{c}_k^T \boldsymbol{\Phi} \boldsymbol{c}_k$. The winning codeword has the largest value of τ_k. Since the codeword $\boldsymbol{c}_k$ contains only four nonzero pulses, the numerator of (3.7) can be expressed as [5]

$$C^2 = (\sum_{i=0}^{3} s_i d(m_i))^2 \tag{3.8}$$

where s_i and m_i are respectively the sign and position of the i^{th} pulse. The denominator of (3.7) is given by [5]

$$\varepsilon = \sum_{i=0}^{3} \phi(m_i, m_i) + 2\sum_{i=0}^{2} \sum_{j=i+1}^{3} s_i s_j \phi(m_i, m_j) \tag{3.9}$$

3.3 Fixed Codebook Search Schemes in G.729 and G.729A

3.3.1 G.729 Search Scheme

As described above, the purpose of the fixed codebook search is to find the optimum position and amplitude of the four pulses to maximize the value of τ_k. The amplitude of each pulse can be either +1 or –1, and can assume the positions shown in Table 3.2.

In order to simplify the search procedure, the pulse amplitudes are predetermined by using the sign of the correlation signal *d(m)*. This choice of signs for a given combination of pulses maximizes the correlation term in (3.7) [5]. Therefore, before entering the codebook search, the following steps are carried out. First, the signal *d(m)* is decomposed into its absolute value $d'(m) = |d(m)|$ and its sign, which characterize

TABLE 3.2 STRUCTURE OF THE 17-BIT FIXED CODEBOOK

Pulse	Amplitude	Positions	Bits
i_0	$s_0 : \pm 1$	m_0: 0,5,10,15,20,25,30,35	1+3
i_1	$s_1 : \pm 1$	m_1: 1,6,11,16,21,26,31,36	1+3
i_2	$s_2 : \pm 1$	m_2: 2,7,12,17,22,27,32,37	1+3
i_3	$s_3 : \pm 1$	m_3: 3,8,13,18,23,28,33,38 4,9,14,19,24,29,34,39	1+4

the pre-selected pulse amplitudes at each of the 40 possible pulse positions. Second, in order to include the preset pulse amplitudes, the matrix $\boldsymbol{\Phi}$ is modified as

$$\phi'(m_i, m_j) = sign[s(m_i)]\, sign[s(m_j)]\, \phi(m_i, m_j) \qquad (3.10)$$

The main diagonal elements of $\boldsymbol{\Phi}$ are scaled to remove the factor 2 in (3.9) so that

$$\phi'(m_i, m_i) = 0.5\phi(m_i, m_i) \qquad m_i = 0,\ 1,\ 2,\ 3,\ \ldots, 39 \qquad (3.11)$$

The correlation in (3.8) is now given by

$$C^2 = (d'(m_0) + d'(m_1) + d'(m_2) + d'(m_3))^2 \qquad (3.12)$$

and the energy in (3.9) is given by

$$\begin{aligned}\varepsilon/2 &= \sum_{i=0}^{3}\phi'(m_i,m_i)+\sum_{i=0}^{2}\sum_{j=i+1}^{3}\phi'(m_i,m_j)\\ &= \phi'(m_0,m_0)+\phi'(m_1,m_1)+\phi'(m_0,m_1)\\ &\quad+\phi'(m_2,m_2)+\phi'(m_0,m_2)+\phi'(m_1,m_2)\\ &\quad+\phi'(m_3,m_3)+\phi'(m_0,m_3)+\phi'(m_1,m_3)\\ &\quad+\phi'(m_2,m_3)\end{aligned} \tag{3.13}$$

The goal of the codebook search is to combine the computation of (3.12) and (3.13) to maximize the value of (3.7). The optimal solution of the four pulse positions that maximizes the value of (3.7) is found by using the nest-loop search scheme. Under this search scheme, all of the possible combinations of the pulse positions are tested. There are four-nested loops, each loop corresponding to one of the pulse positions, and the contribution of a new pulse is added to calculate the value of C and ε in each loop. In the innermost loop, the contribution of all the four pulses are added in C and ε. Therefore, the value of (3.7) in this specific pulse combination, namely, $\{m_0, m_1, m_2, m_3\}$ is compared with the value due to the contribution of the previous set of pulse combinations. The set of pulse combinations, which has the largest value of C^2/ε, is kept for the next round of search. The best pulse position is found after searching all the possible pulse combinations. It is seen from (3.7), (3.12) and (3.13) that for each search loop, 12 additions, 1 multiplication and 1 division are needed to decide one set of positions, once the values of $d'(m)$ and $\phi'(m_i, m_j)$ have been pre-computed. To evaluate all the possible pulse positions, a total of $2^{13} = 8192$ combinations need to be examined. Thus, the computation is very large if this exhaustive search scheme is employed. To reduce the search complexity, simplified methods that set some additional restrictions are used to reduce the number of pulse combinations.

In G.729, a focused search scheme is used. From the above analysis, it is observed that the decision is made in the innermost loop and it also consumes most of the computations. Therefore, the computation will be reduced significantly by decreasing the number of times this innermost loop is entered. In view of the fact that most of the codewords have far less correlation values than the winning codeword (the codeword with the largest value of τ_k), a threshold is used to reduce the set of pulse combinations. It is set before the innermost loop is entered. The threshold is set as

$$thr = c_{av} + k(c_{\max} - c_{av}) \tag{3.14}$$

where c_{av} is the average correlation due to the contribution of the first three pulses, $c_{\max}$ is the maximum correlation due to the contribution of the first three pulses, and k is the coefficient that controls the threshold value, which is set to 0.4 in G.729. The quantities c_{av} and $c_{\max}$ are given by [5]

$$c_{av} = \frac{1}{8}\left(\sum_{n=0}^{7} d'(5n) + \sum_{n=0}^{7} d'(5n+1) + \sum_{n=0}^{7} d'(5n+2)\right) \tag{3.15}$$

and

$$c_{\max} = \max[d'(m_0)] + \max[d'(m_1)] + \max[d'(m_2)] \tag{3.16}$$

where $\max[d'(m_i)]$ is the maximum value of $d'(m_i)$ at the first three position tracks given in Table 3.2.

In the G.729 search scheme, if the intermediate absolute correlation contributed by the first three pulses is less than the value of *thr,* that means the value of the numerator of (3.7) is relatively small, these first three pulse positions are less likely to be part of the optimum codeword. Therefore, in order to reduce the computation, these pulse combinations are discarded instead of them being retained, thus avoiding the search for the fourth pulse. After using the threshold, a large part of the pulse combinations with small correlation values are discarded before entering the innermost loop. However, this search scheme has a disadvantage in that the search time is different from one sub-frame to another. In order to control the search time for the worst case, the focused search limits the number of search loops of the innermost loop up to 90 for a sub-frame. Hence, in the worst case, the total number of pulse combinations searched is $16 \times 90 = 1440$, which is only 17% of that for the exhaustive search, thus providing a significant reduction in computation complexity.

3.3.2 G.729A Search Scheme

The structure of the 17-bit fixed codebook of G.729A is the same as that of G.729; however, the codebook search scheme is completely different. As described above, a fast search procedure based on a nested-loop search approach is used in G.729. In this search, only 1440 combinations of the pulse positions are tested in the worst case instead of the possible 8192 combinations of the pulse positions (that is, only 17.5%). In G.729A, in order to further speed up the search procedure, the search criterion C^2 / ε is tested for an even smaller percentage of the possible combinations of the pulse positions using a depth-first tree search approach. In this approach, the P excitation pulses in a sub-frame are partitioned into M subsets of N_m pulses. The search begins with the first subset and proceeds with subsequent subsets according to a tree structure in which the subset m is searched at the m^{th} level of the tree. The search is repeated by changing the order in which the pulses are assigned to the position tracks. In G.729A, the candidate pulses are partitioned into 5 tracks, T_0, T_1, T_2, T_3, T_4, as shown in Table 3.3. The depth-first tree search shown in Table 3.4 performs two assignments, which are further divided into two subsets. The search procedure is summarized as follows.

For the first assignment, the depth-first tree search starts from determining the most-likely pulse positions (i_0, i_1). In the first subset, G.729A first finds the positions of the first two maxima of $|d(n)|$ in track T_2 and combines with all the 8 positions in track T_3. Thus, we have $2 \times 8 = 16$ combinations being tested in the first-stage search. When the optimal positions (i_0, i_1) are ready, the second-stage search then determines the positions (i_2, i_3) by exhaustively testing all the $8 \times 8 = 64$ combinations in tracks T_0 and T_1. This will give a total of $16 + 64 = 80$ combinations searched. By replacing T_3 with T_4, a similar search procedure is performed. Thus, the number of pulse positions searched in the first assignment is $2 \times 80 = 160$. For the second assignment, we also need to follow a similar search procedure. Hence, this search scheme requires a search of 320 combinations in total. This is about only 3.9% of all the possible combinations of the pulse positions. It is seen that using this search scheme, the search time for every frame is fixed.

TABLE 3.3 STRUCTURE OF THE ALGEBRAIC CODEBOOK

Pulse No	Track	Positions
0	T_0	0,5,10,15,20,25,30,35
1	T_1	1,6,11,16,21,26,31,36
2	T_2	2,7,12,17,22,27,32,37
3	T_3	3,8,13,18,23,28,33,38
	T_4	4,9,14,19,24,29,34,39

TABLE 3.4 SUBSET ASSIGNMENT OF DEPTH–FIRST TREE SEARCH FOR G.729A

First assignment			Second assignment		
Subset 1	i_0	T_2	Subset 1	i_0	T_3 or T_4
	i_1	T_3 or T_4		i_1	T_0
Subset 2	i_2	T_0	Subset 2	i_2	T_1
	i_3	T_1		i_3	T_2

3.4 Proposed Algebraic Codebook Search Scheme

From Table 3.1, it is obvious that the complexity for the algebraic codebook search has been reduced substantially in G.729A compared to that of G.729. In order to further reduce the complexity, we now propose a new fixed codebook search scheme. This new search scheme has a structure similar to that of G.729 except for two modifications.

As described in Section 3.2, the fixed codebook search criterion is to maximize the value of τ_k. The codeword with the largest value of τ_k will be chosen as the final codeword. We can imagine τ_k as the slope of a graph where the horizontal axis is energy ε, and the vertical axis is the correlation C^2. The winning codeword is the one with the largest slope. It is observed that only a small part of the codeword has a chance of being the winning codeword, and most of the codewords have, for their slopes, values that are far less than the value of the slope of the winning codeword [33]. This gives us a hint that we should focus our search only on those codewords that have relatively large slopes. This can be achieved in the following way.

Based on the conclusion in [29], it is observed that if the search is focused on the positions with large values of τ_{m_i} defined as

$$\tau_{m_i} = \frac{d(m_i)}{\phi(m_i, m_i)}, \tag{3.17}$$

the winning codeword can still be found. It is to be noted that the significance of τ_{m_i} lies in the fact that the pulse position with a large value of τ_{m_i} is more likely to be the part of the winning codeword. Therefore, the codebook structure in Table 3.2 is reordered according to the value of τ_{m_i}. Those positions with a large value of τ_{m_i} are put in the beginning of the codebook and are searched first. An example of a reordered codebook is presented in Table 3.5. From this table, it can be seen that for pulse #1, position 25 is the most probable place in Track T_0 and position 30 the least possible place. Therefore, if the search is focused on the first M positions in each track, the search time should be reduced. There is a trade off between the search complexity and the speech quality when choosing the value of M. The size of the search space for each sub-frame is $2 \times M^4$ when using the four-nested loop search scheme. According to [29], when $M = 5$, there is a 97% chance that one of the five positions might be a member of the winning codeword. Hence, in the proposed search scheme, the original codebook in G.729 is reordered according to τ_{m_i}, and the search focused on the first five pulse positions in each track. It is seen from Table 3.5 that when $M = 5$, only those positions in the highlighted area are searched. Using this modification, the number combinations of the pulse positions reduces to $2 \times 5^4 = 1250$, when the nested loop search method is used. However, this number is still far more than that of G.729A (320 per sub-frame). To further reduce the pulse combinations, a second modification is introduced.

TABLE 3.5 AN EXAMPLE OF REORDERED CODEBOOK

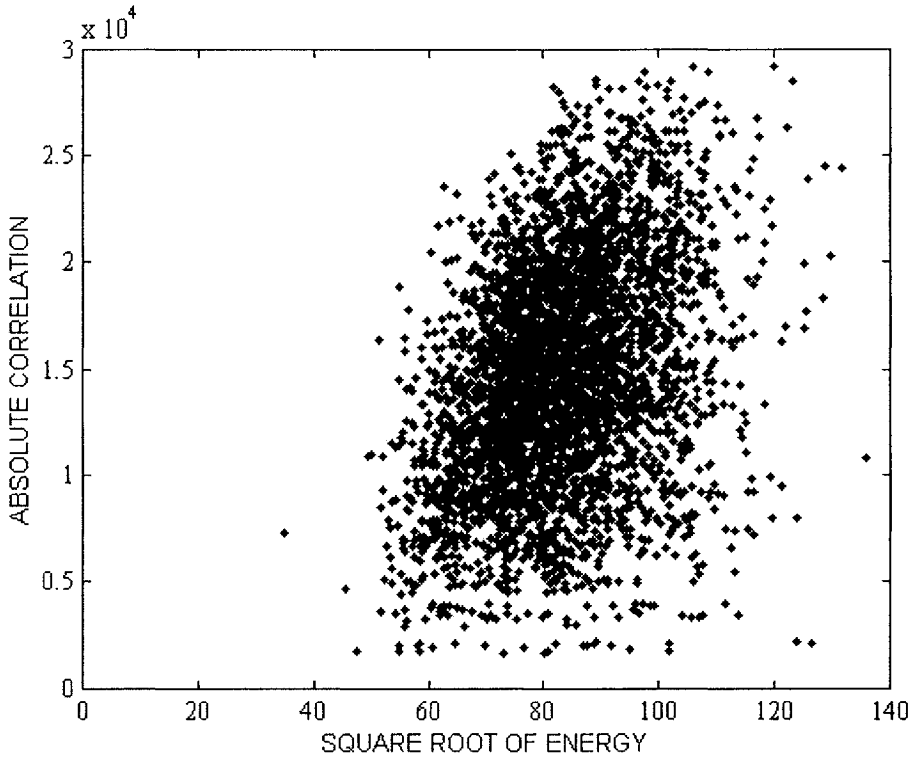
Pulse	Track	M=1	M=2	M=3	M=4	M=5	M=6	M=7	M=8
1	T_0	25	5	15	10	35	0	20	30
2	T_1	36	6	26	16	31	21	11	1
3	T_2	37	7	12	17	22	27	2	32
4	T_3	8	3	13	18	38	28	33	23
	T_4	4	9	34	39	24	29	14	19

Instead of one threshold used in G.729, two thresholds are predefined in the proposed search scheme. The first threshold, *thresh*1, is computed based on the average and the maximum of the absolute value of the correlation of the first two pulses, whereas the second threshold, *thresh*2, is calculated based on the contribution of the absolute value of the correlation of the first three pulses. After the first two pulses are added in the search loop, $d(m) = d(m_0) + d(m_1)$ is compared with *thresh*1 to decide whether to enter the third loop. If this value is smaller than *thresh*1, it means that the correlation between the target signal *x(n)* and the impulse response of *h(n)* is small at these pulse positions. Hence, these pulse positions are most probably not part of the winning codeword and thus discarded. Otherwise, the search is continued in the third loop. When the third loop is entered, $d(m) = d(m_0) + d(m_1) + d(m_2)$ is compared with *thresh*2. If $d(m)$ is greater than the threshold *thresh*2, the search is continued in the fourth loop. In the fourth loop, since all the positions for the four pulses are available, the value of τ_k is obtained, and this value is compared with the τ_k value of the previous codeword. The codeword with the larger value of τ_k is retained. The search is continued for the codeword having the largest value of τ_k.

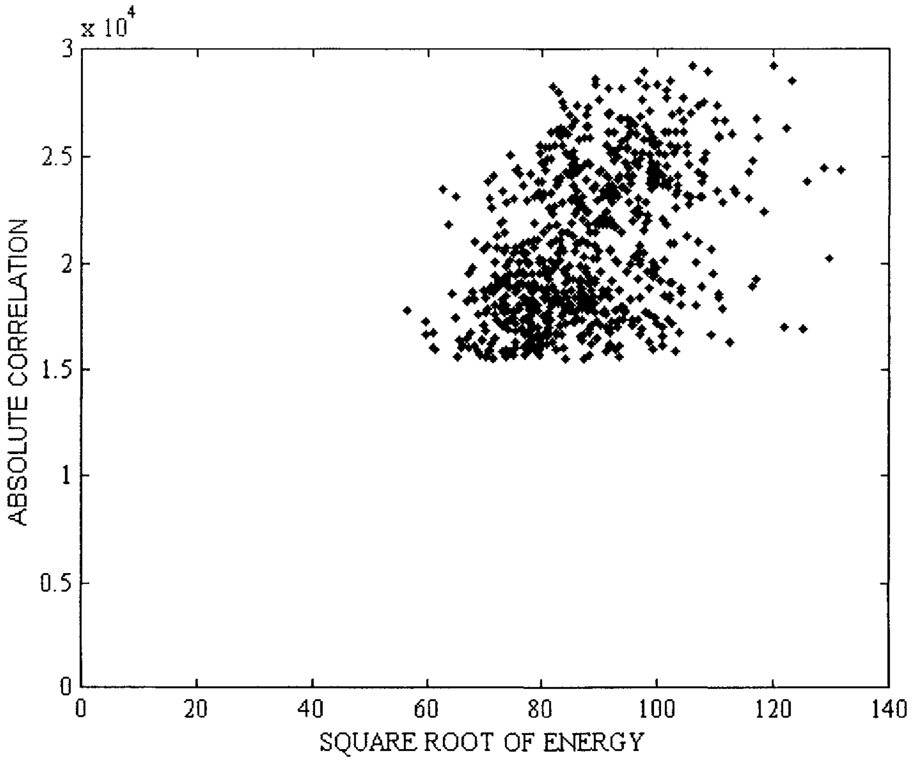
The two thresholds are computed using the following formulas:

$$\begin{aligned} thresh1 &= av2 + \alpha_1(max2 - av2) \\ thresh2 &= av3 + \alpha_2(max3 - av3) \end{aligned} \tag{3.18}$$

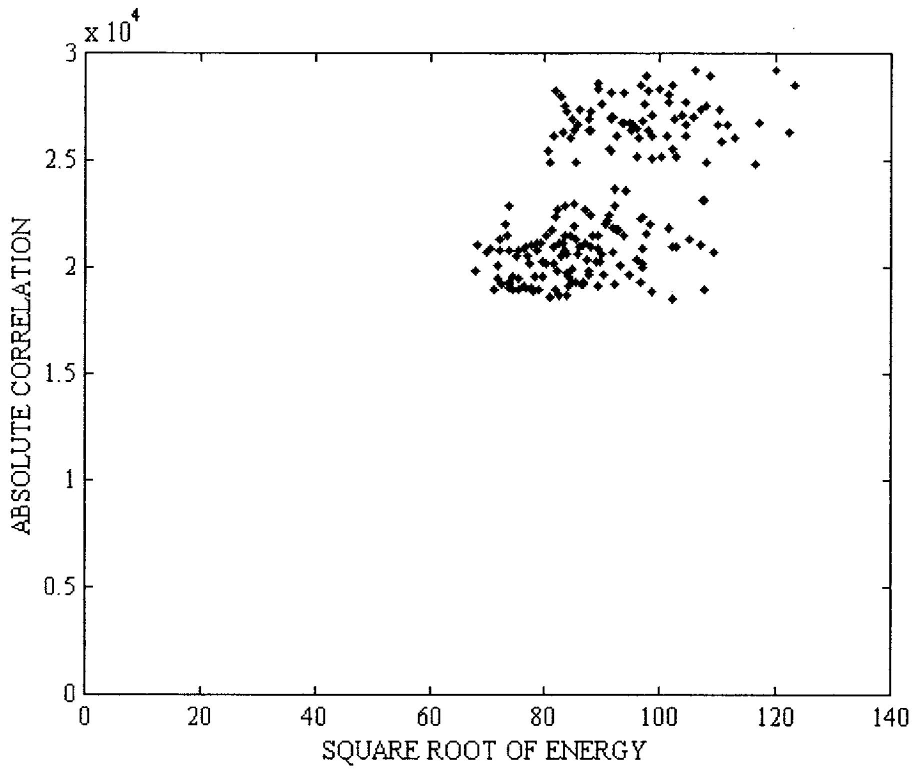
where *av*2, *av*3 are the averages of the absolute values of the correlations values due to the contributions of the first two and the first three pulses respectively, *max*2 and *max*3 are the maxima of the absolute values of the correlations due to the contributions of the first two and the first three pulses respectively, and α_1 and α_2 are the coefficients that control the values of the two thresholds. Therefore, the search space is controlled by α_1 and α_2. If they are small, the search space is large. A major concern is an appropriate choice of the values α_1, α_2 and M to control the search space with little sacrifice in the perceptual quality of the speech. By comparing the results of various combinations of α_1, α_2 and M on a variety of speech samples, it is found that the combination of $\alpha_1 = 0.1$, $\alpha_2 = 0.4$ and $M = 5$ yields a good performance in terms of the speech quality and computation complexity. From the simulation results to be presented later, the average number of combination of pulse positions searched reduces to 178 with the proposed two modifications, and this number is about 40% less than that of G.729A. In terms of the speech quality, the average PESQ score is reduced from 3.726 in G.729A to 3.592 in the proposed scheme; thus, there is almost no degradation in the perceptual quality of the speech. Figure 3.1 shows an example of the codewords searched for the three different search schemes. It can be seen that only a small part of the codewords are searched in the proposed search scheme. These have relatively large absolute correlation values. The winning codeword is most probably located in this region.


x 10^4
3
2.5
2
1.5
1
0.5
0
ABSOLUTE CORRELATION
0
20
40
60
80
100
120
140
SQUARE ROOT OF ENERGY

(a)

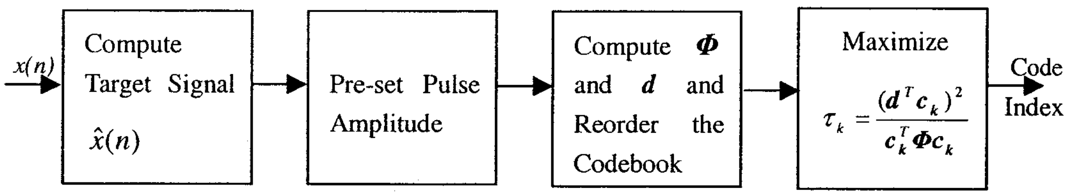

x 10^4
3
2.5
2
1.5
1
0.5
0
ABSOLUTE CORRELATION
0
20
40
60
80
100
120
140
SQUARE ROOT OF ENERGY

(b)



(c)

Figure 3.1 An example of the search space for the different search schemes.
(a) Full search (8192 points) (b) G.729 focused search (736 points)
(c) Proposed focused search (199 points)



x(n)
Compute Target Signal $\hat{x}(n)$
Pre-set Pulse Amplitude
Compute $\boldsymbol{\Phi}$ and $\boldsymbol{d}$ and Reorder the Codebook
Maximize $\tau_k = \frac{(\boldsymbol{d}^T \boldsymbol{c}_k)^2}{\boldsymbol{c}_k^T \boldsymbol{\Phi} \boldsymbol{c}_k}$
Code Index

Figure 3.2 Block diagram illustrating the proposed fixed codebook search

A block diagram illustrating the proposed fixed codebook search process is shown in Figure 3.2. The target signal $\hat{x}(n)$ is first computed according to (3.2). Then, the pulse amplitude is pre-set, the correlation matrix $\boldsymbol{d}$ and the matrix $\boldsymbol{\Phi}$ are computed according to (3.5) and (3.6) respectively. Next, the fixed codebook structure is reordered according to (3.17). Also, the two thresholds *thresh1* and *thresh2* are pre-determined using (3.18). Finally, the optimum positions of the four pulses are obtained by maximizing the value of τ_k given by (3.7), and the code index transmitted.

3.5 Simulation Results

In this section, we first present simulation results for the complexity of the fixed codebook search used in G.729, G.729A and the modified coder (which uses the proposed search scheme incorporated in G.729A). For this purpose, we use 8 Nortel speech samples. These include 4 female and 4 male speech samples in the English language. Each sample contains 6 or 7 sentences, each sample having a length of about 3000 frames. In terms of complexity of the codebook search, the average search space and the execution time on a general PC as a percentage of the overall execution time of the coder, are compared for the three coders G.729, G.729A and the modified G.729A. Tables 3.6 and 3.7 present the simulation results in terms of the average search space and the percentage execution time for the three coders. It can be seen that the average search space for the proposed scheme is less than 60% of that of G.729A. Further, the execution time of the proposed search scheme as a percentage of the overall execution time of the modified coder is substantially reduced in comparison to the corresponding quantity for G.729A.

Next, we present simulation results concerning the speech quality for the three coders. When evaluating the speech quality in the time domain, two objective

TABLE 3.6 COMPARISON OF THE AVERAGE SEARCH SPACE

Speech sample \ Search scheme	G.729	G.729A	Proposed
English (Female)	1025	320	174
English (Male)	1027	320	182

TABLE 3.7 COMPARISON OF THE EXECUTION TIME AS A PERCENTAGE OF THE OVERALL EXECUTION TIME OF THE CODER FOR THE THREE SEARCH SCHEMES

Speech sample \ Coder	G.729	G.729A	Modified
English (Female)	31.4%	12.7%	8.5%
English (Male)	34.1%	12.5%	8.2%

TABLE 3.8 COMPARISON OF THE SEGMENTAL SNR IN dB

Speech sample \ Coder	G.729	G.729A	Modified
English (Female)	8.3949	7.9814	8.0116
English (Male)	8.2486	7.7992	7.8571

distortion measures are generally used, namely, the *SNR* and *SEGSNR*. If *s(n)* is the original speech sample, $\hat{s}(n)$ is the coded speech sample and the speech file has N_T samples, then the *SNR* is defined as

$$SNR = 10\log_{10}\frac{\sum_{n=0}^{N_T-1} s^2(n)}{\sum_{n=0}^{N_T-1}(s(n)-\hat{s}(n))^2} \qquad (3.19)$$

The *SNR* measures the quality after decoding the entire speech signal and hence, the segmental detail of the quality of the speech signal is not measured. However, the *SEGSNR* overcomes this disadvantage. Suppose the original speech signal *s(n)* has N_f number of frames each of length N_s and the decoded speech signal is $\hat{s}(n)$, then the *SEGSNR* is defined by

$$SEGSNR = \frac{1}{N_f}\sum_{i=0}^{N_f-1} 10\log_{10}\frac{\sum_{j=0}^{N_s-1} s^2(N_s i + j)}{\sum_{j=0}^{N_s-1}(s(N_s i + j)-\hat{s}(N_s i + j))^2} \qquad (3.20)$$

The *SEGSNR* measures the speech quality of the decoded signal frame by frame and takes the average over all the frames. This measure has a better correspondence to the auditory quality of speech [2]. However, the *SEGSNR* is not a good measure when an entire frame is almost silent. These types of frames yield large negative *SEGSNR*, which is not a true indication of the overall performance. To overcome this problem, such silent frames are discarded from the speech sample.

In order to compare the speech quality of the three coders, the *SEGSNR* for each of these coders is determined by simulation studies using the same 8 Nortel speech samples mention above. Table 3.8 presents the *SEGSNR* for the three coders. It is seen from this table that the *SEGSNR* is somewhere between those of G.729 and G.729A. Thus, if *SEGSNR* is used as a measure of the speech quality, then the performance of the modified coder is somewhat superior to that of G.729A.

Although the *SEGSNR* is a better measurement of the speech quality than the *SNR*, it still cannot accurately reflect the perceptual speech quality of a CELP speech coder [2]. Therefore, an objective method is used to estimate the speech quality and is contained in the ITU-T recommendation P.862 [4]. This standard uses a score for the purpose of the perceptual evaluation of the speech quality (PESQ), and has a correlation of 0.935 with the MOS score. Thus, it can accurately reflect the perceptual speech quality of a speech coder. Based on this standard, the speech qualities of G.729, G.729A and the modified coder are evaluated using 16 Nortel speech samples, consisting of the 8 speech samples mentioned earlier, and 4 female and 4 male speech samples in the Chinese language. Each sample contains 6 or 7 sentences, each sample having a length of about 3000 frames. The PESQ scores for each group of the speech samples are given in Tables 3.9 to 3.12, and the average scores over all the 16 speech samples in Table 3.13. It is seen from Table 3.13 that the PESQ score for the modified coder is reduced only slightly compared to that of G.729A. However, the average PESQ score for the modified coder is 3.6, which is above the minimum level of acceptable speech quality.

TABLE 3.9 THE PESQ SCORE FOR THE ENGLISH LANGUAGE SPEECH SAMPLES (FEMALE SPEAKERS)

Coder / Speech sample	G.729	G.729A	Modified
Feng0000	3.627	3.551	3.376
Feng0100	3.716	3.789	3.574
Feng0200	3.763	3.677	3.541
Feng0300	3.714	3.626	3.500
Average score	3.705	3.661	3.494

TABLE 3.10 THE PESQ SCORE FOR THE ENGLISH LANGUAGE SPEECH SAMPLES (MALE SPEAKERS)

Coder / Speech sample	G.729	G.729A	Modified
Meng0000	3.487	3.475	3.329
Meng0100	3.907	3.847	3.709
Meng0200	3.889	3.823	3.674
Meng0300	3.934	3.844	3.801
Average score	3.804	3.747	3.628

TABLE 3.11 THE PESQ SCORE FOR THE CHINESE LANGUAGE SPEECH SAMPLES (FEMALE SPEAKERS)

Coder / Speech sample	G.729	G.729A	Modified
Fchi0000	3.701	3.626	3.475
Fchi0100	3.717	3.664	3.535
Fchi0200	3.727	3.647	3.533
Fchi0300	3.789	3.707	3.597
Average score	3.734	3.661	3.535

TABLE 3.12 THE PESQ SCORE FOR THE CHINESE LANGUAGE SPEECH SAMPLES (MALE SPEAKERS)

Speech sample \ Coder	G729	G729A	Modified
Mchi0000	3.935	3.882	3.786
Mchi0100	3.834	3.746	3.602
Mchi0200	3.894	3.863	3.707
Mchi0300	3.917	3.844	3.734
Average score	3.895	3.834	3.707

TABLE 3.13 THE AVERAGE PESQ SCORE FOR THE16 SPEECH SAMPLES

PESQ \ Coder	G729	G729A	Modified
Average PESQ score	3.785	3.726	3.592

3.6 Conclusion

In this chapter, the fixed codebook search schemes used in G729 and G729A have been described in detail and a new search scheme proposed to reduce the search complexity. In the proposed search scheme, two thresholds have been introduced to reduce the size of the search space for the codebook. The codebook structure is reordered based on the parameter τ_{m_i}, that depends on the correlation signal and the elements $\phi(m_i, m_i)$ of the covariance matrix $\boldsymbol{\Phi}$. The average search space has been reduced significantly by focusing only on the pulse positions with large values of τ_{m_i} and large values of the correlation signal $d(m)$. Simulations concerning the search complexity and speech quality have been carried out on a general PC using speech samples from Nortel Net works for the three different fixed codebook search schemes, namely those of G.729 and G.729A, and the proposed one. The simulation results indicate that, on average, the search space for the proposed scheme is less than 60% of that of G.729A. However, in terms of the perceptual speech quality, the PESQ score for the modified coder is slightly lower than that of G.729A. The final PESQ score of the modified coder is 3.6, which is still above the minimum level of acceptable speech quality. In the next chapter, we will deal with the results of implementation of G.729A and the modified coder on a DSP board.

Chapter 4

Implementation and optimization on TMS320VC5416

This chapter describes optimized implementations of G.729A and the modified coder on the TMS320VC5416 DSP board. The main purpose of this chapter is to implement the two speech coders on this board and use the implementation results to compare the average run-time for the two coders. This chapter contains two major parts. First, the implementation based on the host channel is tested and the run-time of the initial implementations of the two coders obtained. Then, two kinds of optimizations are carried out to reduce the run-time of the initial implementations. The chapter is organized as follows. Section 1 is an introduction of the basic features of the C5416 DSP board. Section 2 illustrates, in detail, the implementation on the board. Section 3 describes the optimization procedures. Section 4 gives a summary of the implementations and the optimization results.

4.1 Introduction to the TMS320VC5416 Board

The TMS320VC5416 DSK is a 16-bit fixed-point digital signal processor in the TMS320 DSP family [35,36,38,39], which is used by DSP designers to develop various DSP applications. The C5416 DSP board includes a C5416 based target hardware and Code Composer Studio (CCS) software. The C5416 has a processing speed of 160 MHz. Following is a brief review of its basic structure.

(1) Central Processing Unit (CPU)

The C5416 CPU contains a 40-bit arithmetic logic unit (ALU), which includes a

40-bit barrel shifter, two independent 40-bit accumulators, a 17×17-bit multiplier coupled with a 40-bit adder for a single cycle MAC (multiply/accumulate) operation, a compare, select and store unit (CSSU) for the add/compare selection of the Viterbi operator, an exponent encoder (EXP) to compute the exponent of a 40-bit accumulator in a single cycle, a data address generation unit (DAGEN), and a program address generation unit (PAGEN) [36].

(2) Bus structure

The C5416 architecture is built around four pairs of 16-bit buses, in which each pair consists of an address bus and a data bus. These are the program bus pairs and three data bus pairs. The program bus pair carries the instruction code from the program memory. The three data bus pairs denoted by (CAB, CB), (DAB, DB), and (EAB, EB) interconnect the various units within the CPU. In addition, the pairs (CAB, CB) and (DAB, DB) are used to read from the data memory, whereas the pair (EAB, EB) carries the data to be written to the memory.

(3) Memory

The C5416 provides a 16K on-chip ROM, a 64K on-chip single access RAM (SARAM), a 64K dual access RAM (DARAM) and 256K words of flash ROM. The memory includes the program memory and data memory. The program memory space contains the instructions to execute and the tables used in the execution. The data memory space stores the data used by the instructions. The RAM is always mapped into the data space, but may also be mapped into the program space when the OVLY (overlay) bit is set to 1. The ROM is generally mapped into the program space, but can also be mapped into data space when the DROM (data ROM) bit is set to 1. However, on C5416, the ROM can only be used as the program space. Figure 4.1 presents the memory maps for the C5416 that is used in this project.

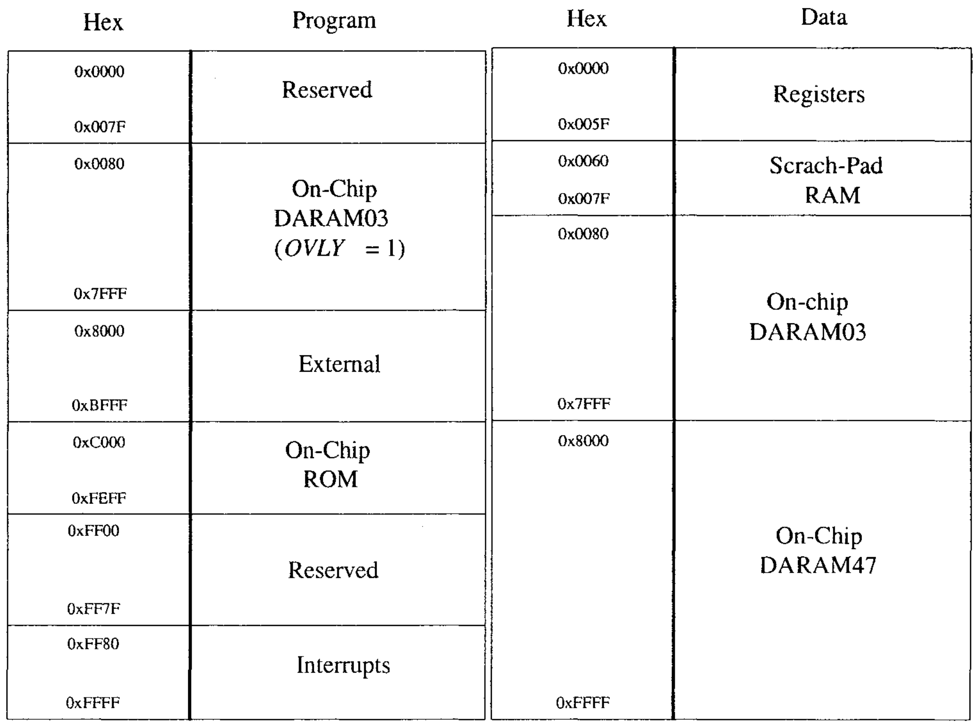


Figure 4.1 Memory maps for the C5416

4.2 Implementation on C5416

This section presents in detail the implementation of the G.729A and the modified coder on the C5416 DSP board. First, the mechanism of loading the coder to the C5416 DSP board is introduced. Then the method for collecting the statistical information of the program is described. Finally, the initial implementation results of both the coders are presented.

4.2.1 Loading the Program to TMS320VC5416 Board

Data transfer is essential for any digital signal processing application. The TI DSP/BIOS (basic input-output system) kernel provides basic runtime services to manage the data transfer [34]. The host channel control and the pipe module handle the input/output for DSP/BIOS applications. The DSP/BIOS pipes are used to buffer the streams of the program input and output data. The data transfer is scheduled through the use of the DSP/BIOS software interrupts (SWI). In this project, the host channel control module and the pipe module are used to exchange the data between the host PC and the target DSP board, and the statistics objects (STS) module employed to gather the run-time information of the different subroutines. We now describe the procedure to transfer the data between the host PC and the target DSP board using the host channel.

In order to understand how the host channel works, we look into the pipe object (Figure 4.2), which is managed by a pipe module and used to internally implement the host channel. Pipes are designed to manage the I/O block. Each pipe object has a buffer divided into a fixed number of frames, and each frame is set to a fixed length. In this project, the frame length is set to 80 samples. This is for the purpose of real time implementation when the sampling rate is 8 KHz and frame length is 10 ms. Figure 4.2 illustrates the two ends of the pipe [34].

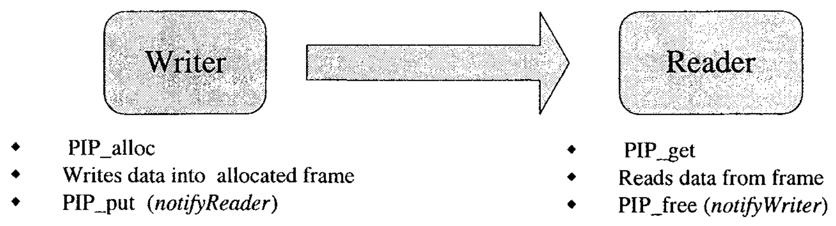


Figure 4.2 The two ends of a pipe

At the writer end, frames of data are written into the pipe, and at the reader end, frames of data are read out from the pipe. The data notification functions, namely *notifyReader* and *notifyWriter*, are employed to synchronize the data transfer. These functions are triggered when a frame of data is written or read to notify the program that the data is available or a frame is free. To read a full frame from a pipe, the following procedures are performed [34]:

- The program first checks the number of full frames available to be read from the pipe. The program runs the function *PIP_getReaderNumFrames* to return the number of full frames in a pipe object.
- If the number of full frames is greater than 0, the function then calls *PIP_get* to get a full frame from the pipe.
- Before returning from the *PIP_get* call, DSP/BIOS checks whether there are additional full frames available in the pipe. If so, the *notifyReade*r function is called at this time.
- Once *PIP_get* returns, the data in the full frame can be read by the

application.

The procedures to write a frame to the pipe are quite similar. When a frame of data is read out from the pipe, the reader end calls *PIP_free* to clear this frame to the pipe. Then, the *notifyWriter* function is triggered to notify the program that a frame is free and the writer end should call *PIP_alloc* to allocate the next frame.

The host channel works in a way similar to that of the pipe module, when it is employed to transfer data between the host PC and the DSP target. Figure 4.3 is the block diagram of our test system.

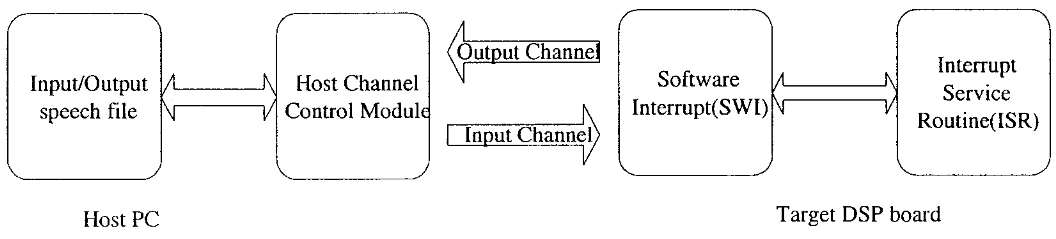
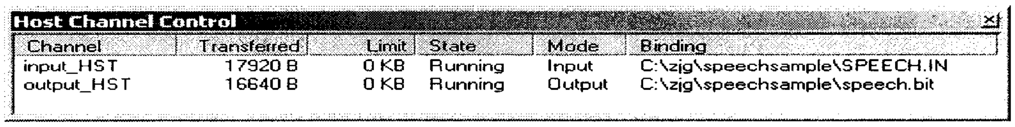


Figure 4.3 Block diagram of the data transfer mechanism between the PC and the DSP board

It can be seen from Figure 4.3 that the input and output channels are controlled by the host channel control (HST) module and the software interrupt (SWI) module. The host channel control module is used to manage the input and output streams. The input streams read the data from the PC to the target DSP board. The output streams transfer the data from the target board to the host PC. Each host channel is internally implemented using a pipe object. To use a particular host channel, the program uses *HST_getpipe* to get the corresponding pipe object and then transfers the data by calling the *PIP_get* and *PIP_free* operations for the input or *PIP_alloc* and *PIP_put* operations for the output. In the configuration file, two new channels are added in the host channel control module, and these are the *input_HST* and *output_HST*. Then, setting the properties of these two channels and binding the speech files in the host PC to the corresponding channels are carried out. A speech sample file is used as the input signal to the program. This file is stored in the host PC and is bound to the input channel. Another speech file, which is created in the host PC, is bound to the output channel to record the synthesized output speech. An example of binding the input/output files to the corresponding channels is shown in Figure 4.4.



Host Channel Control

Channel	Transferred	Limit	State	Mode	Binding
input_HST	17920 B	0 KB	Running	Input	C:\zjg\speechsample\SPEECH.IN
output_HST	16640 B	0 KB	Running	Output	C:\zjg\speechsample\speech.bit

Figure 4.4 Binding the input/output files to the input/output channels

As illustrated in Figure 4.3, the HST controls one end of the pipe and the SWI controls the other end of the pipe. The SWI module in DSP/BIOS provides a software interrupt capability. It is programme-triggered through a call to a DSP/BIOS API (application program interface) [37]. The SWI is trigged every time there is an empty frame in the output channel and a full frame of data in the input channel. In the C5416 DSP board, there is a mailbox for each SWI object, which is used to determine whether or not to post the software interrupt. Since there are two events that must happen together to trigger the SWI, the *SWI_andn* API is used to trigger the SWI. As illustrated in Figure 4.5 [37], the *SWI_andn* function treats the mailbox as a bit mask. When the input channel contains a full frame, it notifies the *SWI_andn* to clear the second smallest bit of the mailbox; when the output channel contains an empty frame, it uses *SWI_andn* to clear the smallest bit of the mailbox. When the value of its mailbox becomes 0, the SWI is posted. Therefore, the SWI is posted only when there is a full frame in the input channel and an empty frame in the output channel. After the SWI is posted, the interrupt service routine (ISR) is executed. In this project, the G.729A coder and the modified coder are configured as the interrupt service routines.

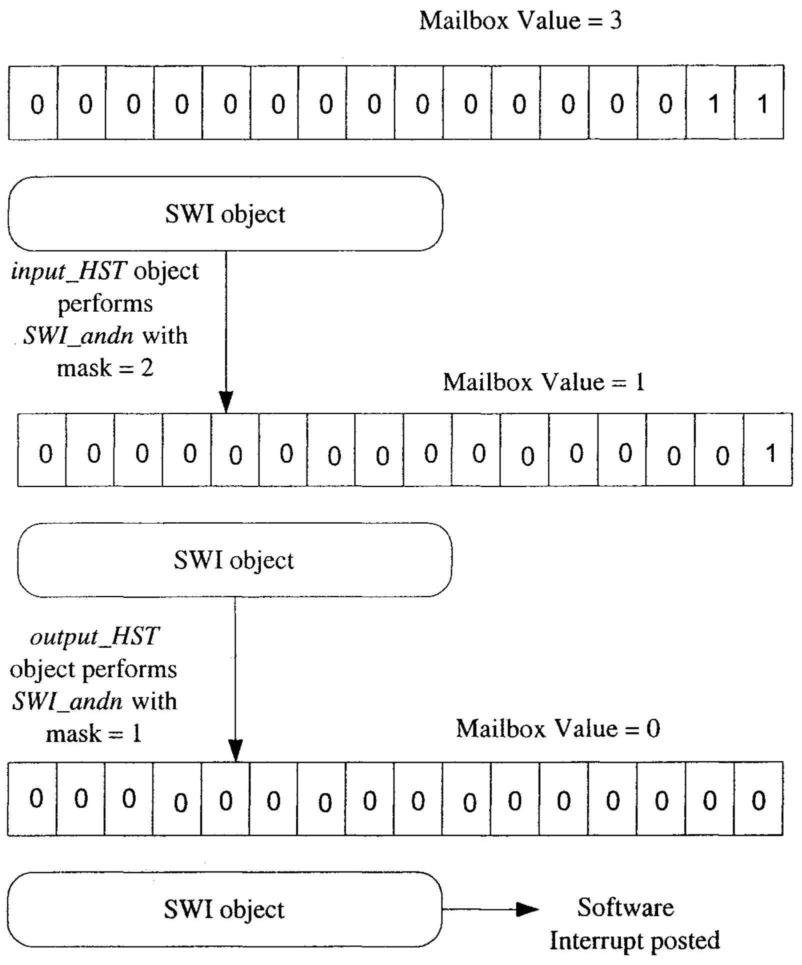


Figure 4.5 The *SWI_andn* mechanism

Using the host channel is an easy way to deal with the data transfer between the host and the target board. During the initial implementation of a DSP algorithm, it can be used to compare the output data file with the expected result. Later, when the implementation result is the same as expected, the program is modified, since the host channel cannot be used for a real-time implementation.

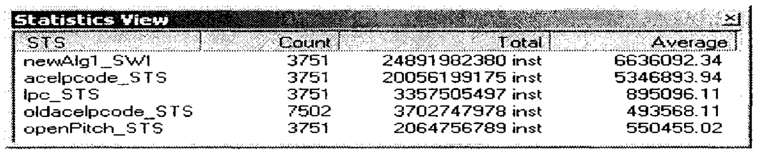
4.2.2 Statistics Object Manager (STS Module)

DSP/BIOS statistics objects are used to track the number of the CPU cycles used by the various routines during the execution. This module manages the statistics objects, which store the key statistics while a program runs. Using this tool, we can gather useful information, such as the number of cycles needed to execute a particular function and the number of times the function is called. The *STS* module can obtain the information for a software interrupt automatically, and is called implicit instrumentation. However, it can also gather the information for the different parts of the algorithm by using a pair of functions *STS_set* and *STS_delta*, and is called explicit instrumentation. Here is how these two functions are used:

```
STS_set(&stsObj, CLK_gethtime());
        ' do routine '
STS_delta(&stsObj, CLK_gethtime());
```

STS_set saves the value of the *CLK_gethtime* as the contents of the previous value field (set value) in the STS object. *STS_delta* subtracts this set value from the new value passed from the second *CLK_gethtime*. The result is the difference between the time recorded before the routine started and after it was completed, that is, the time it took to execute the routine. The host can display the number of times the routine was performed, the total time taken to perform the routine, and the average time. In this project, the *STS* module is used to collect the information for a few major subroutines in the coding process, namely, the LPC analysis, pitch search and the fixed codebook search. Figure 4.6 shows an example of the *Statistics View* window. The statistical information shown in the first line corresponds to the execution of the software interrupt named newAlg1_SWI, which is the modified coder in our project. The information shown in the other lines of the *Statistics View* window is an example of the explicit instrumentation, which corresponds to a few of the essential subroutines in the modified coder. It is noted that the result is shown in run cycles. It can be converted into run-time using the formula

$$T = \frac{number \quad of \ cycles \ \times 1000}{160 \ \times 10^{6}} \ ms$$


Statistics View

STS	Count	Total	Average
newAlg1_SWI	3751	24891982380 inst	6636092.34
acelpcode_STS	3751	20056199175 inst	5346893.94
lpc_STS	3751	3357505497 inst	895096.11
oldacelpcode_STS	7502	3702747978 inst	493568.11
openPitch_STS	3751	2064756789 inst	550455.02

Figure 4.6 An example of *Statistics View* window

4.2.3 Implementation Results

Since this implementation is bit-exact with the ITU-T standard, the output speech file is the same as that obtained on a general PC. Therefore, it is not necessary to have a comprehensive test of the speech quality on a number of speech samples, as has been done in Chapter 3. The speech sample used in this test is from the G.729A standard. It has a length of 3751 frames. Table 4.1 illustrates the average run-time for the G.729A and a few of its subroutines, and Table 4.2 the corresponding results for the modified coder. In these tables, the average run-time is the total run-time divided by the number of frames in the speech sample. It can be seen from these tables that the average run-time of the modified fixed codebook search is reduced by about 30% compared to that of the search algorithm in G.729A. However, the average run-time of the two coders are still far beyond the real-time requirement of being able to process one frame in 10 ms. Therefore, some optimization procedures are carried out in the next section to reduce the run-time of the two coders.

TABLE 4.1 IMPLEMENTATION RESULTS OF THE G.729A CODER

STS Object	Total run-time for all the frames (ms)	Average run-time /frame (ms)
G.729A	165081.51	44.01
Fixed Codebook Search	32521.17	8.67
LPC Analysis	21005.6	5.60
Open Loop Pitch Search	12903.44	3.44

TABLE 4.2 IMPLEMENTATION RESULTS OF THE MODIFIED CODER

STS Object	Total run-time for all the frames (ms)	Average run-time /frame (ms)
Modified Coder	155591.48	41.48
Fixed Codebook Search	23106.16	6.16
LPC Analysis	21005.6	5.60
Open Loop Pitch Search	12903.44	3.44

4.3 Optimization of the C Code

In order to reduce the run-time of the two coders, some optimization procedures are employed. The optimization procedures consist of compiler optimization and replacing the basic operations with C5416 intrinsic functions.

4.3.1 Using C5416 DSK compiler optimization

The C compiler is able to perform various optimizations. The compiler tools include an optimization program that improves the execution speed by performing such tasks as simplifying loops, rearranging statements and expressions, and allocating variables into registers [40]. High-level optimizations are performed in the optimizer and low-level, target-specific optimizations occur in the code generator (Figure 4.7).

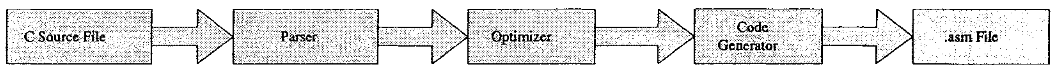


Figure 4.7 Compiling C program using optimizer [40]

There are four options to control the file-level optimization when compiling the program. These four options are:

-o0 Optimizes register usage, eliminates the unused code and simplifies the expressions and statements.

-o1 Uses **–o0** optimizations and optimizes locally.

-o2 Uses **–o1** optimizations and optimizes globally.

-o3 Uses **–o2** optimizations, removes all the functions that are never called, simplifies functions with return values that are never used.

In our project, the **–o3** option is used and is combined with other compiling options such as the **–pm** and **–op** options. The **–pm** option is used when compiling multiple source files. The **–op** option is used to control the program level optimization. It indicates as to whether the functions in the other modules can call a module's external functions or modify the module's external variables. When this program level optimization is used, the compiler performs several optimizations that are rarely used during file-level optimizations [40]. These optimizations include the following: (a) if a particular argument in a function always has the same value, the compiler replaces the argument with the value and passes the value instead of the argument, (b) if a return value of a function is never used, the compiler deletes the return code in the function, and (c) if a function is not called directly or indirectly by the main function, the compiler removes the function. There are four possibilities that can be selected for the **–op** option:

-op0 specifies that the module contain functions and variables that are called or modified from outside the source code provided to the compiler.

-op1 specifies that the module contains variables modified from outside the source code provided to the compiler but does not use functions called from outside the source code.

-op2 specifies that the module contains neither functions nor variables that are called or modified from outside the source code provided to the compiler.

-op3 specifies that the module contains functions that are called from outside the source code provided to the compiler, but does not use variables modified from outside the source code.

In this project, since the modules of the algorithm contain neither functions nor variables that are called or modified from outside the source code, the **-op2** option is used in our program. Therefore, the option for the compiler optimization is a combination of **–o3**, **-pm**, and **–op2**.

4.3.2 Using Intrinsic Functions of C5416 to Replace C Functions

Code Composer Studio (CCS) provides some intrinsic functions such as calling functions in C/C++ and produce assembly language, that can be used [40]. The compiler in C5416 recognizes a number of intrinsic functions, which can be conveniently used to reduce the execution cycles for these functions. It is very convenient to use the intrinsic operators in the C program and get a more efficient code at the same time. Most of the intrinsic functions in C5416 have the same formula as the C functions in the G.729A, and the overflow control mechanism is also the same. Therefore, most of the functions in the *basic_op.c* of the modified coder are replaced with the corresponding intrinsic functions in C5416. Table 4.3 lists all the C-functions and the corresponding intrinsic functions replacing them. Since the intrinsic functions are written in the assembly language, the program after using the intrinsic functions is much more efficient than the original C program.

TABLE 4.3 INTRINSIC FUNCTIONS USED IN THE MODIFIED CODER

Function in basic_op.c	Intrinsic Function	Description
Word16 abs_s(word16 var1)	Short _abs(short src)	Creates a 16-bit absolute value
Word32 L_abs(word32 L_var1)	Long _labs(long src)	Creates a 32-bit absolute value
Word16 norm_s(word16 var1)	Short _norm(short src)	Produces the number of left shifts needed to normalize src
Word16 norm_l(word32 L_var1)	Short _norm(long src)	Produces the number of left shifts needed to normalize src
Word16 add(word16 var1, word16 var2)	Short _sadd(short src1, short src2)	Adds two 16-bit integers, producing a saturated 16-bit result
Word32 L_add(word32 var1, word32 var2)	long _lsadd(long src1, long src2)	Adds two 32-bit integers, producing a saturated 32-bit result
Word32 L_mac(word32 var3, word16 var1, word16 var2)	Long _smac(long src, short op1, short op2)	Multiplies op1 and op2, shifts the result left by 1, and adds it to src, produces a saturated 32-bit result
Word32 L_msu(word32 L_var3, word16 var1, word16 var2)	Long _smas(long src, short op1, short op2)	Multiplies op1 and op2, shifts the result left by 1, and subtracts it from src, produces a saturated 32-bit result
Word16 mult(word16 var1, word16 var2)	Short _smpy(short src1, short src2)	Multiplies src1 and src2, and shift the result left by 1. Produces a saturated 16-bit result
Word32 L_mult(word16 var1, word16 var2)	long _lsmpy(long src1, long src2)	Multiplies src1 and src2, and shift the result left by 1. Produces a saturated 32-bit result
Word16 negate(word16 var1)	Short _sneg(short src)	Negates the 16-bit value with saturation. _sneg(0xffff8000) =>0x00007ffff
Word32 L_negate(word32 L_var1)	long _lsneg(long src)	Negates the 32-bit value with saturation. _sneg(0x80000000) =>0x7fffffff
Word16 sub(word16 var1, word16 var2)	Short _ssub(short src1, short src2)	Subtracts src2 from src1. Producing a saturated 16-bit result
Word32 L_sub(word32 L_var1, word32 L_var2)	long _ssub(long src1,longsrc2)	Subtracts src2 from src1. Producing a saturated 32-bit result

The optimized implementation results of the two coders are presented in Tables 4.4 and 4.5. All the results contained in Tables 4.1, 4.2, 4.4 and 4.5 are summarized in Table 4.6. From this table, it is clear that the optimization process has been very effective in that there is an 81% reduction in the runtime of the codebook search algorithms of the two coders and a 78% reduction in the runtime of the coders themselves. Finally, it is observed that the average runtime for the proposed codebook search algorithm is reduced by 30% and the runtime for the modified coder by 6% compared to the corresponding run times for the G.729A coder.

TABLE 4.4 OPTIMIZATION RESULTS OF THE G.729A CODER

STS Object	Total run-time (ms) for all the frames	Average run-time / frame (ms)
G.729A	35709.52	9.52
Fixed Codebook Search	6114.13	1.63
LPC Analysis	5664.01	1.51
Open Loop Pitch Search	2513.17	0.67

TABLE 4.5 THE OPTIMIZATION RESULTS OF THE MODIFIED CODER

STS Object	Total run-time (ms) for all the frames	Average run-time / frame (ms)
Modified Coder	33796.51	9.01
Fixed Codebook Search	4201.12	1.12
LPC Analysis	5664.01	1.51
Open Loop Pitch Search	2513.17	0.67

4.4 Summary

In this chapter, the G.729A and the modified coder have been successfully implemented on the TMS320VC5416 platform, and the initial implementation results further optimized to reduce the run-time of the two coders. It has been found that the run-time for the proposed fixed codebook search scheme has been reduced by about 30% compared to that of the G.729A search scheme. This result should be compared to that obtained in Chapter 3, wherein the average search space for the proposed fixed codebook search scheme was 40% less than that of G.729A when both of the codebook search algorithms were run on a general PC.

On an average, it takes about 9 ms to run the optimized modified coder on C5416 DSP platform. However, in terms of the worst case, the run-time is slightly larger than 10 ms. Furthermore, in real time implementation, the coding and decoding process should integrate with other algorithms, such as the VAD and echo cancellation algorithms, to get a better bandwidth efficiency and better speech quality. Therefore, further work needs to be done on the program to meet the real time implementation requirement. For the coding process, most of the time-consuming signal processing functions should be written in assembly language instead of using only the intrinsic functions.

TABLE 4.6 COMPARISON OF THE RUN-TIMES (MS) OF THE TWO CODERS

SWI / Function	**G.729A**			**Modified Coder**		
	Before Optimizations	After Optimizations	Reduced by (%)	Before Optimizations	After Optimizations	Reduced by (%)
Total coding process	44.01	9.52	78	41.48	9.01	78
Fixed codebook search	8.67	1.63	81	6.16	1.12	82
LPC Analysis	5.60	1.51	73	5.60	1.51	73
Open Loop Pitch Search	3.44	0.67	80	3.44	0.67	80

Chapter 5

Conclusion and Future Research

5.1 Conclusion

Low-bit-rate speech coders are widely employed in most of the existing packet voice communication systems to improve bandwidth efficiency, and there is a high demand for low complexity speech coders. In this thesis, our research has focused on further reducing the complexity of the standard speech coder, namely, G.729A. Since the fixed codebook search accounts for about 30% of the total coding process, a fixed codebook search algorithm has been proposed to replace the existing search scheme in G.729A. In the proposed fixed codebook search scheme, two modifications have been incorporated in G.729 using a focused search method. First, the fixed codebook structure has been reordered and the search focused on the first five pulse positions in each track. Then, two thresholds, which depend on the average and maximum absolute values of the correlation values of the first two and the first three pulses, have been defined. By appropriately choosing these parameters that control the values of the two thresholds, the average codebook search space has been reduced significantly, without much degradation in the perceptual speech quality. The simulation results on a general PC indicate that the average codebook search space for the proposed algorithm of the modified codec is reduced about 40% compared to that of G.729A. As to the perceptual speech quality, the PESQ score is reduced to 3.592 from 3.726 for the G.729A coder. This score is still above the minimum level of acceptable speech quality.

Another contribution of the thesis is the implementation of the G.729A and the modified speech coder on the TI 16-bit fixed point TMS320VC5416 DSP board. The initial implementation results of the two coders have been optimized to further reduce the run-time of the speech coders. The implementation results show that the run time of the proposed fixed codebook search scheme is reduced by about 30% compared to that of the G.729A codec, thus confirming the effectiveness of the proposed fixed codebook search scheme. According to the optimized results on the C5416 platform, the final run-time for the modified coder has been reduced to 9.01 ms from 9.52 ms of that of the G.729A codec.

5.2 Scope for Future Research

Further work could be carried out on further optimizing the C code for real time implementation. Although considerable reduction of the run-time has been achieved by using the intrinsic functions of the DSP board, it is not the most efficient way. It is better to use the assembly code to rewrite those functions that consume heavy computation. In G.729A speech coder, three of the signal processing filters are used frequently and consume much of the computation. These three filters are *Residu( )*, *Convolve( )* and *Syn_filt( )*. In order to further reduce the run-time of the speech coder, these filters must be implemented using assembly language and optimized. Finally, other algorithms such as the voice activity detection (VAD) and echo cancellation need to be combined with the speech coder to enhance the bandwidth efficiency and the speech quality.

REFERENCES

[1] M.N.S. Swamy, M. O. Ahmad, R. Khatibi, "Low Bit Rate Speech Coding", Internal report, Concordia University, February, 2000.

[2] W.B. Kleijn and K.K. Paliwal, *Speech Coding and Synthesis*. Elsevier, 1995.

[3] *Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 Kb/s*, ITU-T G.723.1, March, 1996.

[4] *Perceptual evaluation of speech quality(PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs*, ITU-T recommendation P.862, Feb, 2001.

[5] *Coding of Speech at 8 Kb/s Using Conjugate-Structure Algebraic Code-Excited Linear-Prediction (CS-ACELP)*, ITU-T Recommendation G.729.

[6] D. O'Shaughnessy, *Speech Communications—Human and Machine*, 2nd edition, IEEE Press, 2000.

[7] A.S.Spanias, "speech coding: A tutorial review," in *Proc. IEEE*, vol.82, pp.1541-1582, Oct.1994

[8] G. Kubin, B.S. Atal, and W.B. Kleign, "Performance of noise exitation for unvoiced speech," *Proc.IEEE* Workshop on speech coding for Telecom. Pp.35-36, Oct. 1993.

[9] I.A. Atkinson, A.M. Kondoz, and B.G. Evans, "Time envelope vocoder, a new

LP based coding strategy for use at bit rates of 2.4 Kb/s and below," *IEEE J. Selected Areas Communication.*, vol.13, pp.449-457, Feb. 1995.

[10] I.A. Atkinson, A.M. Kondoz, and B.G Evans, "Time envelope LP vocoder: A new coding technique at very low bit rates," *Proc. European Conf. on speech Communication and Technology(Madrid)*, pp. 241-244, Sept. 1995.

[11] J.-H. Chen, R.V. Cox, Y.-C. Lin, N. Jayant, and M.J. Melchner, "A low delay CELP coder for the CCITT 16 Kb/s speech coding standard," *IEEE J. Selected Areas Communication.*, vol. 10, pp.830-849, June 1992.

[12] *Reduced complexity 8 Kb/s CS-ACELP speech codec*, ITU-T G.729 Annex A, Nov,1996.

[13] A. McCree, K. Truong, E.B. George, T.P. Barnwell III, and V. Viswanathan, "A 2.4 Kb/s MELP coder candidate for the new U.S. Federal Standard," *Proc. IEEE Int. Conf. on Acoustics, Speech, Signal Processing(Atlanta)*, pp. 200-203, May 1996.

[14] J. Makhoul "Linear prediction: A tutorial review," *Proc. IEEE.* 63, 561-580.

[15] M.R. Schroeder and B.S. Atal, "Code-excited linear prediction(CELP): High quality speech at very low bit rates," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing, Tampa, FL*,1985, pp. 937-940.

[16] F. Itakura, "Line spectrum representation of linear predictive coefficients," *J. Acoust. Soc. Amer.*, vol. 57, Suppl. no. 1, p. S35, 1975.

[17] Redwan. Salami et al., "Description of the proposed ITU-T 8 Kb/s speech coding standard," in *Proc. IEEE* Workshop on Speech Coding, 1995, pp.3-4.

[18] A. Kataoka, J. Ikedo, and S. Hayashi, "LSP and gain quatization for the proposed ITU-T 8 Kb/s speech coding standard," in *Proc. IEEE* Workshop on Speech Coding, 1995, pp.7-8.

[19] D.Florencio, "Investigating the use of asymmetric windows in CELP vocoders," in *Proc. ICASSP, Minneapolis, MN*, 1993, pp. II427-II430.

[20] Y. Tohkura, F. Itakura, and S. Hashimoto, "spectral smoothing technique in PARCOR speech analysis-synthesis," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-26, pp.587-596,1978

[21] M. R. Schroeder, B. S. Atal, and J. L. Hall, "Optimizing digital speech coders by exploiting masking properties of the human ear," *J. Acoust. Soc. Amer.*, vol 66, pp. 1647-1652, 1979.

[22] Redwan Salami et al., "ITU-T G.729 Annex A: Reduced complexity 8 Kb/s CS-ACELP Codec for Digital Simultaneous Voice and Data" *IEEE Communications Magazine*, Sept, 1997. pp 56-63.

[23] J-P Adoul, P.Mabilleau,M.Delprat and S.Motissette. "Fast CELP Coding based on Algebraic codes", *ICASSP*, April,1987. p49.4.1

[24] W.Bastiaan Kleijn, Daniel J.Krasinski and Richard H. Ketdum. "Fast Methods

for the CELP Speech Coding Algorithm" *IEEE Trans on ASSP*, Vol 38,No.8.August 1990,p1330-1341.

[25] A.Kataoka, Takehiro Moriya, Shinji Hayashi "An 8 Kb/s Conjugate Structure CELP (CS-CELP) Speech Coder", *IEEE Trans on ASSP*, Vol 4, No.6. November,1996, p401-411.

[26] C.Laflamme, J-P Adoul,H.Ysu,and S.Morissette "On Reducing Computational Complexity of Codebook Search in CELP Coder Through the Use of Algebraic Codes", *ICASSP-90*, Vol 1,p177-180

[27] Redwan Salami, Claude Laflamme, Bruno Besette and Jean-Pierre Adoul." ITU-T G.729 Annex A: Reduced Complexity 8kbps CS-ACELP Codec for Digital Simutaneous Voice and Data" *IEEE Communication Magazine*, Sept,1997,p56-63

[28] Miguel Arjona Ramirez and Max Gerken "Efficient Algebraic Multipulse Search", *ITS'98 proceedings*, Vol 1p231-236,1998.

[29] N.K.Ha "A Fast Search method of algebraic codebook by reordering search sequence", *ICASSP-1999*, Vol 1 pp21-24.

[30] Fu-Kun Chen and Jar-Ferr Yang "Maximum-Take-Precedence ACELP: A Low Complexity search Method", *ICASSP-2001*, pp.693-696.

[31] J.Adoul, P.Mabilleau, M.Delprat, and S.Morisette, "Fast CELP coding based on algebraic codes," in *ICASSP-1987*, pp. 1957-1960.

[32] J.Adoul and C.Lamblin, "Comparison of some algebraic structures for CELP

coding of speech," *ICASSP-87*, pp.1953-1956

[33] C.Laflamme et al., "16 kbps wideband speech coding technique based on algebraic CELP," in *ICASSP 91*, pp. 13-16.

[34] *TMS320C54x DSP/BIOS User's Guide.* Texas Instruments, Dallas, TX

[35] *TMS320C54x Code Composer Studio Help*, Texas Instruments, Dallas, TX

[36] *TMS320C54x DSP Reference Set, Volume 1: CPU and Peripherals,* Texas Instruments, Dallas, TX, March 2001.

[37] *TMS320C5000 DSP/BIOS Application Programming Interface (API) Reference Guide,* Texas Instruments, Dallas, TX, November 2001.

[38] *Code Composer Studio Getting Started Guide*, Texas Instruments, Dallas, TX, November 2001.

[39] *TMS320C54x Code Composer Studio Tutoria*", Texas Instruments, Dallas, TX.

[40] *TMS320C54x Optimizing C/C++ Compiler User's Guide*, Texas Instruments, Dallas, TX, June 2001.

APPENDIX

In this appendix, a brief description of the programs developed for this thesis is given. The programs are developed on TI TMS320VC5416 DSK board. A CD-ROM containing these programs is included in this thesis. The list of the programs along with their descriptions are given below:

acelp.c	find algebraic codebook
basic_op.c	basic operators
bits.c	bit stream manipulation routines
cod_ld8a.c	main coder function
coder.c	main codec function. In this function, the coder and the decoder are combined.
cor_func.c	correlation functions
de_acelp.c	algebraic codebook decoder
dec_gain.c	decoding the pitch and codebook gains
dec_lag3.c	decoding of fractional pitch lag with 1/3 resolution
de_ld8a.c	main decoder routine
dspfunc.c	contain three DSP functions: *Pow2*, *Log2* and *inv_sqrt*

filter.c	three filter functions to compute the convolution, residual signal and synthesis filter
lpc.c	LPC analysis function
pitch_a.c	pitch related functions
post_pro.c	post-processing of output speech
postfilter.c	performs adaptive post-filtering on the synthesis speech
pre_proc.c	preprocessing of the input speech
pred_lt3.c	Compute the result of long-term prediction with fractional interpolation of resolution 1/3
qua_gain.c	quatization of the adaptive codebook gain and fixed codebook gain
tab_ld8a.c	contains all the tables used by the modified codec
newAlg1.cdb	The configuration file for the implementation on the C5416 DSP board
newAlg1.cmd	The command file for the implementation on the C5416 DSP board