

ECG COMPRESSION USING LPC AND ADAPTIVE
CODE BOOK

By

VIVEKANAND CHENGALVALA

Bachelor of Technology

Nagarjuna University

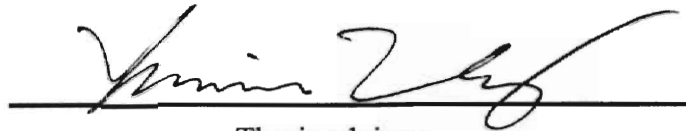
Guntur, India

2001

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the degree of
MASTER OF SCIENCE
December, 2003

ECG COMPRESSION USING LPC AND ADAPTIVE
CODE BOOK

Thesis Approved:



Thesis advisor



Dean of the Graduate College

ACKNOWLEDGEMENTS

It was indeed a great privilege to be a part of the electrical and computer engineering department of the Oklahoma State University. Five semesters spent towards working for a master's degree were filled with numerous opportunities and great experiences of both the subject and the people involved with it. This thesis completes all the formal requirements for my graduation at Oklahoma State University, and I wish to thank one and all who have walked with me towards this goal.

To start off, I thank my parents, sister and my aunt Dr. C. Sarada for extending a warm support and giving me this opportunity to come here to the United States and be able to prove myself as a capable engineer. Their love and affection always remains in my thoughts and this education would be incomplete without sincerely expressing my thanks to them.

Also I would like to thank my group members' Srinivasan, and Quinggo Fan for their interactive support ranging from subject discussion to working on projects together and having fun in the spare time.

I would like to thank the members of the Visual Communication and Image Processing Laboratory (VCIPL), Dr. Guoliang Fan, Amjad Awadeh, and my friend Sunil. K. Madgula for immense support for the completion of this project.

Living in Stillwater has definitely been a valuable experience. I would like to sincerely thank my roommates and others who made living together such a wonderful and unforgettable experience.

Lastly I would thank my advisor, Dr. Yumin Zhang for giving me a chance to work with him and financially supporting me. His wonderful and meticulous guidance on various topics have helped me learn a lot and has also made my determination stronger. I also acknowledge Professors Louis G. Johnson and Keith A. Teague for reviewing my thesis and serving as my thesis committee.

TABLE OF CONTENTS

Chapter	Page
1.Introduction.....	1
1.1 Thesis problem.....	2
1.2 Thesis outline.....	2
1.3 The Heart and the Electro Cardio Gram	2
1.3.1 Overview of the Heart and physiology	3
1.3.2 Overview of ECG	4
2.Theory of Data Compression.....	6
2.1 Data compression.....	6
2.2 ECG data compression.....	7
2.3 Previous work	7
3.ECG Compression Algorithm.....	11
3.1 Encoding	11
3.2 Comparison of ECG and speech signals.....	14
3.3 Modeling.....	14
3.4 Recursive filtering.....	17
3.5 Residue matching.....	18
3.6 Reconstruction	21
4.PRD vs Compression Tradeoff.....	24
4.1 Results.....	24
4.2 Future Work	26
5.Realization of Algorithm on TI DSP	27
5.1 Overview of DSP Algorithm Development.....	27
5.2 DSP Product Development	28
5.3 Software Development.....	29
5.4 Peripheral Setup Libraries.....	30
5.5 MATLAB Link	31
5.6 Work accomplished	33

5.7 Future Work	33
6.Summary	35
BIBLIOGRAPHY	36
APPENDIX A: MATLAB Code	38
APPENDIX B: Typical DSP Device Nomenclature	45
APPENDIX C: Which TI DSP should one use	46

LIST OF FIGURES

Chapter	Page
1. Anatomy of heart	3
2. Important feature of the ECG signal.....	5
3. A frame from ECG data.....	13
4. ECG frame containing just one peak.....	13
5. Correlator	15
6. Correlator followed by an inverse filter.....	15
7. Another representation of correlator followed by inverse filter	16
8. A block diagram of entire encoding process.....	17
9. A block diagram for residue matching.....	19
10. A snap shot of first few entries in the code book.....	20
11. The new compression scheme	22
12. The reconstruction process	23
13. (a) Original signal frame (b) Reconstructed signal frame	25
14. Error for the entire reconstructed signal	25
15. Software Gap	29
16. Developers Kit	31

LIST OF TABLES

Average pulse at different ages.....	12
--------------------------------------	----

ABBREVIATIONS

- AR:** Auto Regressive process
- AZTEC:** Amplitude Zone Time Epoch Coding
- CCS:** Code Composer Studio from Texas Instruments
- CORTES:** Coordinate Reduction Time Encoding System
- CSL:** Chip Support Library
- DMA:** Dynamic Memory Access
- DPCM:** Differential Pulse Code Modulation
- DSP:** Digital Signal Processor
- DSPLIB:** DSP Function Library from Texas Instruments
- ECG:** Electro Cardio Gram
- GUI:** Graphical User Interface
- IDE:** Integrated Development Environment
- LPC:** Linear Predictive Coding
- NSRDB:** Normal Sinus Rhythm Database
- PRD:** Percentage Root Mean Square Difference
- RTDX:** Real Time Data Exchange
- TP:** Turning Point Data Reduction Algorithm

CHAPTER 1

Introduction

The research conducted for this thesis was an effort to develop a new compression algorithm for Electro Cardio Gram (ECG) signals to aid patients suffering from heart diseases. Millions of people worldwide are affected by congestive heart diseases. Many cardiac abnormalities can be detected by ECG interpretation, including enlargement of heart muscle, electrical conduction blocks, and insufficient blood flow. The need for ECG signal compression exists in many transmitting and storing applications. For example, real time transmission of ECG signals over public telephone lines can be helpful in remote telemedicine applications. Long-term (24-48 hours) wearable monitoring tasks (like Holter), usually requires continuous 12 or 24-hours ambulatory recording. An ECG signal must consist of 3 individual leads, each recording 10 bits per sample, and 500 samples per second. Some ECG signals may require 12 leads, 11 bits per second, 1000 samples per second. When converted to a digital format, this single ECG record requires a total of 1.36 gigabytes of computer storage! [1] Approximately 10 million ECGs annually recorded for the purposes of comparison and analysis in the United States alone [2]. Efficient compression of this ECG data can drastically bring down the memory requirements, making ambulatory ECG monitoring devices commercially feasible. Also, effective storage of ECG signals is required to expand the database of ECG signals for future comparison and evaluation.

1.1 Thesis problem

Preliminary simulations have been conducted to test the already developed ECG compression algorithms. Results to date indicate that a compression of 30:1 is achieved [3] and we believe that if ECG compression is done using the standard Linear Predictive Coding combined with an adaptive codebook can outperform these compression algorithms. For this reason, current research efforts are directed in part at finding a way to put this idea to work and standardize the thresholds and codebook sizes and ECG frame lengths to be grabbed from the streaming data.

1.2 Thesis outline

The body of the thesis is divided into five chapters. The next section of this chapter provides a discussion of the heart and ECG. This chapter is intended to broaden the reader's understanding the thesis problem described above. The theory of compression algorithms and previous work done in the ECG compression field is discussed in Chapter 3. Encoding, modeling, recursive filtering and residue matching and decoding is discussed in detail in Chapter 4. A complete overview of the algorithm will be discussed here. In chapter 5, various approaches to realize the developed algorithm are discussed. In the concluding chapter, summary for the thesis is provided, followed by some suggestions for future research.

1.3 The Heart and the Electro Cardio Gram

The essence of the thesis can be clarified by a general description of the heart and of the electro cardio gram. The section is divided into two sub sections. The first sub

section deals in a general way with the anatomy and physiology of the normal heart. The second sub section provides the motivation for and a functional description of the ECG. A review of the literature in this area will lead to a more complete formulation of the thesis problem.

1.3.1 Overview of the Heart and physiology

The heart is a hollow, cone-shaped muscle located between the lungs and behind the sternum (breastbone). Two-thirds of the heart is located to the left of the midline of the body and 1/3 is to the right (see Figure 1). [4]

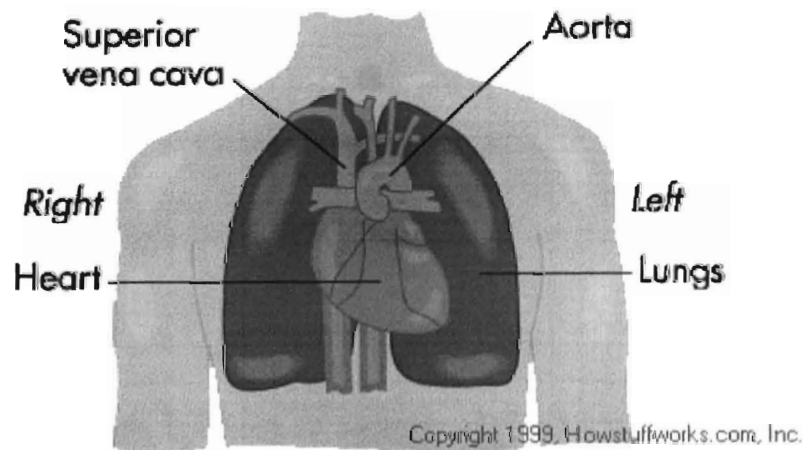


Fig. 1 Anatomy of Heart

The apex (pointed end) points down and to the left. It is 5 inches (12 cm) long, 3.5 inches (8-9 cm) wide and 2.5 inches (6 cm) from front to back, and is roughly the size of your fist. The average weight of a female human heart is 9 ounces and a male's is 10.5 ounces. The heart comprises less than 0.5% of the total body weight [4].

1.3.2 Overview of ECG

An ECG is the recording (“gram”) of the electrical activity (“electro”) generated by the cells of the heart (“cardio”) that reaches the body surface. This electrical activity initiates the heart’s muscular contraction that pumps the blood to the body. Each ECG recording electrode provides the view of this electrical activity that it “sees” from its particular position on the body surface. Observation of 12 views provided by the routine clinical ECG allows you to “move around” this electrical activity just as though you were seeing the heart from various viewpoints.

The ECG recording plots voltage on its vertical axis against time on its horizontal axis. Measurements along the horizontal axis indicate the overall heart rate and regularity, and also the time intervals required for electrical activation to move from one part of the heart to the other. Measurements along the vertical axis indicate the voltage measured on the body surface. This voltage represents the “summation” of the electrical activation of all the cardiac cells.

Many cardiac abnormalities can be detected by ECG interpretation, including enlargement of heart muscle, electrical conduction blocks, insufficient blood flow, and death of heart muscle due to a blood clot. The ECG is also the primary method to measure the disturbances in heart rate and regularity. ECG can also reveal the abnormal levels of ions in blood, such as potassium and calcium and abnormal function of glands such as thyroid [5].

A typical ECG waveform consists of the P, Q, R, S and T sections. P wave marks the activation of atria. Important features of ECG are shown in Figure 2. [6]. The activation of the left atrium, which collects oxygen-rich blood from the lungs, and the right atrium, which gathers oxygen-deficient blood from the body, takes about 90 msec. Next in the ECG cycle comes the QRS complex. The heart beat cycle is measured as the time duration of the second of the three parts of the QRS complex, the large R peak. The QRS complex represents the activation of the left ventricle, which sends oxygen-rich blood to the body, and the right ventricle, which sends oxygen-deficient blood to the lungs [2]. During the QRS complex, which lasts about 80 msec, the ventricles relax in the long T wave while the atria prepares for the next beat. These features of the ECG signal are used by a cardiologist to analyze the functioning of the heart and note arrhythmia.

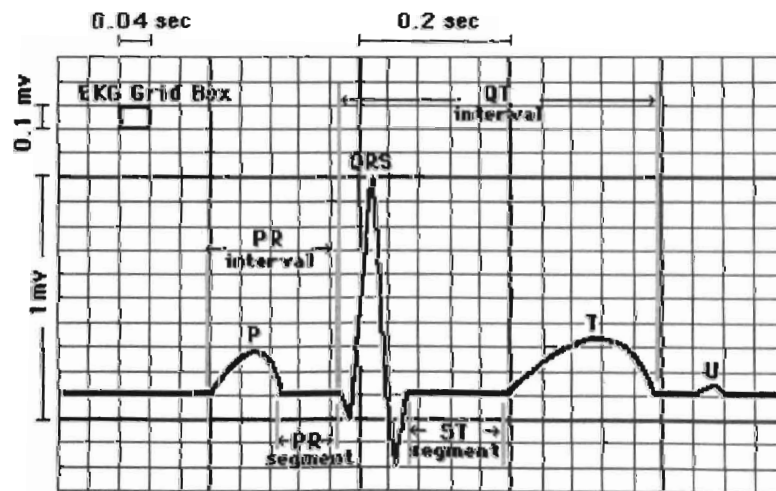


Fig. 2 Important features of the ECG signal

CHAPTER 2

Theory of Data Compression

The purpose of this chapter is to investigate a general theory of compression algorithms and is divided into three sections. The first section gives a brief overview general classification of compression algorithms. The second section describes about what type of algorithms suit the needs for the ECG data compression and the third section gives an overview of previous work done in the compression arena.

2.1 Data Compression

Data compression algorithms are broadly classified into

- a) Lossless data compression
- b) Lossy data compression

Lossless compression is used when it is important that the reconstructed and original signals should be exactly identical, e.g. software or source code compression. The compression ratios in lossless compression schemes are not very impressive. Lossy compression is used when the reconstructed signal need not necessarily be the same as the input signal but “close enough” to be useful. Most of the video, audio and image compression techniques fall into this category, where in high compression ratios can be accepted.

2.2 ECG data compression

In general, very small loss can be tolerated in biomedical signals because of the wealth of information embedded in them. The same applies for the ECG as well. As discussed in section 2.1, lossless compression techniques are not that attractive because of the poor compression ratio. So, we should look for a lossy compression technique, yet keeping the loss to a very minimal amount. Percentage RMS Difference (PRD) is one of the most popular distortion measures used for comparing ECG compression algorithms [7]. So, we are looking for algorithms which have a small value for PRD.

2.3 Previous Work

The necessity for ECG compression was identified and is an active research area for the past 30 years. Several algorithms have been developed over this time which can be classified in the following way [8].

- a) Direct data compression
- b) Transformation methods
- c) Parameter extraction techniques

Direct data compression methods base their detection on the redundancies in the data itself. Transformation methods try to capitalize on the redundancies in the transformed domain. In both these cases, original signal is reconstructed by the inverse process followed for compression. Parameter extraction techniques use an irreversible approach with which a particular characteristic or parameter of the signal is extracted. My algorithm falls into the third category.

An overview of various ECG Compression algorithms developed so far:

- a) Classical direct data compression methods
 - a. Tolerance comparison data compression techniques
 - b. Data compression by Differential Pulse Code Modulation (DPCM)
 - c. Entropy coding
- b) Direct ECG data compression schemes
 - a. The AZTEC technique
 - b. The turning point technique (TP algorithm)
 - c. The CORTES scheme
 - d. Fan and SAPA Techniques
 - e. ECG data compression by DPCM
 - f. Entropy coding
 - g. Peak picking compression
 - h. ECG cycle to cycle compression

Most of the data compression algorithms rely on utilizing prediction or interpolation algorithms. These algorithms try to reduce the redundancy in the signal by examining successive or neighboring data samples. A prediction algorithm uses *a priori* knowledge of some previous samples while interpolation algorithm uses the knowledge of both previous and future data samples. In tolerance comparison data compression, a preset error threshold is employed to eliminate data samples. It makes sense to think that if the error threshold level is high, higher data compression rate can be achieved which obviously brings down the reconstructed fidelity of the signal. DPCM attempts to

capitalize on the inter sample dependency in the data where as, entropy coding focuses on the non-uniform probability distribution of quantized signal amplitudes.

Polynomial predictors are based on finite difference techniques which constraints an n th order polynomial. Predicted data are obtained by extrapolating the polynomial one sample point at a time. Unlike the case of prediction, polynomial interpolators utilize both past and future data points. Low order polynomial interpolators are found to be very efficient for ECG data compression [8].

DPCM is also a popular method for compression. Here, the correlated input signal is represented by an uncorrelated signal which is the estimation error signal. Upon reconstruction, original signal is preserved without any loss of information. The major source of error in this scheme is due to the quantization of the estimation error signal. In general, the variance of the estimated error signal is very small compared to the variance in the input signal. Data compression by entropy coding can be achieved by assigning varying length code words to a given quantized sequence according to their frequency of occurrence. This method tries to capitalize on the unequal probability distributions of the quantized signal levels.

Some compression schemes are developed specifically for ECG compression. The Amplitude Zone Time Epoch Coding (AZTEC) originally developed for preprocessing of real-time ECGs for rhythm analysis. It is based on the tolerance comparison based methods. It became a popular compression technique for ECG monitors and databases with an achieved compression ratio of 10:1. But, in most cases, it is not acceptable to the cardiologists because of the discontinuities (step like quantization) that occur in the

reconstructed waveform. A smoothing parabolic filter can be used to remove the discontinuities but that leads to amplitude distortion to the ECG waveform.

The turning point (TP) data reduction algorithm was developed to reduce the sampling frequency of an ECG signal without diminishing the elevation of large amplitudes. The algorithm processes three data points at a time; a reference point (X_0) and two consecutive data points (X_1) and (X_2). So, it retains one of the two points X_1 and X_2 depending on which point best approximates the slope of original three points. This implies that a fixed compression ratio of 2:1 can be achieved using this TP algorithm. [8]

The Coordinate Reduction Time Encoding System (CORTES) algorithm is a hybrid of the AZTEC and TP algorithms. CORTES applies TP algorithm to the high frequency regions (QRS complex) and it applies AZTEC to the low frequency regions. Parabolic smoothing is applied to the AZTEC portions of the reconstructed data. Performance evaluation of the AZTEC, TP and CORTES algorithms were reported in [8] for ECGs sampled at 200 Hz at 12 bit resolution to be 5:1, 2:1 and 4.8:1 respectively.

Recently proposed Wavelet domain compression algorithms have a performance of up to 30:1 compression. Other neural network based algorithms are proposed which have compression ratios of about 23:1 [7]. So, the objective is to achieve a better performance than the existing algorithms.

CHAPTER 3

ECG Compression Algorithm

Our compression algorithm takes the advantage of quasi-periodic nature of the ECG signal. A lot of inter and intra beat correlation exists in the frames of ECG signal. Our algorithm takes advantage of both these correlations to achieve maximum compression ratio possible. Most of the earlier works concentrated on storing the ECG data frame, but in our algorithm, the entry to the codebook is the residue from the LPC.

This chapter is divided into six sections. The first section describes the encoding scheme followed. The second section brings out the analogy between ECG and speech signals and how the standard LPC (originally developed for speech) can be applied for the ECG signals.

Section three gives the signal processing related issues for the implementation of the algorithm. Recursive filtering and residue matching aspects are discussed in sections four and five respectively. Section six describes about the reconstruction scheme to recover the signal from the compressed data.

3.1 Encoding

The process of our encoding algorithm starts from extracting frame-by-frame data of ECG signal using a varying length window. Heart beats can vary as shown below [4]

Age	Pulse
Newborn	130
3 months	140
6 months	130
1 year	120
2 years	115
3 years	100
4 years	100
6 years	100
8 years	90
12 years	85
Adult	60 - 100

Table.1 Average pulse at different ages

To accommodate two peaks, we need to take an initial frame from the ECG signal with a frame length of at least 300. In our algorithm, we picked a frame length of 330. Figure (3) shows this ECG signal. Now, we find the peaks in this frame. Once we identify the first three peaks, we grab a window centering the middle peak. This way, each window is made sure to have only one peak. So, the length of window depends on successive peaks in the ECG. Hence, windows vary in length.

Figure (4) shows the Window grabbed from figure (3). Observe that there is only one peak per window.

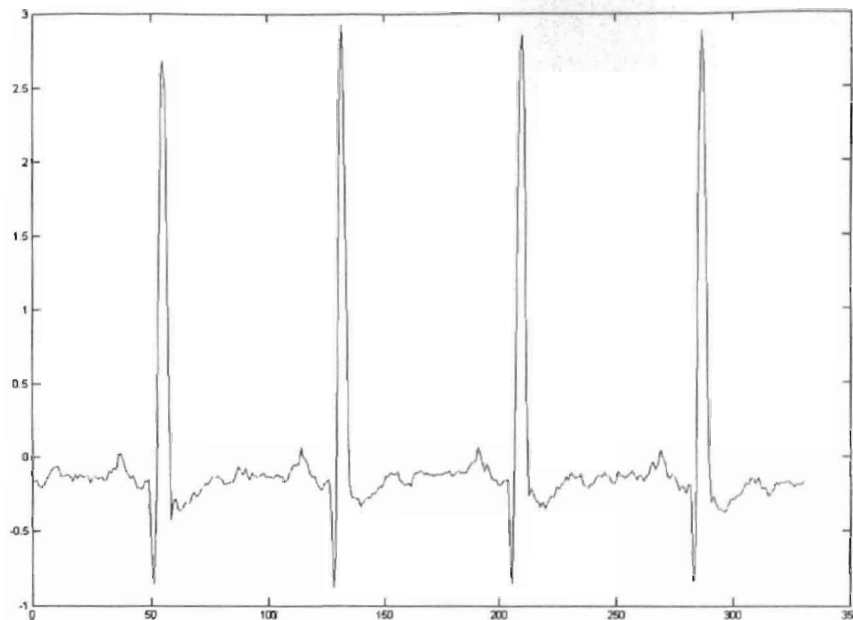


Fig. 3 A frame from the ECG data

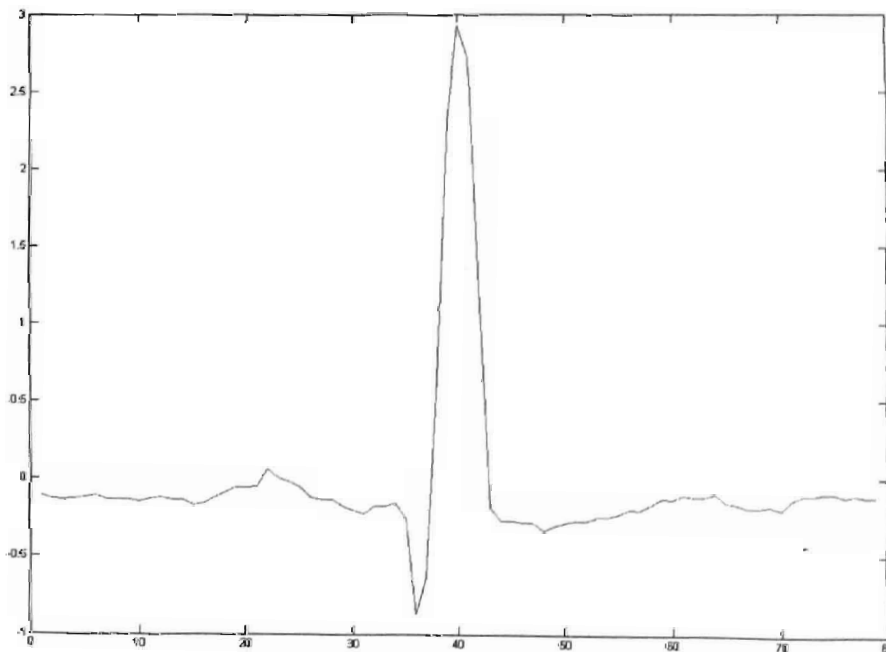


Fig. 4 ECG frame containing just one peak

3.2 Comparison of ECG and Speech Signals

In vowel production, air is forced from lungs by contraction of muscles around the lung cavity. Air then flows past the vocal cords, which are two masses of flesh, causing periodic vibration of the cords whose rate gives the pitch of the sound the resulting periodic puffs of air act as an excitation input, or source to the vocal tract. The vocal tract is the cavity between the vocal cords and the lips, and acts as a resonator that spectrally shapes the periodic input, much like cavity of a musical instrument [9].

Similarly, the heart automatically beats hour after hour, day after day because of a special group of cells that have the ability to generate electrical activity on their own. These cells separate charged particles. This produces electrical impulses in the pacemaker cells which spread over the heart, causing it to contract. Although the pacemaker cells create the electrical impulse that causes the heart to beat, other nerves can change the rate at which the pacemaker cells fire and the how strongly the heart contracts. These nerves are part of the Autonomic Nervous System [4].

3.3 Modeling

A simple source/filter model can be used to model this phenomenon. This model works only if the vocal tract is an LTI system with a periodic impulse like input. But, in general, the time varying source and system can also non-linearly interact in a complex way. So, analysis/synthesis models are embarked. AR model with inverse filtering is a popular model to model such speech.

An all pass filter basically acts as a correlator. In figure 5, S (n) is correlated.

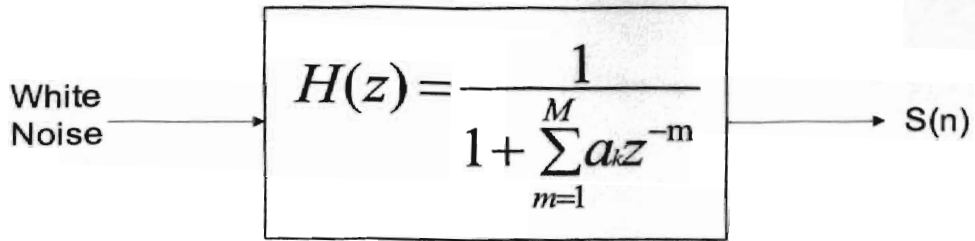


Fig. 5 Correlator

If an inverse filter follows the all pass filter, it will de-correlate the signal and the output will be white noise, as shown in figure 6.

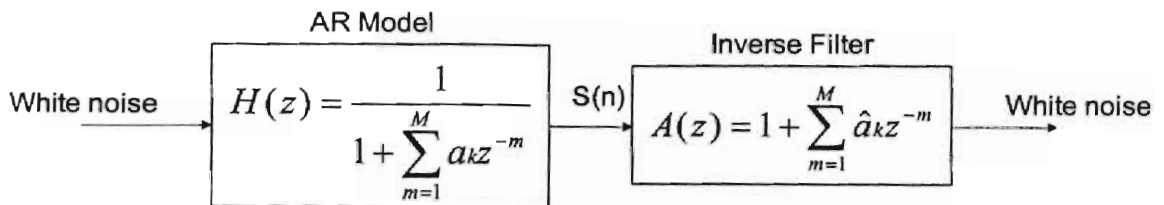


Fig. 6 Correlator followed by an inverse filter

A signal decorrelator is basically a linear predictor [10]. A linear predictor decorrelates the signal by minimizing the difference between the current sample and a linear combination of past samples. The error $e(n)$ is given by the following equation.

$$e(n) = S(n) - \sum_{m=1}^M a_m S(n - m)$$

This can be represented as shown in figure (7).

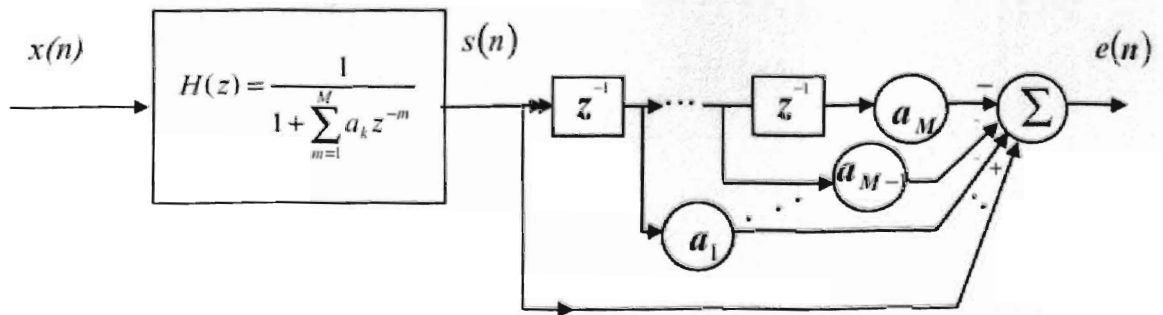


Fig. 7 Another representation of correlator followed by an inverse filter

We don't have access to the AR process and we don't know its coefficients. So, $e(n)$ is minimized in the least squares sense. It can be shown that the predictor will identify the coefficients of the AR process [9]. If the underlying signal didn't come from an AR process, $S(n)$ will be at best approximated by the coefficients obtained by the linear predictor. Levinson-Durbin algorithm can be used to minimize the error as it is very efficient. MATLAB uses Levinson-Durbin algorithm while calculation the Linear Prediction Coefficients. [11]

In performing analysis over each window, we estimate that the transfer function parameters (poles and zeros), as well as the parameters that characterize the input of our discrete time model. The short time stationary condition requires that the parameters of the underlying system are nearly fixed under the analysis window and therefore that their estimation is meaningful.

Once we pass the frame through the recursive filter, we get the residue and the LPC coefficients. The frame can be reconstructed perfectly by inverse filtering the residue using the LPC coefficients. But, compression is achieved by effectively managing the residue. For speech signal processing, we throw away the original residue, and

generate white noise with the same power spectral density of the residue and pass that through the inverse filter. So, the reconstructed signal sounds intelligible. But, for biomedical signals, we don't want intelligible signals. We want almost perfect reconstruction, because of the wealth of information in the signals. So, for this, we cannot throw away the residue, as we will require that for reconstruction. But, if we transmit the residue along with the LPC coefficients, we are actually increasing the signal size because, the residue is of the same length of the original frame and LPC coefficients add as overhead. So, an innovative way needs to be found out to effectively transmit the residue. For this, the idea of codebook is introduced. A block diagram representation of the entire encoding process is shown in figure (8).

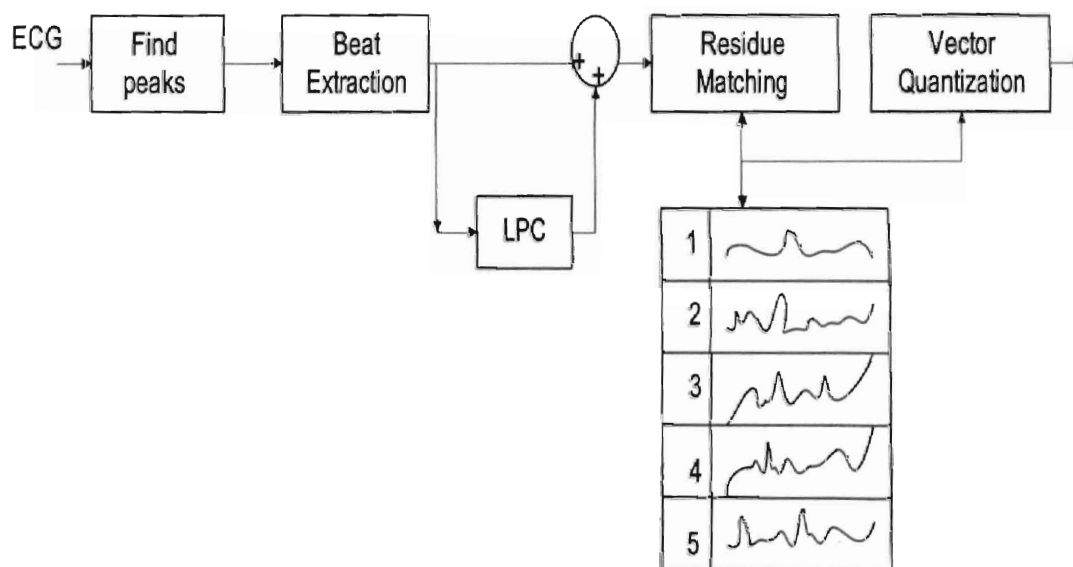


Fig. 8 A block diagram of entire encoding process

3.4 Recursive Filtering

Each frame of data is recursively filtered to extract the LPC coefficients and the residue is stored in the codebook as shown in the figure. So, while transmitting, the LPC

coefficients and the index in the code book are transmitted. At the receiving end, the same codebook is present. So, the respective residue is extracted and inverse filtered to get the original frame. The idea of compression creeps in when we don't store all the residues. If a residue with nearest match is identified, that index number is transmitted. Else, the residue is appended to the codebook and the new index is transmitted along with the new entry. The codebook at the receiver end is also updated with this new entry, thus making sure that the codebooks at both the receiver and transmitter sides are always the same. So, we start with a blank code book and keep on filling with residues. At a certain stage, this codebook is assumed to be saturated with residues and an append occurs very rarely. So, ideally, we end up transmitting only the LPC coefficients and codebook index number per every frame and the compression ratio is thus very high.

3.5 Residue Matching

Once we obtain the new residue, it is to be compared with all the previous entries in the codebook and if a close enough match is spotted, it needs to return that index number or else, it needs to append the new residue. So, for comparison purposes, we inverse filter all the codebook entries and reconstruct the signal with those residues and compare with the original frame of data using the PRD error criterion. So, the compression ratio achieved is dependent on this error threshold. If the reconstruction error is very loose, a match is easily identified. On contrary, if the error threshold is demanding, we end up with a large codebook as it is difficult to find an acceptable residue from the codebook. A block diagram for residue matching is shown in figure (9).

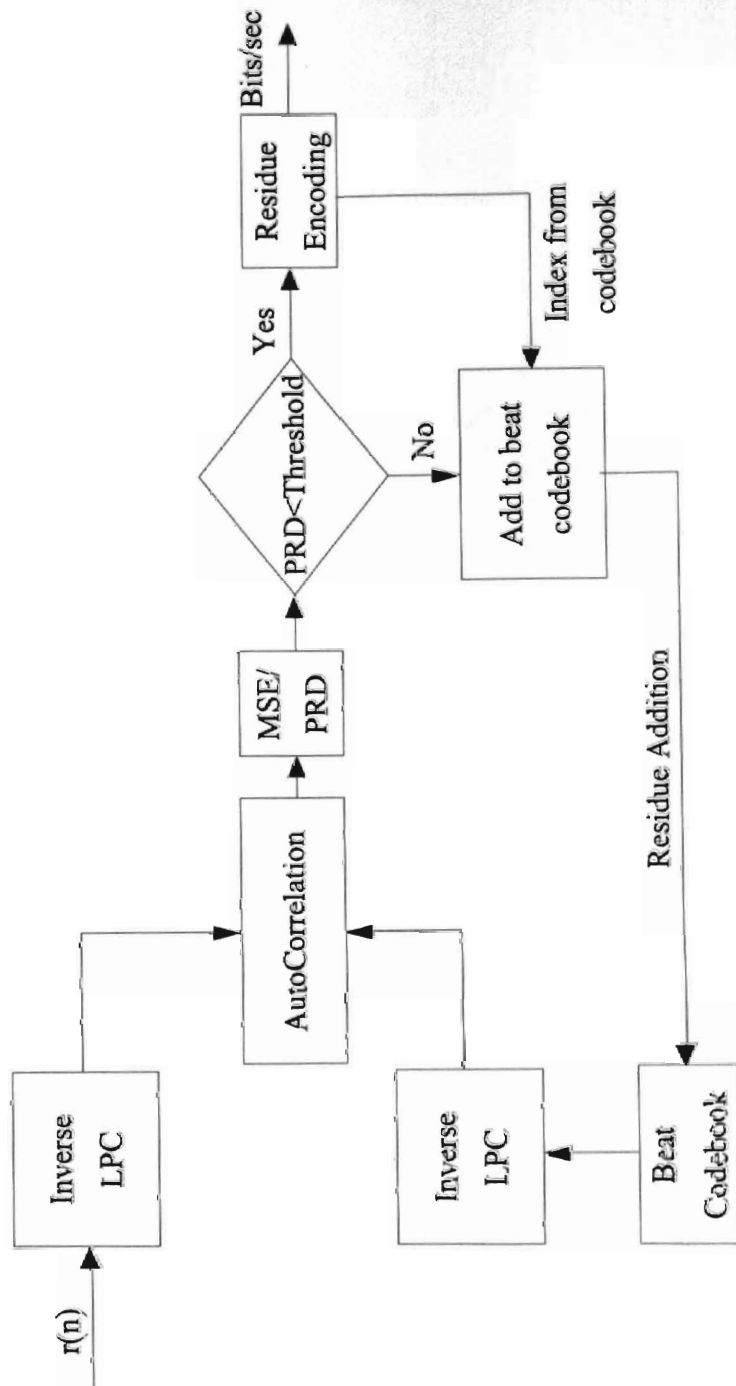


Fig. 9 A block diagram for residue matching

Now, as we have a varying length of frame, we have varying length of residue (residue is of the same length as that of the filtered frame). So, it is hard to find a

matching residue in the codebook. If we can find out a way to overcome this difficulty, the idea is good and under ideal case, we end up with good compression ratios.

On observing the codebook and stepping through the matching algorithm, it's been observed that even when two residues are closely matched, the algorithm doesn't identify that because of the difference in the length. A snap shot of a few entries in the codebook is shown in figure (10).

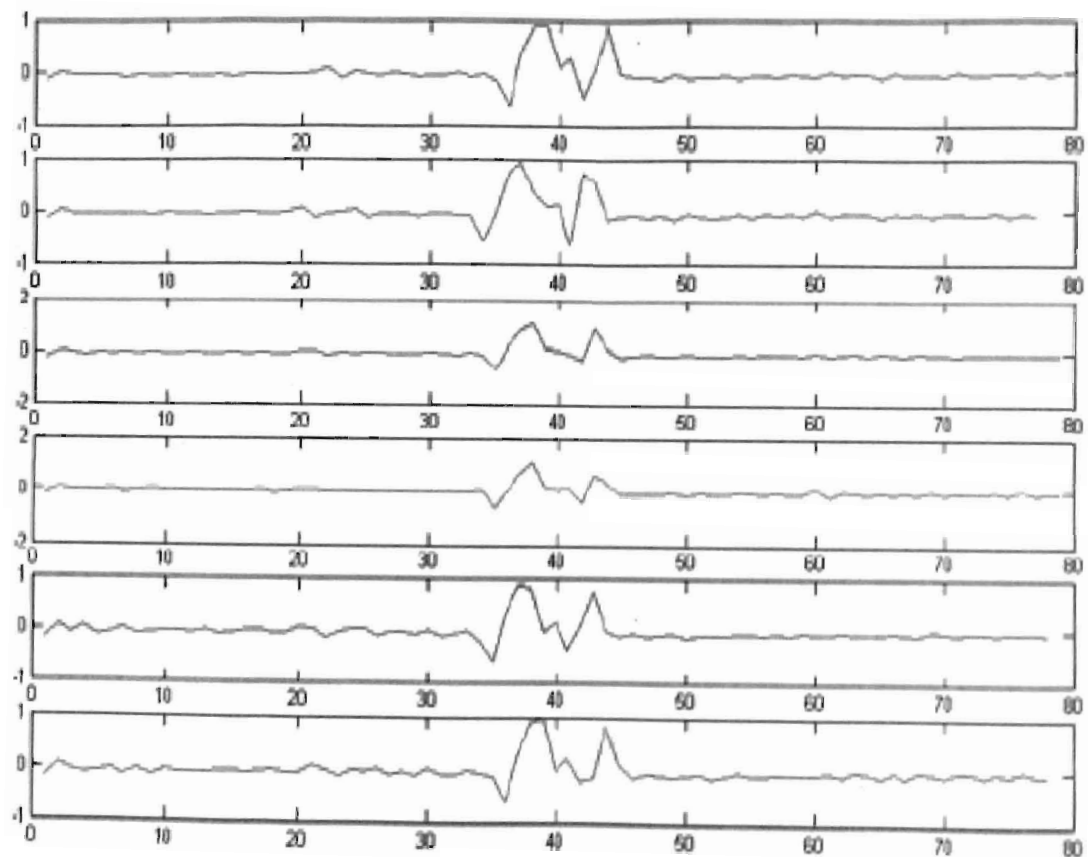


Fig. 10 A snap shot of first few entries in the codebook

On observation, it can be said that the residues look pretty similar but are of different lengths. This led us to the idea of partitioning the codebook. Various partitioning options are tried and partitioning into three with (0:30, 31:60 and 61:end) is

found to be ideal. This sounds reasonable on observing the codebook in figure (10). Now, we end up with a partitioned codebook, and LPC coefficients and three indices in the codebook needs to be transmitted. This is slightly an overhead but, the size of the codebook reduced by 90%. The first and the second partitions in the codebook saturated with a very small number of entries and the third partition had considerably large entries. This is expected because of the varying length in the third segment. The new scheme is shown in figure (11).

3.6 Reconstruction

After the codebook is obtained from the encoding process, the parameters that need to be transmitted per frame are the LPC coefficients and the three indices in the codebook. The index numbers in the bit form are decoded and looked up in the codebook and the frame is extracted. Now, this frame is inverse filtered using the respective LPC coefficients to obtain the ECG data frame. This is repeated for all the frames and the reconstructed frames are concatenated to obtain the ECG signal. A block diagram of the reconstruction process is shown in figure (12)

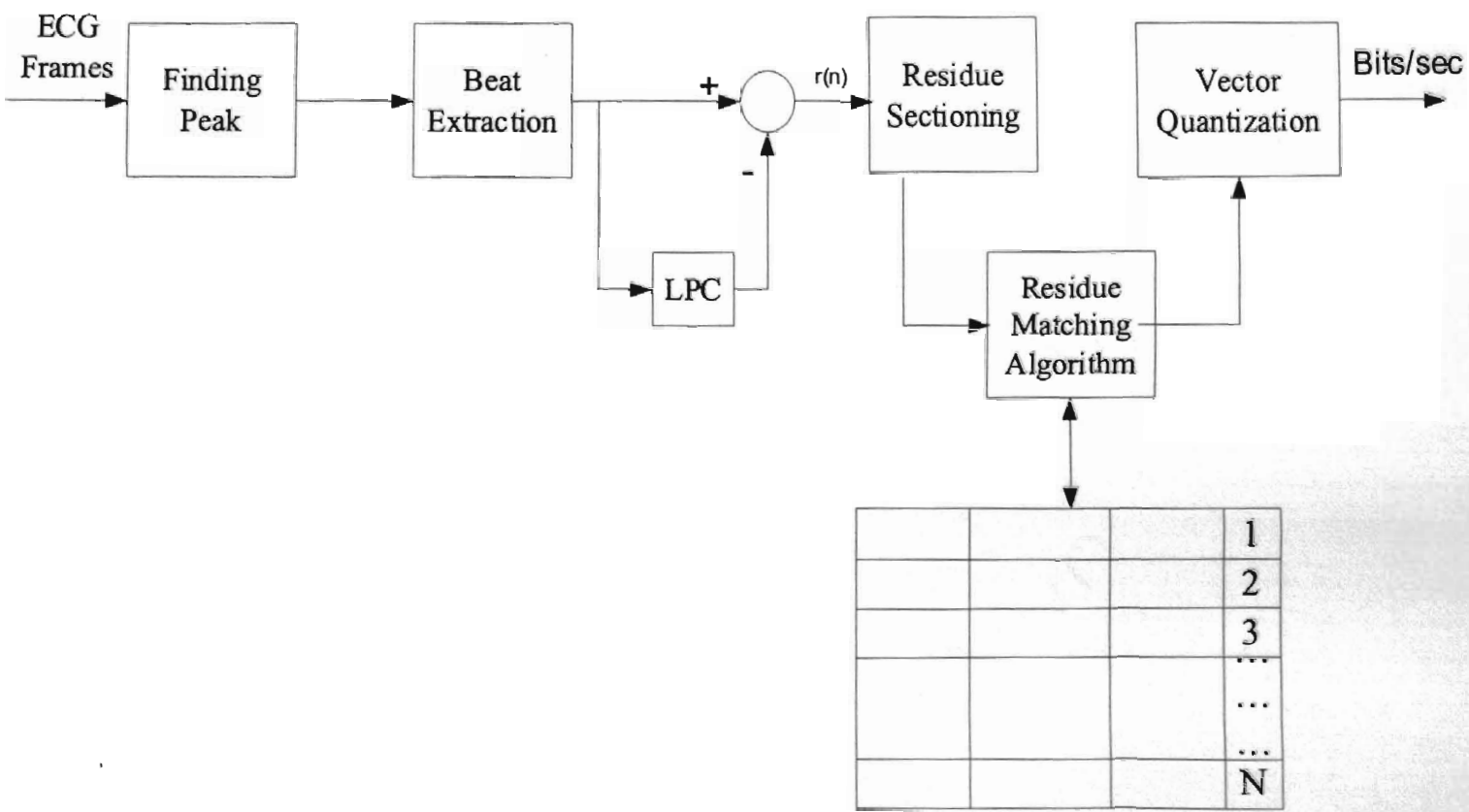


Fig. 11 The new compression scheme

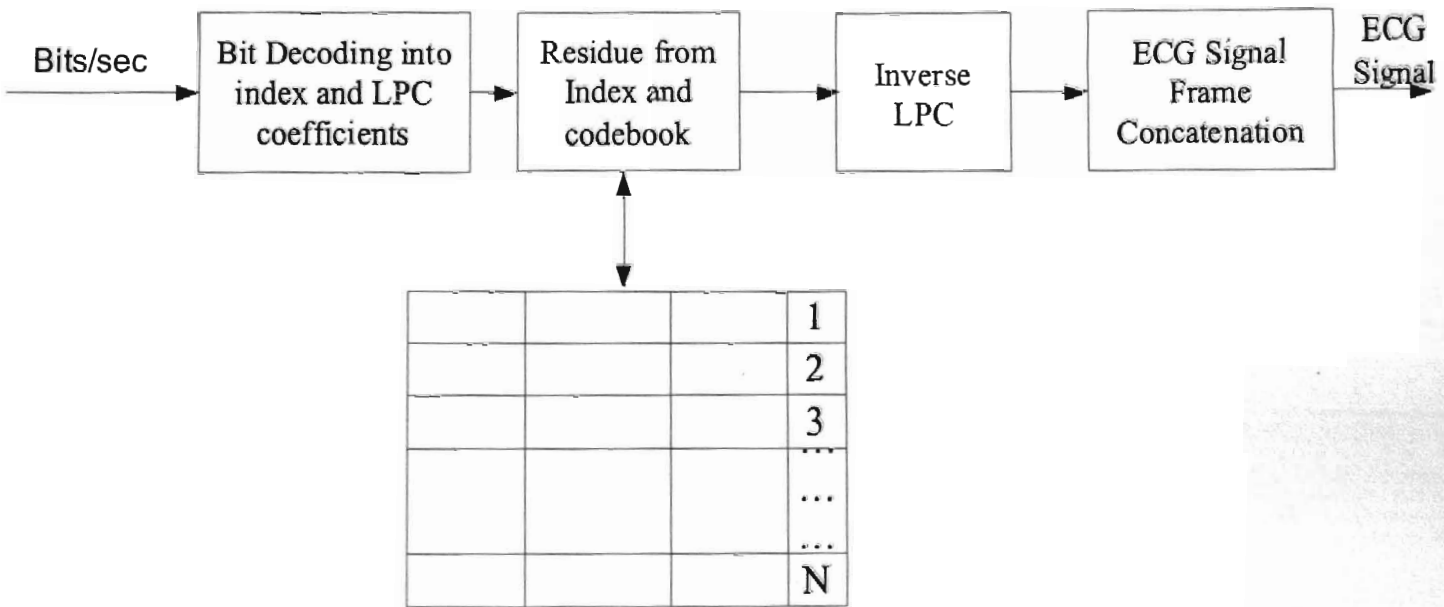


Fig. 12 The reconstruction process

CHAPTER 4

PRD vs. Compression Tradeoff

The work so far presents a new algorithm for ECG signal compression. By partitioning the codebook, less residue frames needed to be saved, and hence, a higher compression ratio is achieved. Acceptable PRD can be specified before hand, which is used as threshold when updating the codebook. A requirement of lower PRD results in a larger codebook and vice-versa. For a preset mean PRD of less than 8%, the codebook tends to saturate for 128 entries for any ECG signal in the MIT-BIH normal sinus rhythm database (NSRDB) database. This chapter is divided into two sections. The first section gives the compression results achieved on the NSRDB and the second section gives the future work that can be done in this area.

4.1 Results

The efficiency of the proposed algorithm is evaluated using MIT-BIH normal sinus rhythm database (NSRDB). The first minute of the three signals with record numbers 16265, 16273, and 16420 are processed for checking and correctness of the algorithm. A PRD of 6.875%, 5.99%, and 3.6785% is correspondingly recorded, while a mean compression ratio of 40:1 is recorded.

Figure (13) shows the original and reconstructed signal record 16265 picked up randomly at a certain point of time and Figure (14) shows the error for the entire reconstructed signal.

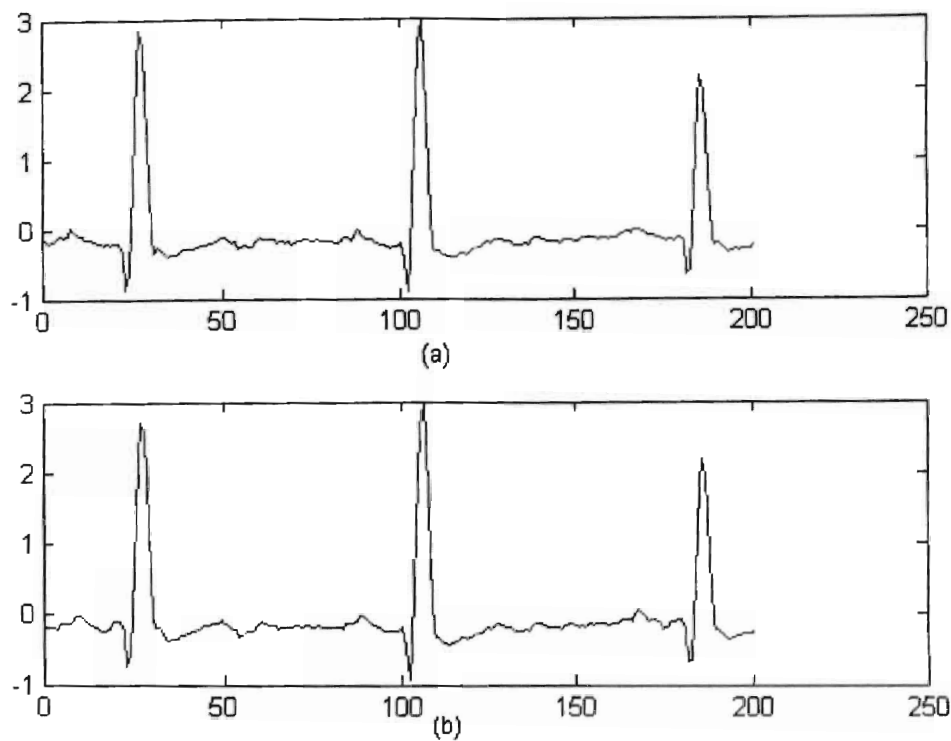


Fig. 13 (a) Original Signal frame (b) Reconstructed signal frame

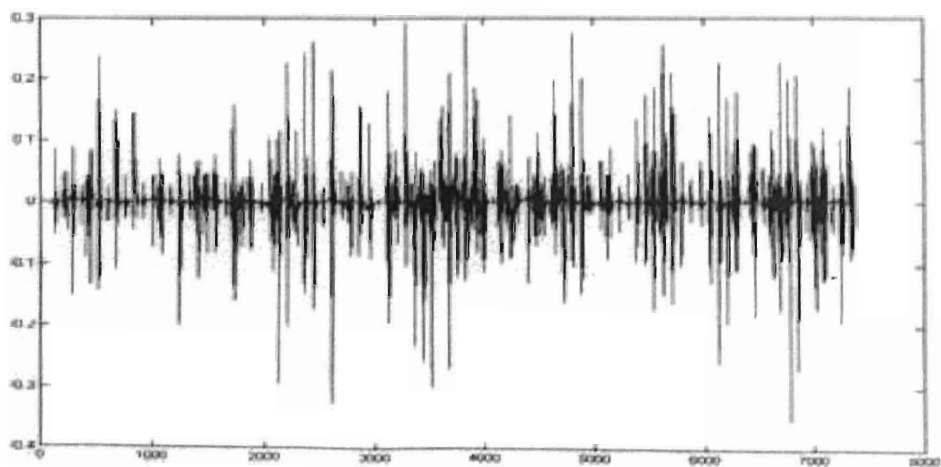


Fig. 14 Error for the entire reconstructed signal

4.2 Future Work

This concept of codebook can be extended to the idea of Universal Codebook, which contains the residue signals for all types of ECG signals with arrhythmia, congestive heart diseases. Therefore, any ECG signal can be compressed to a very low bit rates without losing fidelity and there is no necessity to update or transmit the code vectors because of this standard codebook.

CHAPTER 5

Realization of algorithm on a TI DSP

The previous chapter proposes a new algorithm for ECG compression. This chapter primarily focuses on the realization issues for the algorithm on a Texas Instruments Digital Signal Processor (DSP). There are several DSPs available in the market and the criteria for picking a suitable DSP is given in the appendix C. This chapter is organized into three sections. The first section gives an overview of the DSP algorithm development cycle. The second section brings out the software gap that exists in the DSP product development. Software development, set up of peripheral libraries, and realization of the algorithm with MATLAB is discussed in the next sections. The concluding sections provide the work accomplished in the realization and mentions some future work that can be done.

5.1 Overview of DSP Algorithm Development

We can create an application in a MATLAB language called the M-Language. In addition to MATLAB, Mathworks provides a collection of m-files or functions called the tool boxes. One of the famous toolboxes is the Signal processing toolbox. The functions that we create can contain function calls to any of the toolboxes that we have. Apart from the toolboxes, MATLAB also has linker, debugger, and GUI development environment called GUIDE. In contrast to MATLAB which is function oriented, SIMULINK is a block diagram oriented tool, where we can create a model by adding blocks from the

SIMULINK libraries. Some of the standard blocks available from MATLAB are the DSP block set, Communications block set. We can visualize the results of the simulation by adding probes, graphs etc in the block diagram. [12]

5.2 DSP Product Development

In general, there are two groups of developers involved in DSP product development.

- a) Algorithm Developers
- b) Software Developers

Algorithm developers are staff level engineers who design specifications for the products. They perform algorithm concept and design in MATLAB or other high level languages. They have no access to DSP hardware.

On the other hand, software developers are not using the system level design tools like MATLAB and SIMULINK. Instead, they are creating DSP implementations directly in C or Assembly. This is problematic because, developing in C/C++ can be prone of manual recoding errors. Also, the specifications that the software developers receive from the algorithm developers might not be entirely clear.

The algorithm developers and software developers do have some problems in common and that is their level of job overlap is very badly inclined. So, people need to know both algorithm development and software development, which is slightly unrealistic requirement. There is no common platform that both groups can share. This leads to the software gap, shown in Figure (15).

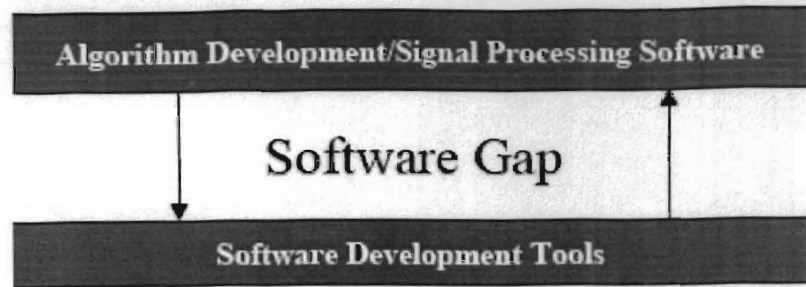


Fig. 15 Software Gap

When developers employ traditional development methods, the following things may happen

- 1) Design flaws detected too late
- 2) Can't compare implementation to design specifications
- 3) Can't analyze real time DSP data to diagnose problems
- 4) Algorithm developers can't test ideas without DSP programming
- 5) Can't transfer data from DSP to MATLAB for analysis

Results:

- 1) Slow design cycle times
- 2) Higher development costs
- 3) High risk of design failure
- 4) Missed opportunity for product innovation

5.3 Software Development

DSP/BIOS is a scalable real-time kernel. It is designed for applications that require real-time scheduling and synchronization, host-to-target communication, or real-time instrumentation. DSP/BIOS provide preemptive multi-threading, hardware abstraction, real-time analysis, and configuration tools. The main feature of DSP/BIOS is

to provide a Real-time preemptive scheduler giving real time analysis and at the same time, have hardware abstraction.

The benefits of using DSP/BIOS are it is fully optimized for TMS 320 DSP and supported on C 6000 and uses minimal MIPS and memory and more importantly integrated with the code composer studio. [13]

In traditional method for system setup, user must create code for:

- a) Interrupt vector table
- b) Global settings: cache, interrupt table mapping, interrupt enable
- c) Peripheral setup: timers, serial ports, DMA
- d) Linker command files with system memory map

So, it has lots of details to learn, tedious to write the code, difficult to debug. Code composer studio provides a graphical tool to write this code.

5.4 Peripheral Setup Libraries

There are several libraries provided by the Texas Instruments and we require two libraries for evaluating the ECG compression algorithm. They are the chip support library and the DSP function library.

Chip Support Library (CSL) is a function library to control on-chip peripherals like CACHE, DMA, EDMA, EMIF, HPI, IRQ, MCBSP, PWR, TIMER etc. DSP Function Library (DSPLIB) is an optimized, C or Assembly-callable and has functions for adaptive filtering, Correlation, FFT, Filtering and Convolution, Math, Matrix etc. There was problem setting up the Chip Support Library and BIOS. Also, there was no

access to the ECG measuring equipment to interface directly with the DSK. So, BIOS (real time analysis) was not paid much attention. The signal was successfully imported to the DSK through the Universal Serial Bus (USB) port. To implement the algorithm, LPC algorithm has to be implemented and it was not supported by the DSPLIB. There was requirement for other functions which are not supported by the DSPLIB of code composer studio. To overcome the writing everything again from scratch in C/Assembly language, it is a good idea to have a common tool for both algorithm development and software development.

5.5 MATLAB Link

MATLAB provides this tool in its latest version. One can directly do the algorithm implementation in MATLAB and optimize the code at the high language level and then, link with the Code Composer Studio (CCS) of Texas Instruments using Real Time Data Exchange (RTDX). Also, one can do the software development starting with the SIMULINK model. [13] This is shown in figure (16).

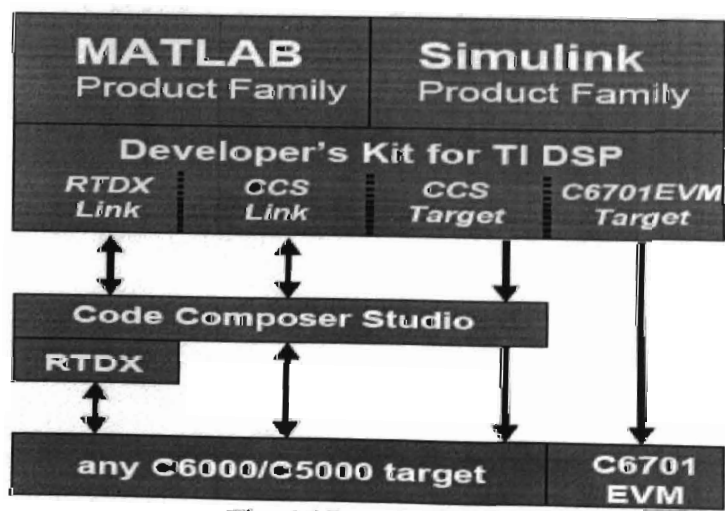


Fig. 16 Developers Kit

MATLAB Link for Code Composer Studio provides four components that work with and use CCS IDE and TI Real-Time Data Exchange (RTDX)

Link for Code Composer Studio IDE: Permits to use objects to create links between CCS IDE and MATLAB. From the command window, applications in CCS IDE can be run, data from target memory can be sent to and received, and processor status can be checked, as well as other functions such as starting and stopping applications running on the digital signal processors.

Link for Real-Time Data Exchange Interface: Provides a communications pathway between MATLAB and digital signal processors installed on the PC. Using objects in the MATLAB Link for Code Composer Studio, channels to processors on boards can be opened and data can be sent and received about the processors and executing applications.

Embedded Objects: Provides object methods and properties that allow to access and manipulate information stored in memory and registers on digital signal processors, or in the Code Composer Studio project. From MATLAB information from the project can be gathered, and can be worked with the information in MATLAB, doing things like converting data types, creating function declarations, or changing values, and return the information to your project--all from the MATLAB command line. [14]

Hardware-in-the-Loop: Enables to write scripts in MATLAB that exercise functions from the project on the target processor. From MATLAB, data can be generated, and can be sent to the target and a C function can be used in the program to manipulate the data on

the hardware or simulator. Afterwards, the output can be returned to MATLAB for analysis of the results. [14]

There was a problem in opening the channels for real time data exchange (RTDX) from MATLAB, and so, data could not be successfully sent to and received from the DSK.

5.6 Work accomplished

DSK accepts The ECG signal is successfully modified into a DSK compatible signal by including proper header information and get the signal frames to the board and amplify the frames and calculate the FFT of the frames get them back from the board and displayed on the computer. As the DSP/BIOS could not be configured properly, probes were used to exchange data between the host PC and the buffer on the target. The algorithms were developed in MATLAB and C/C++ implementations of those were required to realize and the effect of software gap was faced.

From MATLAB, the RTDX channels could not be opened properly. The signal was sent to the DSK but there was no way of getting the signal back. The signal could be probed via watch variables in the execution of the CCS project in debug mode.

5.7 Future work

The algorithms developed in MATLAB can be converted into C/C++ implementations so that they can be realized on CCS. The RTDX channels in MATLAB can be configured properly, so that the MATLAB algorithms can directly be realized on

DSK without any modifications. There are several release versions of MATLAB and CCS and they are not quite compatible. There are several patches available from Texas Instruments but, they are not helpful either. Hopefully, these issues will be resolved in their next release.

CHAPTER 6

Summary

A new algorithm for ECG compression has been developed which out performed other contemporary algorithms on the NSRDB database from MIT. A mean compression ratio of 40:1 is achieved.

There are primarily three future research directions for this work.

- a) This work can be extended to other databases in the MIT database
- b) Other algorithms developed for speech can also be tested for their compression ratios.
- c) The developed algorithm can be realized on a TMS 320 processor for marketing the product.

BIBLIOGRAPHY

- [1] <http://www.americanheart.org>
- [2] <http://hcs.harvard.edu/~weber/HomePage/Papers/ECGCompression/>
- [3] Yaniv Zigel, Arnon Cohen and Amos Katz, "ECG signal compression using analysis by synthesis coding," *IEEE Trans. Biomed. Eng.*, vol. 47, pp. 1308–1316, Oct. 2000.
- [4] <http://science.howstuffworks.com/heart.htm>
- [5] Galen S. Wagner *Practical Electrocardiography*, 10th ed, Philadelphia: Lippincott Williams & Wilkins 2001.
- [6] http://cal.nbc.upenn.edu/lgcardiac/ecg_tutorial/hearttrate.htm
- [7] Vivekanand Chengalvala, Amjad Awadeh, Sunil K.Madgula and Yumin Zhang, "An Experimental Study for Individualized ECG compression using LPC and Adaptive Code Book for Long Time Recording", International Signal Processing Conference(ISPC), Dallas, TX. 2003.
- [8] Sateh M.S. Jalaeddine, Chriswell G. Hutches, Robert D. Strattan and William A Cobberly, "ECG Data Compression Techniques--A Unified Approach," *IEEE Transactions on Biomed. Eng.*, Vol. 37, No. 4, pp. 329-343. 1990.
- [9] Thomas F. Quatieri, *Discrete-Time Speech Signal Processing Principles and Practice*, Prentice Hall Inc., NJ 2002
- [10] http://hitchcock.dlt.asu.edu/media3/a_spanias/eee506-s02/PDF-506/EEE506-LECT11.pdf
- [11] The Math Works - Signal Processing Toolbox

[12] <http://www.mathworks.com>

[13] <http://dspvillage.ti.com>

[14] <http://www.mathworks.com/access/helpdesk/help/toolbox/ccslink/ccslink.shtml>

APPENDIX A

MATLAB Code

```
% this program uses a varying window length
clear all;
fp=fopen('frames.m','w');
global data
global codebook
load -ASCII ecg4.csv;

data=ecg4(:,2);

% data=data(1:3500);
% data=cat(1,data,data);
codebook={};
residue=[];
bitUsed=6;
% res={};
codebook.index(1:100)=[1:100];

i=1;
% this loop will search for a 3 consecutive peaks, and then take a window in the middle
of them
start=0; %index for starting of a new window.
loopcount=0;comp=[0 0 0];incr=0;ind=[0 0 0];incr1=0;incr2=0;incr3=0;ent=[];
while start<(length(data)-330)% for i=1:4 %length(data)

    loopcount=loopcount+1;
    x=data(start+1:start+330);
    cc=findPeak(x);

    if start==0,

        left=floor((cc(1)+cc(2))/2)+start ;
        right=floor((cc(2)+cc(3))/2)+start;
    else
        right=floor((cc(1)+cc(2))/2)+start;
    end

%   right=floor((cc(1)+cc(2))/2)+start;
```

```

right-left;
if loopcount==1
    fprintf(fp,'%d %f ',left-1,data(1:left-1));
end
start=right+1;

frame=data(left:right);

left=start;
fhamwin=frame;%.*w;
% y=lpc(fhamwin,8);
y=lpc(fhamwin,2);
y=real(y);
temp=filter(y,1,fhamwin);
% res.row1 {loopcount}=temp(1:end);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Time to see if the current residue need to be stored in the code book, or
% an entry from the code book can produce an acceptable error
if loopcount ~ =1,
    [comp,ind] = chkerrVer5(temp,y); % the value of comp determines whether the
residue value should be
end % entered into the codebook or not, comp=1 the residue with a
best match exists,

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if comp(1)==0| comp(1)==2, %comp=0 no residue of similar length found,
comp=2 mse is not acceptable. So comp=0,2 needs update
    incr1=incr1+1;
    codebook.row1 {incr1}=temp(1:30); % incr is an index in the codebook
    ent(1)=codebook.index(incr1);
else
    ent(1)=ind(1); % ent value is to be written in file.
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if comp(2)==0| comp(2)==2, %comp=0 no residue of similar length found,
comp=2 mse is not acceptable. So comp=0,2 needs update
    incr2=incr2+1;
    codebook.row2 {incr2}=temp(31:60);

```

```

        ent(2)= codebook.index(incr2);% incr is an index in the codebook
    % codebook.length(incr)=length(temp);
    %codebook.index(incr)=incr;
    %end

%   if (ind(2)==0 | ind(2)==2)
%       ent(2)= codebook.index(incr2);
else
    ent(2)=ind(2); % ent value is to be written in file.
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if comp(3)==0| comp(3)==2,          %comp=0 no residue of similar length found,
%comp=2 mse is not acceptable. So comp=0,2 needs update
        incr3=incr3+1;
        codebook.row3 {incr3}=temp(61:end);          % incr is an index in the codebook
        codebook.length(incr3)=length(temp(61:end));
        ent(3)= codebook.index(incr3);
    else
        ent(3)=ind(3); % ent value is to be written in file.
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

yQuant=(round(y*2^bitUsed))/(2^bitUsed);
    fprintf(fp,'\n %2.5f %2.5f %2.5f  %d %d %d ', yQuant,ent);
end
fclose(fp);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% FindPeak.m %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% this function will find the peaks in s sequence
% inputs are the sequence, and the Tolerance
function [zz]=findPeak(x)

peakTh=0.5*max(x);
i=1;
zz=[]; % initialize this array for storing the peak indices
index=1;
yy= find(x>peakTh); % find the indices exceeding the threshold
lenofy=length(yy);

```



```

yy(lenofy+1)=0; % pad with zeros for avoiding "index exceeds matrix dim" error at the
end.
yy(lenofy+2)=0;
j=1;
while j<lenofy
    count=1;
    temp=[];
    temp(count)=yy(j);
    count=count+1;
    for q=1:5 % check for a maximum of five point vicinity near a peak.
        if (((yy(j+1)-yy(j)) ==1))
            temp(count)=yy(j+1);
            j=j+1;
            count=count+1;
        end
    end
    j=j+1;
    zz(index)=floor(mean(temp)); % compute the avg index of the 5 point vicinity near the
    peak and say that average as index of peak.
    index=index+1;
end
return

```

%%%%%%%%%% ChkerrVer5.m %%%%%%%%%%

% this function recieves residue frames and lpc coeffs and checks for best match in
%codebook.

```

function [var,index] = chkerrVer5(resi,lp1)
global codebook
var=[];
index=[];
len= length(resi(61:end));
orgsig= resi;thld1=20; thld2=10; thld3=15;
chk=find(codebook.length==len);
look=1000;

```

%%%%%%%%%%
%%%%%%%%%%

```

reconorg=filter(1,lp1,orgsig(1:30));
for k= 1:length(codebook.row1)
    sig=codebook.row1 {k};
    reconsig=filter(1,lp1,sig);
    % err is an array, storing the mse of the original and reconstructed signals from
    matched residues

```

```

    err1(k)=prd(reconorg,reconsig);
end
[look,in1]=min(err1);
if (look>thld1),
    var(1)=2;
    index(1)=2;
else
    index(1)=codebook.index(in1);
    var(1)=1;
end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
reconorg=filter(1,lp1,orgsig(31:60));

```

```

for k= 1:length(codebook.row2)
    sig=codebook.row2{k};
    reconsig=filter(1,lp1,sig);
    % err is an array, storing the mse of the original and reconstructed signals from
    matched residues
    err2(k)=prd(reconorg,reconsig);
end
[look,in2]=min(err2);
if (look>thld2),
    var(2)=2;
    index(2)=2;
else
    index(2) = codebook.index(in2);
    var(2)=1;
end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

reconorg=filter(1,lp1,orgsig(61:end));
if isempty(chk)
    var(3)=0;
    index(3)=0;
else
    for k= 1:length(chk)
        sig=codebook.row3{chk(k)};
        reconsig=filter(1,lp1,sig);
        % err is an array, storing the mse of the original and reconstructed signals from
        matched residues
        err3(k)=prd(reconorg,reconsig);
    end

```

```

    err1(k)=prd(reconorg,reconsig);
end
[look,in1]=min(err1);
if (look>thld1),
    var(1)=2;
    index(1)=2;
else
    index(1)=codebook.index(in1);
    var(1)=1;
end

```

```

%%%%%%%%%%
%%%%%%%%%%
reconorg=filter(1,lp1,orgsig(31:60));

```

```

for k= 1:length(codebook.row2)
    sig=codebook.row2 {k};
    reconsig=filter(1,lp1,sig);
    % err is an array, storing the mse of the original and reconstructed signals from
    matched residues
    err2(k)=prd(reconorg,reconsig);
end
[look,in2]=min(err2);
if (look>thld2),
    var(2)=2;
    index(2)=2;
else
    index(2) = codebook.index(in2);
    var(2)=1;
end

```

```

%%%%%%%%%%
%%%%%%%%%%

```

```

reconorg=filter(1,lp1,orgsig(61:end));
if isempty(chk)
    var(3)=0;
    index(3)=0;
else
    for k= 1:length(chk)
        sig=codebook.row3 {chk(k)};
        reconsig=filter(1,lp1,sig);
        % err is an array, storing the mse of the original and reconstructed signals from
        matched residues
        err3(k)=prd(reconorg,reconsig);
    end

```

```

look=find(err3<thld3);
if isempty(look),
    var(3)=2;
    index(3)=2;
else
    index(3)=codebook.index(chk(k));
    var(3)=1;
end
end
return

```

%%%%%%%%%% Reconstruction.m %%%%%%%%%%%

```

close all
fp1=fopen('frames.m','r');
ecg=[];
status=0;
% for LPC of order 8 use y=10
% y=[10];
y=[4];
% for LPC of order 8, use zi=zeros(8,1)
zi=zeros(2,1);
% y(1)=1;
init=fscanf(fp1,'%d',1);
[ecg,init1]=fscanf(fp1,'%f',init);
% for i=1:1:90,
while status==0,
    % for LPC of order 8 use 9
    [y,count]=fscanf(fp1,'%f',3);
    [ent,cnt]=fscanf(fp1,'%d',3);
    if ent(1)~=0
        vivek= cat(1,codebook.row1 {ent(1)},codebook.row2 {ent(2)},codebook.row3 {ent(3)});
        [reconsig, zf]=filter(1,y,vivek,zi);
        %zi=zf;
        % reconsig=filter(1,y,codebook.row {ent});
        ecg=cat(1,ecg,reconsig);
    end
    status=fseek(fp1,20,0);
    % a=feof(fp1)
    fseek(fp1,-20,0);
end
figure
subplot(2,1,1)

```

```
plot(data(1700:1900));  
subplot(2,1,2)  
plot(ecg(1700:1900));
```

```
MSE = mse(ecg-data(1:length(ecg)))  
PRD= prd(ecg,data(1:length(ecg)))  
figure  
plot(data(1:length(ecg))-ecg);
```

APPENDIX B

Typical DSP Device Nomenclature

	TMS	320	C	6412	GDK	()	600	
Prefix								Device Speed Range
TMX = Experimental device								C6000™ DSPs
TMP = Prototype device								150 MHz 500 MHz
TMS = Qualified device								167 MHz 600 MHz
SMJ = MIL-PRF-38535, QML								200 MHz 720 MHz
SM = High Rel (non-38535)								233 MHz
OMAP = OMAP								250 MHz 5E0 (500-MHz core, 100-MHz EMIF)
Device Family								300 MHz 6E3 (600-MHz core, 133-MHz EMIF)
32 or 320 = TMS320™ DSP Family								400 MHz 7E3 (720-MHz core, 133-MHz EMIF)
Technology								Temperature Range
C = CMOS								Blank = 0°C to 90°C, commercial temperature, default for C6000 DSPs
DM = Digital Media								Blank = -40°C to 100°C, default for C54x™ DSPs
E = CMOS EPROM								A = -40°C to 105°C, extended temperature (C6000 DSPs)
F = CMOS Flash EEPROM								A = -40°C to 85°C, extended temperature (C2000 DSPs)
LC = Low-Voltage CMOS (3.3 V)								H = 0°C to 50°C
LF = Flash EEPROM (3.3 V)								L = 0°C to 70°C
UC = Low-Voltage CMOS [3 V (1.8-V core)]								M = -55°C to 125°C
VC = Low-Voltage CMOS [3 V (2.5-V core)]								S = -55°C to 125°C (C5000 DSPs)
								S = -40°C to 125°C (C2000 DSPs)
								Package Type
								FN = 38-lead PLCC
								GDK = 548-pin plastic BGA
								GDP = 272-pin plastic BGA
								GDY = 289-pin MicroStar BGA™
								GEL = 181-pin PGA
								GFN = 256-pin plastic BGA
								GGU = 144-/169-pin MicroStar BGA
								GSW = 176-/240-pin MicroStar BGA
								GHH = 179-pin MicroStar BGA
								GHK = 257-/288-pin MicroStar BGA
								GJC = 352-pin plastic BGA
								GJL = 352-pin plastic BGA
								GLS = 384-pin plastic BGA
								GLW = 340-pin plastic BGA
								GLZ = 532-pin plastic BGA
								GNV = 284-pin plastic BGA
								GNZ = 352-/548-pin plastic BGA
								GZG = 289-pin MicroStar BGA
								GZZ = 201-pin MicroStar BGA
								PG = 64-pin PQFP
								PAG = 64-pin TOFP
								PBK = 128-pin LOFP
								PCM = 144-pin PQFP
								PGE = 144-pin LOFP
								PGF = 176-pin LOFP
								PQ = 132-pin PQFP
								PYP = 208-pin PowerPAD™ plastic QFP
								PZ = 100-pin LOFP
								VF = 32-pin LOFP

C6000 DSPs		Device		C5000 DSPs		C2000 DSPs		C3x DSPs	
6201	6701	549	5420	240	2810			30	
6202	6711B	5401	5421	241	2812			31	
6202B	6711C	5402	5441	242				32	
6203B	6712	5402A	5470	243				33	
6204	6712C	5404	5471	2401A					
6205	6713	5407	5501	2402A					
6211	DM640	5409	5502	2403A					
6211B	DM641	5409A	5509	2404A					
6411	DM642	5410	5509A	2406A					
6412		5410A	5510A	2407A					
6414		5416	5910						
6415		54CST							
6416		54V90							

For the actual device-specific part numbers, see the Product Specification Guides in this document.

APPENDIX C

Which TI DSP should one use?

There are several DSPs available from Texas Instruments. Typical device nomenclature is provided in Appendix B. In this appendix, a brief overview of C2000, C5000 and C6000 platforms are given along with their specific applications. The appendix concludes with the choice made.

C.1 TMS320C2000™ DSP platform

It provides the digital control industry with the highest level of on-chip integration to deliver system cost reduction, and powerful computational abilities that enable software innovation. The TMS320C28x™ DSP generation is the foundation for this diverse platform. This generation delivers power and control advantages that allow designers to implement advanced, cost efficient control systems. Listed below are a few platform specific applications:

- Industrial drives
- Servo motion control
- Power supplies
- Pumps, fans, HVAC
- Factory automation
- Consumer goods
- Appliance controllers
- Transportation
- Office equipment

C.2 TMS320C5000™ DSP platform

It is optimized for the consumer digital market- the heart of the mobile Internet- and its convergence with the other consumer electronics. The new TMS320C55x™ DSP generation delivers the most power-efficient DSPs ever, with a roadmap as low as 0.05 mW/MIPS and speeds up to 300 MHz. The C55x™ DSPs are completely software compatible with existing TMS320C54x™ DSPs, the established industry leader in power-efficient performance. Listed below are a few platform specific applications:

- 2G, 2.5G and 3G cell phones
- Portable products
- Wireless modems
- Digital audio players
- Networking
- IP phone
- Digital still cameras
- Voice over Packet
- Personal digital assistants

C.3 TMS320C6000™ DSP platform

It is optimized for highest performance and ease-of-use in high-level language programming. The C6000™ fixed-and floating-point DSPs anchor multi-service broadband infrastructure like 3G wireless, DSL and cable, plus other MIPS-intensive applications such as advanced digital imaging. The new TMS320C64x™ DSP core scales operating speeds beyond 1 GHz and achieves 10x performance improvements over the TMS320C62x™ DSP. Listed below are a few platform specific applications.

- DSL & pooled modems
- Wireless base stations
- Network cameras

- Digital imaging
- Voice over Packet
- Wireless LAN
- Central office switches
- Private branch exchange
- Speech recognition

C.4 DSP Starter Kit

The ECG algorithm is quite similar to the speech algorithms. The TMS320C6713 DSP Starter Kit (DSK) developed jointly with Spectrum Digital is a low-cost development platform designed to speed the development of high precision applications based on TI's TMS320C6000 floating point DSP generation. The kit uses USB communications for true plug-and-play functionality. Both experienced and novice designers can get started immediately with innovative product designs with the DSK's full featured Code Composer Studio v2.2 IDE and eXpressDSP Software which includes DSP/BIOS and Reference Frameworks.

VITA

②

Vivekanand Chengalvala

Candidate for the Degree of

Master of Science

Thesis: ECG COMPRESSION USING LPC AND ADAPTIVE CODE BOOK

Major Field: Electrical and Computer Engineering.

Biographical Information:

Personal Data: Born in the village of Narsapur, W.G. Dist, Andhra Pradesh, India on August 13th 1980.

Education: Graduated from Tetrahedron Junior College, Hyderabad India in 1997. Received Bachelors of Technology in Electronics and Communications Engineering from Nagarjuna University, Andhra Pradesh, India in 2001. Completed Master's Degree in Electrical and Computer Engineering at Oklahoma State University, Stillwater, Oklahoma, December 2003.

Experience: Employed by Oklahoma State University, Department of Electrical and Computer Engineering as a Graduate Teaching Assistant and Graduate Research Assistant from January 2002 to present (December 2003).