Circuits and Systems Group
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-MS-2014-08

M.Sc. Thesis

Guaranteed Quality ECG Signal Compression Algorithm

Dongni Fan

Abstract

The aim of the project is to develop an ECG signal compression algorithm that has a high compression ratio while guaranteeing signal quality. An electrocardiography (ECG) signal is a representation of cardiac activity and has an need to be compressed to reduce data storage requirements and energy cost of transmission. Previous ECG signal compression techniques have shown steady improvement on compression ratio. However, these techniques generally lack quality considerations, so their applications are limited. We present a discrete cosine transform (DCT) based compression scheme and use beat detection which considerably improve the compression ratio. The quality of the compressed signal is configurable, and the accuracy of the signal is maintained given a signal quality requirement.

The algorithm is implemented in a software/hardware solution. Some parts need to be done in the software. As a proof of concept, we have chosen the filter to be implemented in hardware. Mathworks HDL coder was used for generating RTL code and testbenches.

Results show that our algorithm is capable of maintaining the specified quality, has a better compression ratio compared to previous work and is also capable to compress noisy ECG signals.

Guaranteed Quality ECG Signal Compression Algorithm

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Computer Engineering

by

Dongni Fan
born in Ankang, China

This work was performed in:

Ultra-Low Power DSP Group
imec - Holst Centre
Eindhoven, the Netherlands

Delft University of Technology

Delft University of Technology
Department of
Software and Computer Technology
imec - Holst Centre
Ultra Low Power DSP Group

The undersigned hereby certify that they have read and recommend to the Faculty of Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Guaranteed Quality ECG Signal Compression Algorithm"** by **Dongni Fan** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: September 25, 2014

Chairman:

Prof.dr.ir. A.-J. van der Veen, TU Delft

Advisors:

Alex Young, imec - Holst Centre

Dr.ir. Rene van Leuken, TU Delft

Committee Members:

Dr.ir. Stephan Wong, TU Delft

Dr.ir. Rob Remis, TU Delft

Mario Konijnenburg, imec - Holst Centre

Abstract

The aim of the project is to develop an ECG signal compression algorithm that has a high compression ratio while guaranteeing signal quality. An electrocardiography (ECG) signal is a representation of cardiac activity and has an need to be compressed to reduce data storage requirements and energy cost of transmission. Previous ECG signal compression techniques have shown steady improvement on compression ratio. However, these techniques generally lack quality considerations, so their applications are limited. We present a discrete cosine transform (DCT) based compression scheme and use beat detection which considerably improve the compression ratio. The quality of the compressed signal is configurable, and the accuracy of the signal is maintained given a signal quality requirement.

The algorithm is implemented in a software/hardware solution. Some parts need to be done in the software. As a proof of concept, we have chosen the filter to be implemented in hardware. Mathworks HDL coder was used for generating RTL code and testbenches.

Results show that our algorithm is capable of maintaining the specified quality, has a better compression ratio compared to previous work and is also capable to compress noisy ECG signals.

Acknowledgments

I would like to express my great appreciation for Alex Young, for his thoughtful guidance, warm encouragement and constructive comments of this project. Thank you for your trust and patience. Without your support, this could never happen.

I would like to thank Dr.ir. Rene van Leuken for offering valuable advice and guidance for this project. I appreciate your patience and insightful comments during the correction of the thesis.

I would like to thank Mario Konijnenburg, Petri Solanti, Benjamin Busze and Stephan van Beek for their support and assistance during the hardware implementation.

My grateful thanks also goes to imec - Holst centre, which gives me the chance to carry my thesis work with those excellent people. I have been blessed to enjoy the previous 10 months with a friendly group of people.

Furthermore, I am also thankful to all the people that helped me, encouraged me and even questioned me during my thesis work. Your advice is my motivation of completing the work.

Last but not least, I would like to express my deep gratitude to my parents, who trust me and support me as always. You are always the motivation and the reason to make me as the person who I am.

Dongni Fan
Delft, The Netherlands
September 25, 2014

Contents

List of Figures

List of Tables

List of Abbrievations

AC ..... Alternating Current

ASCII ..... American Standard Code for Information Interchange

ASIC ..... Application-Specific Integrated Circuit

AVQ ..... Adaptive Vector Quantization

AZTEC ..... Amplitude Zone Time Epoch Coding

CCSP ..... Cardinality Constrained Shortest Path

CPU ..... Central Processing Unit

CR ..... Compression Ratio

CWT ..... Continuous Wavelet Transform

DC ..... Direct Current

DCT ..... Discrete Cosine Transform

DFT ..... Discrete Fourier Transform

ECG ..... Electrocardiography

RTL ..... Register Transfer Level

HDL ..... Hardware Description Language

FLAC ..... Free Lossless Audio Codec

FAN ..... Fanout-oriented

FFT ..... Fast Fourier Transform

Fs ..... Sample Frequency

FPGA ..... Field Programmable Gate Arrays

JPEG ..... Joint Photographic Experts Group

KLT ..... Karhunen-Loeve Transform

LNPV ..... Last Non-zero Position Vector

LTV ..... Look-up Table Vector

MDCT ..... Modified Discrete Cosine Transform

MSE ..... Mean Squared Error

MP3 ..... Moving Picture Experts Group - 3

PRD ..... Percentage Root Mean Square Difference

PSO ..... Particle Swarm Optimization

QS ..... Quality Score

RAM ..... Random Access Memory

RE ..... Recursive Encoding

RLE ..... Run-length Encoding

RMS ..... Root Mean Square error

ROI ..... Region of Interest

SAPA ..... Scan-Along Polygonal Approximation

SNR ..... Signal to Noise Ratio

S-OMP ..... Simultaneous Orthogonal Matching Pursuit

SPIHT ..... Set Partitioning In Hierarchical Trees

STD ..... Standard Deviation

TH ..... Threshold

VHDL ..... Verilog Hardware Description Language

VQ ..... Vector Quantization

WDD ..... Weighted Diagnostic Distortion

WPL ..... Weighted Path Length

WT ..... Wavelet Transform

1 Introduction

The electrocardiography (ECG) signal is a representation of heart electrical activity. Medical research has used ECG signal to investigate and improve treatment of heart diseases for many years [54]. In this thesis we focus on the problem of ECG signal compression. We aim at developing an algorithm that is capable of achieving high compression ratio and specified quality.

This chapter is an introduction to the thesis, where the motivation, the contributions and the organization of the thesis are described.

1.1 Motivation

The ECG signal has been successfully used in diagnosis of heart disease for a long time [54]. The ECG signal shows the speed and rhythm of heart beats among others, which are relevant to heart activity. Great progress was made in the early 20th century, where the PQRST complex [24] was introduced to model the typical ECG cycle. It simplifies analysis by naming each part of the typical waveforms/deflections by capital letters. Combinations of PQRST complex may indicate different heart activities, for example, a short QT interval may indicate the hypercalcemia, which causes the abnormal heart rhythm [47]. The PQRST complex of a normal ECG cycle is shown in figure 1.1.

Usually, ECG signals are measured in hospital by the traditional 12-lead ECG monitor, which is big and heavy. It prevents and complicates the prediction of disease. A patient's situation is analyzed only after he/she is sent to hospital, while often the case can be very urgent and risky. Patients and doctors are looking for convenient ways to record their heart activities in the daily life, so as to achieve 24-hour monitoring. Consequently, portable/wearable ECG monitors are developed. These devices are usually required to be small-size and low power. They are preferred to have real-time display and analysis of the captured ECG signal. Thus the device can measure and track ECG signals at home, privately and continuously.

A modern ECG monitor includes a number of hardware components and different functionalities. An example was shown by IMEC-NL [32], where a mixed-signal SoC was developed to support a range of applications for ECG signals. Despite of a variety of issues involved in the portable ECG monitor, such as size, speed and cost, storage limit and transmission capacity motivate development of ECG signal compression. Stored ECG signals are presented in digital form. The resolution of ECG signals are often above 10 bits/sample, and the sampling frequency can be as high as 512 hz. This means a real-time storage over just 1 minute requires 37.5 kB of memory. Bit representation of the original signal will be a huge burden for storage. In addition to this, a lot of energy are cost for transmission. The battery life of the wearable ECG monitor

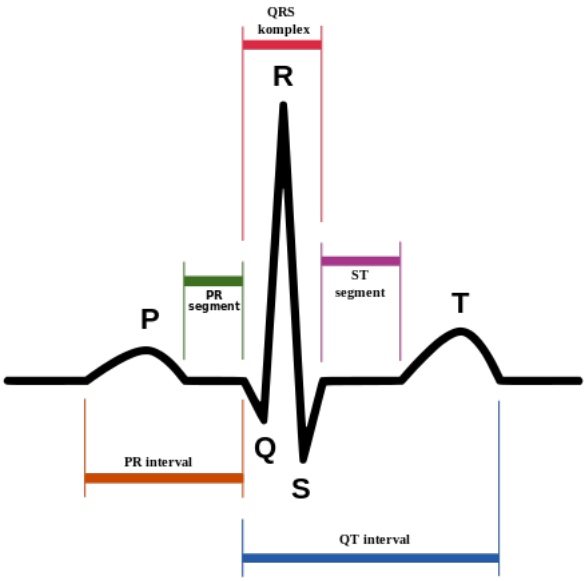


Figure 1.1: Schematic diagram of normal sinus rhythm for a human heart as seen on ECG [49]

limits the energy used for transmitting a large size data. ECG signal compression is feasible, because ECG signals are very regular: stable ECG records are approximately periodic, in other words, redundancy is almost certain. This property is virtuous for compression, since correlations between each cycle can be utilized and some prediction schemes can be adopted. Besides, ECG signals are usually within a certain frequency band. Frequency domain transforms are very likely to offer the best compression ratios. A couple of existing transform-based compression techniques are available and are proved success and efficiency in a number of fields of signal processing, such as the JPEG standard [57] in image processing and the MPEG compression [43] in audio processing. These transform-based algorithms are usually lossy and both are widely used today.

Compression algorithms can be implemented in software and hardware. There is a trade-off between the software implementation and the hardware implementation, where the software provides more flexibility while the hardware allows real-time operation and gives much higher energy efficiency. The combination of software and hardware is preferred in our case, where a CPU is used for ordering the tasks and several customized hardware components are implemented, for example, to accelerate certain computing-intensive functions. Compared with purely software execution, customized hardware implementation can reduce the energy consumption. Several designs of such kind of systems are presented in [33], in which the benefits from hardware implementation are clearly shown.

The issue with the existing lossy ECG signal compression algorithms is that, they commonly lack considering the quality control of the decompressed signal. However, the ECG signal quality is always important and affects the diagnosis very much. This motivates us to design an algorithm that is capable to specify and guarantee the quality.

1.2 Goals

The major goal of this project is to design an ECG signal compression algorithm that is quality-controlled and as a way of compression, can significantly save storage space and/or decrease bandwidth requirement for transmission. The tasks involved include exploration of current ECG compression techniques, development of a compression algorithm, and a hardware implementation of a particular block as an example.

1.3 Contributions

The contributions of this thesis are:

- A novel ECG signal compression method is proposed. It not only achieves high compression ratios but also can maintain the reconstruction quality. Users can specify the desired quality.

- Several utilities for ECG signal processing are developed, including beat (R peak) detection, band-pass filtering, high quality re-sampler and quality controller. They can also be used as regular tools for ECG signal processing.

- A 2-stage lossless data encoder/decoder based on the recursive arithmetic encoding is presented. They improve the compression ratio by 25%.

- A hardware implementation of a component (the filter/oversampler) is presented and simulated. The design is developed using Simulink and the RTL code is generated by the Mathworks HDL coder.

1.4 Outline

The thesis is organized as follows.

Chapter 2: Related Work In this chapter, we start with a brief introduction to some basic concepts of data compression. Thereafter, a literature review of techniques and categories of ECG data compression is presented.

Chapter 3: Quality Configurable Algorithm Development This chapter presents the development of the compression algorithm. It also includes exploration, analysis and discussion of essential blocks in the algorithm.

Chapter 4: Filter Implementation This chapter describes hardware implementation of a filter and the workflow as an example.

Chapter 5: Conclusions and Future Work This chapter concludes the project and discusses future work.

2 Related Work

ECG signal compression is an application of the data compression technique. Generally speaking, data compression is a method that represents information by fewer bits than the original representation. Compressed data usually are not directly interpretable. A decompressing process is required to reconstruct information. Theoretical support on data compression includes information theory, coding theory, statistical redundancy, etc.

The distortion between original information and the reconstructed/decompressed information determines the type of compression is. Lossless data compression will recover data without any information loss, while lossy data compression will have some distortion from the original information, but the compression ratio usually is higher than lossless data compression. Lossless compression contains all the original information, which makes it suitable for any application. Specifically, areas such as text compression and cryptography require special attention to accuracy, where lossless compression is very useful. Typical lossless compression techniques are Run-length encoding for general purpose and free lossless audio codec (FLAC) for audio compression [15], etc. However, in some scenarios, for example JPEG [57], which is a widely used standard image format, a slight information loss may lead to a great saving in storage space. This loss does not disturb users' experience: human eyes cannot differentiate the tiny distortion if it is small enough. A comparison of 2 JPEG images of the different quality loss is shown in the figure 2.1, which are resulted from the same lossless image. The larger the Quality value is, the higher the quality is. There are no significant difference between the two images in the figure 2.1, however the data size of the high quality image is nearly 4 times than the low quality image. Thus to reduce the storage requirement, lossy compression can be a better choice.

A compressor work as an encoder, which converts the input to another form, but usually with fewer bits. Therefore, a great amount of entropy encoders are available for different kinds of compression, even though they are general purpose. However, a compressor is certainly more than an encoder, especially in the real world. Transforms are generally involved, and some mathematical methods or models are also applied. All these components contribute to maximizing the compression performance by preprocessing the input data before sending to the encoder. In summary, the different components and methods increase the variety of data compressor designs, and make space for further research in our work.

Although ECG signal compression is an application of data compression, it has special features and depends on specific developments compared to the general purpose methods. A categorization of ECG compression was introduced by [1], where techniques are categorized as *dedicated* or *general*. Dedicated techniques perform compression mainly from the special property of the ECG signals on time-domain while general techniques involve transforms and have similar schemes as compressors in other fields,

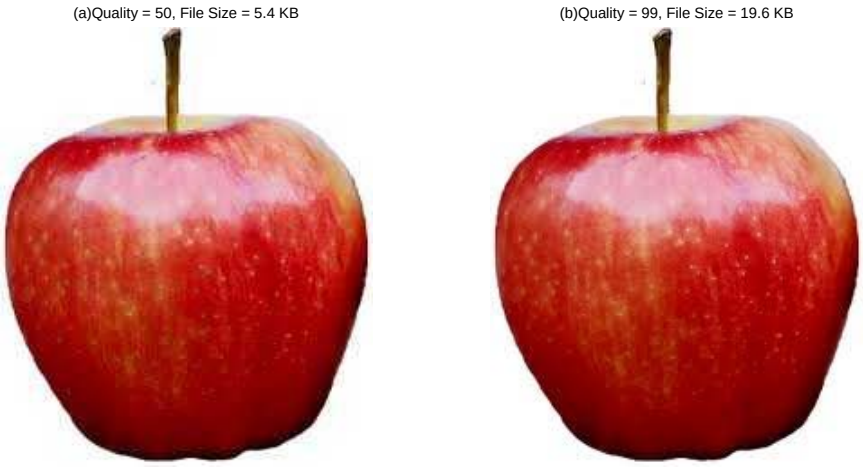


Figure 2.1: Example JPEG Images with Different Quality Specification

e.g., JPEG. We follow this categorization and the study is presented as follows.

2.1 Dedicated Techniques

Dedicated techniques try to extract and utilize time-domain features from ECG signals. This could directly be used as compressing data. Several algorithms belong to this category, e.g., AZTEC [17], SAPA [29], CCSP[26] or even direct entropy encoding [16]. However, most of them were developed many years ago, their compression ratios are not as high as the modern techniques. Research on data compression in the 21th century are more in the category of general techniques, where a lot of novel ideas from data compression can be applied. On the other hand, ECG compressors that make use of general techniques may also work in other applications, which extends the functionality. Due to the above reasons, during this thesis project, we concentrate on the general techniques for ECG compression exploration. This section will introduce 2 typical dedicated techniques: FAN and CCSP.

2.1.1 FAN Algorithm

The FAN method is originally an adaptive sampling method that is used for compressing cardiac waveforms [21], it is a heuristic approach. The idea is to not save all the sample points but only save points that have slopes differing below specified thresholds ε from their neighbors' slopes. The process is repeated point by point so that saved samples are always within the space where the shape approximates the original signal. Errors

are therefore guaranteed to be in certain range.

An evaluation of FAN method on ECG compression was presented in [18]. Relations between the threshold value ε and errors were analyzed. This type of ECG compression algorithm is also generalized as 'Scan-Along Polygonal Approximation' (SAPA) by M. Ishijima in [29]. The idea underlying SAPA is similar to FAN: yielding an approximation that has a desired error from the original input by "scan-along" the signal. An illustration is shown in figure 2.2, where ϵ represents the allowable error between the original signal and the reconstructed signal and can be specified. The approximation itself is regarded as a subset of the original signal, where fewer points are presented to reduce the data storage requirement. Three SAPA methods are proposed and regardless of their differences in criterion, they all achieve a good performance. The compression ratio can be as high as 11 for the 500 hz sampled data, while the error specification was not discussed.

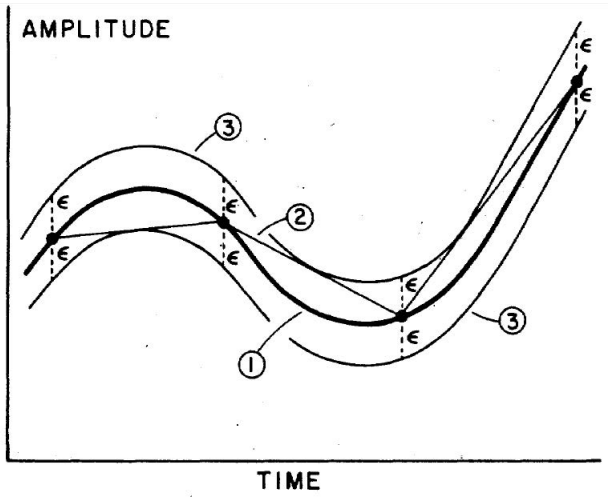


Figure 2.2: Basic concept of polygonal approximation. (1) Original Signal. (2)Polygonal approximation of original signal. (3) Approximation error boundary. [29]

Recent years a relevant enhanced FAN algorithm, FAN+ [10], was developed. It incorporates the FAN method with several advanced algorithms, e.g., Douglas-Peucker line simplification algorithm [56], which increases the compression ratio by 1.5 to 2 times.

2.1.2 CCSP Algorithm

The Cardinality Constrained Shortest Path Problem (CCSP) algorithm [26], on the other hand, frames the problem as to find the shortest path from the first sample point to the last given certain constraints, i.e., quality requirement. It is identified as the resource-constrained shortest path problem.

The goal of CCSP is to minimize the path length, namely the number of vertices in the path. At the same time it should satisfy the reconstruction error requirement. The algorithm starts with assuming a minimal value for the desired number of vertices, whose notation is $m*$, then iteratively searches for the shortest path from the starting

point to j until j reaches the last point. And quality constraints and $m*$ are evaluated and updated within a loop, so that eventually the desired sample points and other side information e.g., distance are extracted as compressed data.

A comparison between the CCSP and the FAN algorithm was shown in [26]. It showed that CCSP was capable to reduce data size more significantly than the FAN algorithm given the same quality specification, especially when the quality requirement is low.

2.1.3 Others

In addition to previously mentioned algorithms, entropy encoding is also used in some compression schemes. A modified Huffman coding technique was introduced by [16] and applied by [45]. This method used the interpolation scheme and results an approximately 0.5 bit smaller entropy for the same quantization step size than prediction scheme. Differential schemes and interpolations are also studied and applied in dedicated techniques.

2.2 General Techniques

General techniques root from compression techniques of other areas, mostly from audio and image compression. Techniques from these areas are numerous and have a great variety. Development of most new compression techniques also gives priority to audio signals or image data. This inspires the use of them for ECG data compression. The ECG signal can be approximated by audio signals, or by some transformations to image data. So the compatibility is always good between compressing ECG and audio/image. Modern general techniques for ECG compression include transform-based techniques, Vector Quantization (VQ) [40], compressed sensing, etc. Hybrid methods are also frequently used to exploit maximal performance from each block.

2.2.1 Transform-based Technique

Figure 2.3: Simplest Transform-based Compression

Transforms are a set of techniques that map the input data to another set. For instance, the Discrete Fourier Transform (DFT) can map a signal to the frequency domain by converting the input to complex sinusoids. This type of technique has

been widely used in the signal processing field. The transformed data may have better numerical features for compression than the original signal form. Assuming storing a sinusoidal signal over a period of 100s, the number of bits used in time-domain representation is highly related to the amplitude and the length of the signal. But after transformation to the frequency domain, information is concentrated in a certain frequency range. Perfect reconstruction is also possible by inverse transform. The transform-based compression technique, or transform coding, is thus developed. A block diagram of the simplest transform-based compression is shown in figure 2.3.

Quantization is to constrain the number of bits used for representing the transform result. Usually it causes a slight distortion to the original data, which makes transform-based techniques lossy. Choice of the entropy encoder scheme is critical to the performance, especially for compression ratio. The type of transform characterizes a transform-based compression algorithm. Considerable types of transforms in mathematics can be utilized for data compression. Cosine/Fourier Transform and Wavelet Transform are used most frequently for compression.

2.2.1.1 Discrete Cosine Transform Compression

The Discrete Cosine Transform (DCT) is successfully applied in both audio (e.g., MP3) and image (e.g., JPEG) compression. It is similar to the Discrete Fourier Transform (DFT). Because of being a real transform, it is computationally more efficient than DFT. An illustration of the DCT process is shown in figure 2.4, where clearly most of DCT coefficients are low frequency components close to the DC. This makes it energy compact, which is favorable in compression.

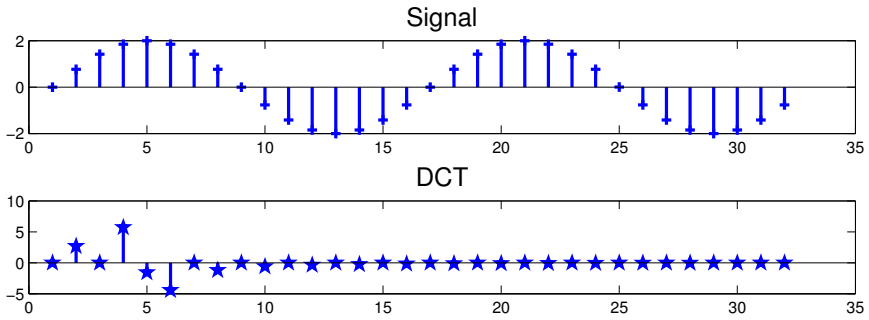


Figure 2.4: DCT Example

V.A. Allen first proposed the application of DCT in ECG data compression in [2]. This work proves the applicability of the DCT method. However, the result is not very good with regard to the compression ratio. Recently an improved DCT ECG compression method was proposed in [8], where the quantization and the encoding method were improved. This method significantly enhances the performance and make the compression ratio comparable or better than other work from the time.

The most commonly used 'DCT' is actually a subset of the DCT family, of which the type is DCT-II. The modified DCT(MDCT), namely DCT-IV, is sometimes used for data compression. It helps to avoid block boundary artifacts but the pre- and post-

processing to recover the signal are somewhat complex. Aside from those, the DCT in 2 dimensions is also considered for ECG data compression. One of such method is presented in [34]. This method finds the beat out from the signal, maps each beat into a matrix and then applies 2D-DCT. Image processing concepts such as JPEG have been also used in ECG data compression. Because the ECG signal is represented by limited number of bits. Gray-level images are represented by gray-levels, then we can use compression of image processing. So a matrix made from the ECG signal can be regarded as a gray-level image, where JPEG concepts may be feasible for compression. A performance comparison of the above DCT methods is shown in the table 2.1.

Table 2.1: Performance Comparison of DCT-based Compression Algorithms

Algorithm	Sample Rate	Resolution	Compression Ratio	Quality Specification
[2]	5 khz		2.8	MSE = 0.2
[8]	360 hz	11 bit/sample	5.0	PRD1 = 5.0
[34]	250 hz	12 bit/sample	4.0	PRD0 = 1.9

2.2.1.2 Wavelet Transform Compression

The Wavelet transform (WT) is a transform that makes use of both time domain features and frequency domain features of an input signal. The WT process is a decomposition process, where a high-pass filter and a complementary low-pass filter are applied in each decomposition. The decomposition is called recursively as the number of stages incremented. So for every stage, 2 versions of the input signal are generated. Apparently, the more stages are, the fewer frequency components are in the output signal. The property of filters is defined by wavelet coefficients. Figure 2.5 shows an example of a 1 level wavelet transform, where the signal is de-composited into 2 parts: approximations representing lower frequencies and details representing higher frequencies.

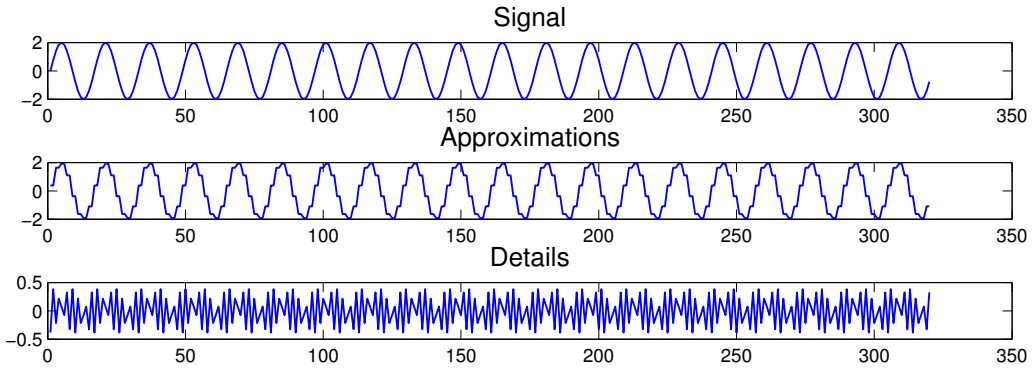


Figure 2.5: Wavelet Transform

The most famous wavelet ECG compression algorithm is set partitioning in hierarchical trees (SPIHT) [46]. This algorithm is designed for wavelet coding and was

initially used for image compression. The fact that a WT has a hierarchical tree structure inspires the partitioning and sorting of subsets of coefficients. Thereafter significant information is stored compactly. [36] applies such a scheme for ECG compression and the result is very good.

[31] is a WT method that uses waveform partitioning and adaptive frame size adjustment. While [52] is a method similar to [34] in section 2.2.1.1 but it uses WT instead of 2D-DCT. Both algorithms claim a good compression ratio, which indicates the WT is also a good choice for ECG data compression. A performance comparison of the above WT methods is shown in the table 2.2.

Table 2.2: Performance Comparison of WT-based Compression Algorithms

Algorithm	Sample Rate	Resolution	Compression Ratio	Quality Specification
[36]	360 hz	11 bit/sample	4.0	PRD1 = 1.19
[31]	360 hz	11 bit/sample	7.9/22.0	PRD0 = 1.1
[52]	360 hz	11 bit/sample	10.0	PRD1 = 1.57

2.2.1.3 Others

The Karhunen-Loeve transform (KLT) for ECG compression is introduced in [42]. [41] uses the Hilbert transform with ASCII representation. [5] incorporates Burrows-Wheeler transform with linear prediction for lossless compression, a slight modification to this algorithm can make it general purpose.

2.2.2 Other Techniques

2.2.2.1 Compressed Sensing

Compressed sensing is a breakthrough technique that was proposed in the past few years [20]. It presents an idea of reconstructing a signal by the knowledge of its sparsity, instead of by its highest frequency. From compressed sensing, fewer samples are needed than the Nyquist Shannon theorem, which causes a big impact in signal processing, as well as in data compression.

Simply, for a signal $x \in R^{N\times 1}$, there are M linear measurements, where each of them is noted as s:

$$s = \Phi x, \Phi \in R^{M\times N} \tag{2.1}$$

By the linear measurement s and the measurement matrix Φ, the signal can be perfectly reconstructed. In compression, only s, which is shorter than original information x, will be used for storage, so the problem becomes how to deduce Φ. And fortunately, there are methods to solve this problem, for instance, orthogonal matching pursuits. The matrix of s is sparse, which improves the storage efficiency, as well as compression efficiency. The scheme of compressive sensing is shown in figure 2.6.

The matrix Φ here plays as a sensor that acquire information from input x. And it also can be seen that the transform from x to s approximates a type of transform discussed in figure 2.3.

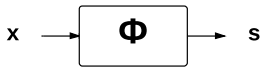


Figure 2.6: Compressive Sensing

In recent years a lot of methods of compressed sensing for ECG data compression have proposed. [19] presents an analog-based compressed sensing acquisition system and an encoder architecture. [44] integrates compressed sensing as a block in a compression system that has similar features as [34]. While [38] proposes a more direct method of using compressed sensing for ECG compression, claim energy efficiency and its hardware implementation.

2.2.2.2 Multichannel Compression

ECG signals are generally multichannel. Some compression techniques have special consideration for this.

A method of preprocessing multichannel data is presented in [11], where channels are reordered to improve transform efficiency. Adaptive vector quantization (AVQ) is adopted in [39], but the quality of the reconstructed signal is low, where the PRD0 can be as high as 17.8%. A coder structure based on AVQ is proposed. [35] uses a sparse approximation method, applying simultaneous orthogonal matching pursuit (S-OMP) for ECG multichannel compression, the compression ratio can be as high as 12.8 when PRD0 is 5%.

2.3 Discussion

From the survey of the ECG compression algorithms above, it is clear that there are a great many choices for this purpose. Those algorithms vary in computing complexity, implementation difficulty, execution speed and compression ratio. It is difficult to determine which one is superior than others, unless the specification is given. However, with respect to development, there is a trend from dedicated technique to general technique. Currently in the category of general techniques, transform-based compression is the most developed one as it succeeds in image and audio processing, which are in great demand nowadays.

The remaining of techniques are also interesting, especially for compressive sensing, which is novel and may bring a lot of improvements, but the performance is uncertain. Time-domain/dedicated techniques are useful, but they did not show very excellent compression ratio and the quality of the decompressed signal can also be low. Other considerations such as multichannel data and lossless compression are interesting to investigate.

However, even though these techniques are interesting regarding to technology, they commonly lack consideration of quality measurement and control. Various quality measurements can be found in the literature as well, but the reason of the choice of the using measurement was rarely mentioned. This makes many of the algorithms impractical in real applications. In our development, we aim at improving the compression ratio whilst maintaining the quality of the signal given the user's specification. In addition, due to the amount of the different quality measurements of the ECG signal compression, it was difficult to make a fair comparison of the compression performance among existing algorithms. However recently, the percentage root mean difference (PRD) appears frequently in the literature, which is also preferred by our design. We will discuss the quality measurement in detail in the section 3.1.3 and the method of quality control in section 3.2.3.

Quality Configurable Algorithm Development

3

In this chapter, the development of the algorithm is discussed and a novel and efficient quality-controlled ECG signal compression algorithm is proposed.

We begin with analysis of the choice of algorithm scheme and in particular, the quality measurement. Afterwards, we describe a direct and light-weight method of achieving our goal based on our preferences. Next, we will show that improvements can be made upon such an algorithm. Lastly, a optimized algorithm with better performance is proposed.

3.1 Design Considerations

A transform-based compression algorithm has the following scheme as shown in figure 3.1. A compression algorithm contains a compressor that compresses the input data and a decompresser that reconstructs the data by using the compressed results. In the compressor part, the block of transform is critical to transform the data into the representation that should be sparse. Quantization block increases the step size and reduces the number of bits, but keep the step size small enough. So we do not lose so much information. Encoding is a process that represents the data in another form with fewer bits than the original representation, which reduces data size without distortion. The corresponding decoding method must also exist so that the original understandable information can be recovered. We prefer a lossless encoding because we want to keep the lossy part in the quantization block and manipulate the quality there. The decompresser is exactly the inverse process of the compressor.

This basic scheme works for most cases but it lacks practical considerations. In the following sections we will look at the practicalities of such method.

3.1.1 Type of Compression

As discussed in the last chapter, modern compression algorithms are usually of general techniques. Particularly, transform-based techniques stand out for the wide applications in image and audio processing. As for image and audio, ECG signals are usually regular, which means that most of the frequency components are centered in certain ranges. This property is used by many transforms, especially the FFT and DCT separate the frequency into the useful frequency band. In our case we prefer these because IMEC itself has a hardware implementation of FFT and it is possible to be used for accelerating compression process. Furthermore, the DCT outperforms FFT because its quantized output is more compliant to the encoder. The implementation of DCT can be derived from using FFT blocks, which simplifies the design. All in all, our design is based on the frequency-domain transforms, in particular the Discrete Cosine Transform (DCT).

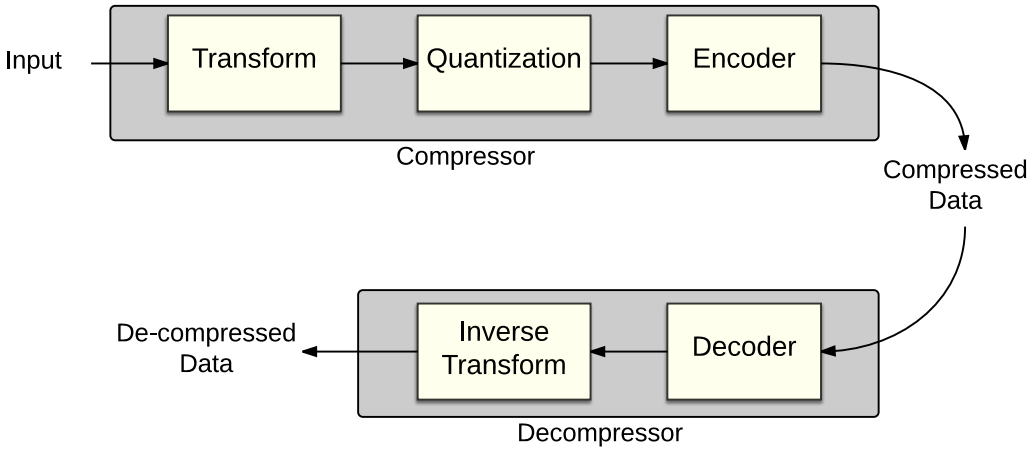


Figure 3.1: Basic Transform-based Compression Scheme

We also consider the issue of lossy and lossless. Generally the lossy compression will give a higher compression ratio than lossless compression. Transform-based compression is usually lossy as well. In our case we observed that with a reasonable loss of information, the morphology of ECG signals stay the same. We give lossy compression priority in our design considerations.

3.1.2 Quality Control

A practical compression algorithm should not focus totally on compression itself. Many applications have requirements for the quality of the de-compressed signal, but we find that many examples in previous work that lack of quality control, even though they can achieve a high compression ratio. A robust compression algorithm should have the ability to maintain the quality of the de-compressed signal while achieving reasonable compression ratio. This is because only good quality signal reconstruction makes sense in reality. We have implemented a quality control block in the system. The design scheme is presented as in figure 3.2.

Since we assume that both encoder and decoder are lossless, the only lossy part in figure 3.1 is quantization. Tuning parameters of the quantizer will affect the level of loss so as to affect the quality of de-compressed signal. This means quality can be controlled. The quality control block actually controls the quantization block. Only when the control block validates that the quality is acceptable, the quantization block can send its result to next stage.

3.1.3 Measurement

Quality of lossy compression is usually determined by comparing the de-compressed data and the original data. If there are no differences at all, then the compression is lossless. Conventional measurements are based on mathematical distortions, such as

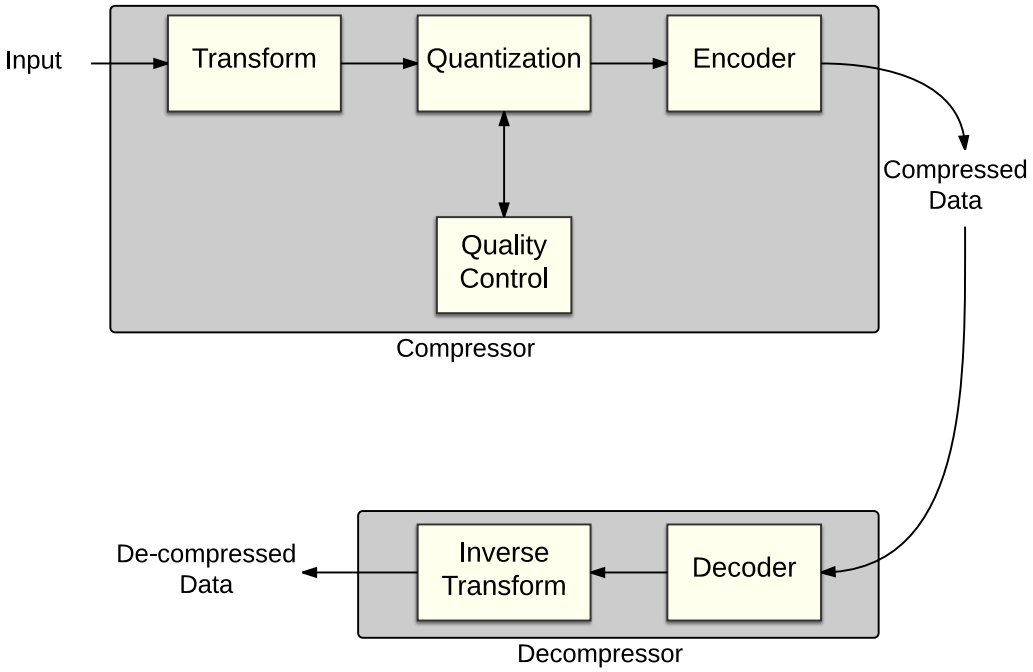


Figure 3.2: Compression Scheme with Quality-Control

percentage root mean square difference (PRD) and signal-to-noise ratio (SNR), etc. These measurements are not specific for ECG signals, they reflect the distortion of signal by statistics criteria. They are of general purpose, so the criteria may not be very accurate to describe the characteristics of a specific signal type. For example, the ECG signal is for medical use, so what concerns medical specification most is the diagnostic feature, which is not covered in the general mathematical descriptions. The Weighted Diagnostic Distortion (WDD) [60] is thus proposed. It uses diagnostic features as criteria and has better feedback from the perspective of medical specialists than other measurements. Table 3.1 shows a comparison between different measurements.

Table 3.1: Comparison between Different Quality Measurements

Measurement	Definition	Feature
PRD	Percentage root mean square difference	
RMS	Root mean square error	
SNR	Signal to noise ratio	often used for data transmission
STD	Standard Deviation	
WDD	Weighted diagnostic distortion	utilization of diagnostics features

Among those, PRD is the most widely used measurement in ECG data compression. WDD may not bring the benefit and it is far more complex, though it takes the ECG

features into consideration. PRD, SNR and STD are mathematically related. PRD is derived from RMS, which measures the power of errors between the original data and the reconstructed data and is used frequently for prediction. The advantages over RMS is its scale-independence [28]. That makes it more accurate across different data sets. There are subtle differences in calculation, PRD has 3 types for ECG data compression, which are numbered 0, 1, and 2. The definitions are listed as following equations(3.1), (3.2) and (3.3).

$$PRD0 = \sqrt{\frac{\sum_{n=1}^{N}(x[n] - \hat{x}[n])^2}{\sum_{n=1}^{N}(x[n])^2}}\% \tag{3.1}$$

$$PRD1 = \sqrt{\frac{\sum_{n=1}^{N}(x[n] - \hat{x}[n])^2}{\sum_{n=1}^{N}(x[n] - offset)^2}}\% \tag{3.2}$$

$$PRD2 = \sqrt{\frac{\sum_{n=1}^{N}(x[n] - \hat{x}[n])^2}{\sum_{n=1}^{N}(x[n] - \bar{x})^2}}\% \tag{3.3}$$

x is the original signal that has a length of N. $\hat{x}$ is the reconstructed/predicted signal and $\bar{x}$ is the mean value of x. Compare to PRD0, PRD1 is optimized by subtracting the offset, which is usually added from database for data storage. PRD2 is further optimized by subtracting the mean value (approximately the DC component). The result is thus more accurate removing a lot of effects from DC offset. An illustration on how the same value of different types of PRDs can affect the reconstructed signal quality is shown in figure 3.3.

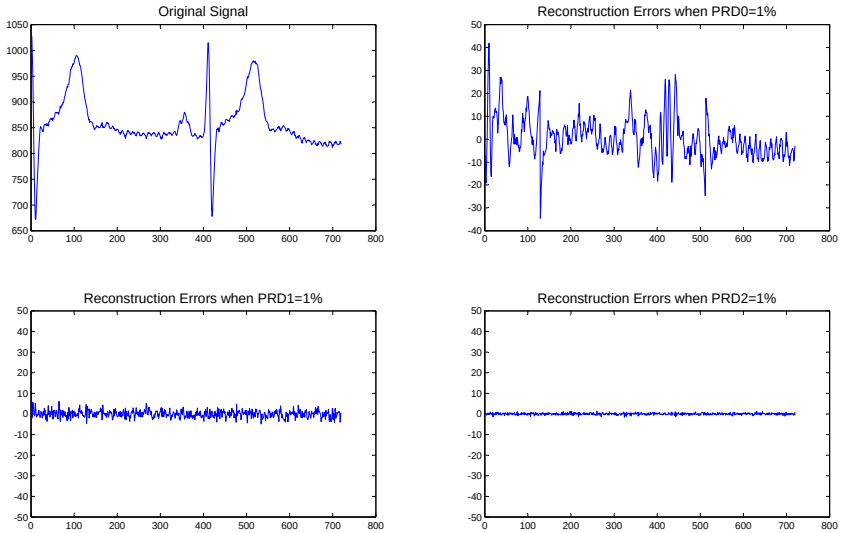


Figure 3.3: Reconstruction Errors with Different Types of PRD; Record 117, Channel 1

From the figure 3.3 we observe that the reconstruction quality is increased from PRD0 to PRD1. With PRD0 of 1%, there are a lot of loss in detail from original signal. For PRD1 and PRD2, the reconstructed signal has acceptable quality while for PRD2, the reconstructed signal presents more details.

Above all, in our discussion and experiments, we will focus on measurements of PRD1 and PRD2. PRD1 is more popular in literature because of its simplicity but we prefer PRD2 because it is more accurate.

Table 3.2 shows the connection between the quality indication and the PRD values (PRD1) [60].

Table 3.2: PRD1 Ranges and Quality Connection

PRD1 (%)	Quality
0-2	Very Good
2-9	Good
9-19	Not Good
19-60	Bad

Besides the quality in terms of reconstruction, the capability of compression is the center criterion that matters in data compression. Upon this compression ratio (CR) is the dominant measurement. The CR is the driver to make the method practical. It is defined by comparing sizes of original data and compressed data, namely:

$$CR = \frac{OriginalSize}{CompressedSize} \tag{3.4}$$

The definition is very obvious to reflect how much data is reduced during the compressing. This measurement significantly affects the data storage and transmission requirements.

3.1.4 Correlations

ECG signals are quasi regular signals, especially for a healthy person who has normal heart beats. The ECG record for each heart beat should be very similar, or at least has approximate trends. This indicates that the information preserved from ECG is of limited size, whereas redundancy must exist. This property does benefit compression a lot. Predictions can thus be made, and some data processing can utilize correlations between each cycle/beat. This is discussed in section 3.3.

3.1.5 ECG Database

To test and compare our compression algorithm, we choose the most popular input source, the MIT-BIH Arrythmia database [22]. This is from a research by MIT and

Beth Israel Deaconess Medical Center starting from 1975. Each record has 2 channels and the sampling frequency is 360Hz. The nominal resolution is 11-bit, each value has been offset by 1024 for the sake of storage. We have used this database for most of our experiments. The datasets used include record 100, 103, 104, 107, 108, 111, 112, 115, 116, 117, 118, 119, 201, 207, 208, 209, 212, 213, 214, 228, 231, and 232.

An IMEC-NL ECG record is also available. It is a 4-channel 12-bit signal with a higher sampling frequency of 512Hz. Since it is measured on a person during the playing of video games, it contains more noise and motion artifacts than MIT-BIH database. We use this as a reference to show the performance of our algorithm in practice.

As mentioned in Chapter 2, we only considered the single channel in our algorithm development.

3.2 A DCT-based ECG Compression Algorithm

With the above considerations, our design will be a DCT-based algorithm with quality-control and several other optimizations. A direct DCT-based compression scheme is proposed in [8], of which the scheme is shown in figure 3.4. The scheme is simple but the result is good with regard to compression ratio. We adopted this scheme to start our design.

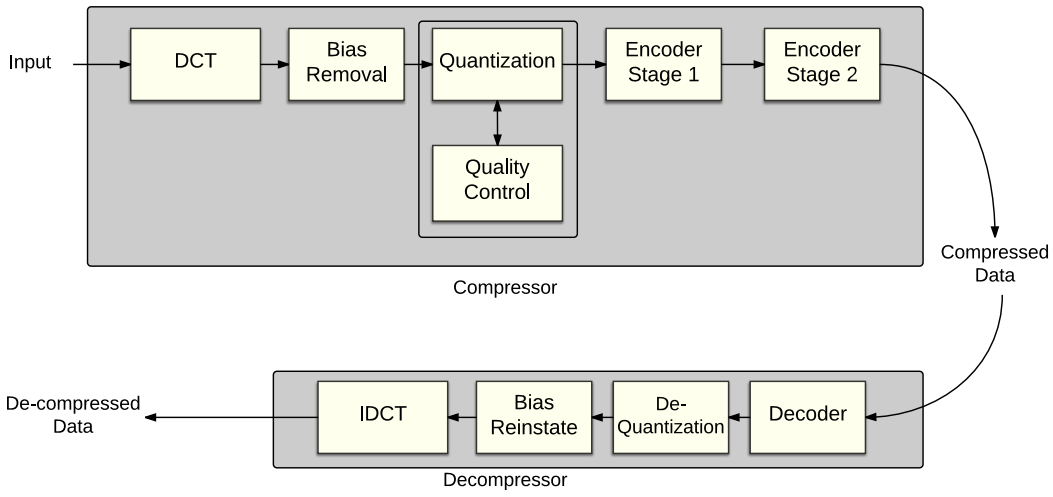


Figure 3.4: Architecture of the Direct DCT-based Compression Method

3.2.1 Discrete Cosine Transform

Given a time-domain input x(n), the basic DCT (also called DCT-II) is defined as follows:

$$y(k) = w(k) \sum_{n=1}^{N} x(n) cos(\frac{\pi(2n-1)(k-1)}{2N}) \tag{3.5}$$

where k = 1, 2, ...N and

$$w(k) = \begin{cases} \frac{1}{N} & k = 1 \\ \sqrt{\frac{2}{N}} & 2 \leq k \leq N \end{cases} \tag{3.6}$$

This is also the default DCT function that is used by Matlab. DCT has many similarities to the FFT, for example, time-domain to frequency-domain conversion. However, the DCT only expresses the signal as a sum of cosine functions. The DCT for this type of signals can produce a sparser matrix than the equivalent FFT. DCT can only be applied to a finite number of points, but the input ECG signal stream could be considered as infinite. In our practical design we have to divide the input signal into blocks of equal-length and the block length should be a power of 2. Due to that the DCT is an accumulating process, the output is scaled by DCT length, more bits are needed to represent these values. However, shorter length does not always imply the improvement. If the DCT segment is too short, the boundary discontinuities become very obvious, which in turn affect the reconstruction quality. For short DCT segments, output data will not be very sparse. Sparse data is what we are looking for in the compression algorithm. This will definitely increase the number of bits of compression. We did tests on various block lengths using the same compression scheme for a dataset from MIT-BIH database. For each record we chose the first 1 minute of the first channel and the PRD1 is 1%. The sample rate is 360 hz. In latter sections we will use this setting as default if not indicated specially. The result is shown in figure 3.5.

As discussed, both the larger length (512) and the smaller length (16) do not benefit the compression ratio as shown in the figure 3.5. The optimal block length here is 64 for 360 hz sample rate, which is not too large or too small. The block length should scale with the sample rate.

The DCT coefficients of each DCT block has the same length. Thus we can form a matrix from them. The matrix of DCT coefficients can be optimized to become more sparse. An easiest method is the normalization, which is to divide every coefficient by the block length N. However, result of experiments show that it does not help improve compression ratio. Because it does not reduce the diversity of the coefficients but just scales them. The effect of scaling will be offset by the quantization.

The shape of the DCT coefficients of each block looks very similar, which indicates the possibility of mean value removal. By subtracting the mean value from each DCT coefficient block, the range of values should shrunk. However, we also consider to remove the median value, because many coefficients are centered around the median value. These two types of value removal can be regarded as the bias removal. We did experiments on record 117, which is a relatively regular signal shown in figure 3.6.

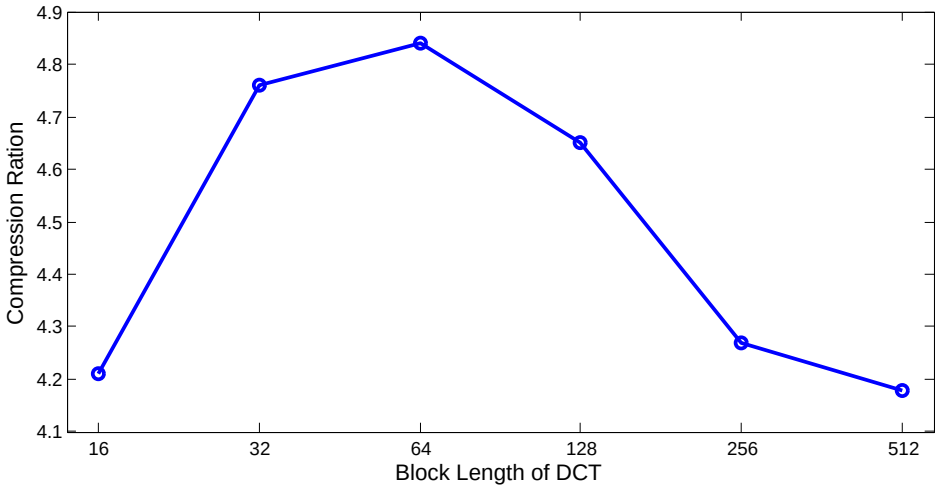


Figure 3.5: Comparison of Average Compression Ratio under Different DCT Length; Sample Rate = 360 hz

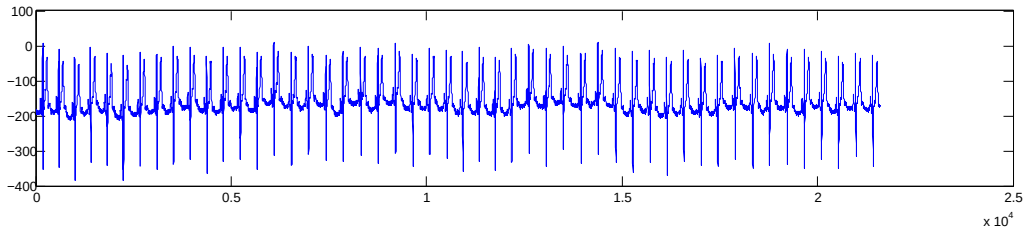


Figure 3.6: Record 117 of 1 Minute, Channel 1

We applied the DCT with and without bias removal and aligned the result as matrices. Results are illustrated in figure 3.7. It is clear that both bias removal methods shrink the range of coefficients. The original DCT coefficients has values as low as around -2000 while both bias removal methods range from only -1000. Nevertheless, it is hard to tell whether removing the mean value or the median value leads to more improvement. We did experiment further on comparison of the compression ratio. The result is shown in table 3.3.

We found the way of removing median value has the best compression ratio, even

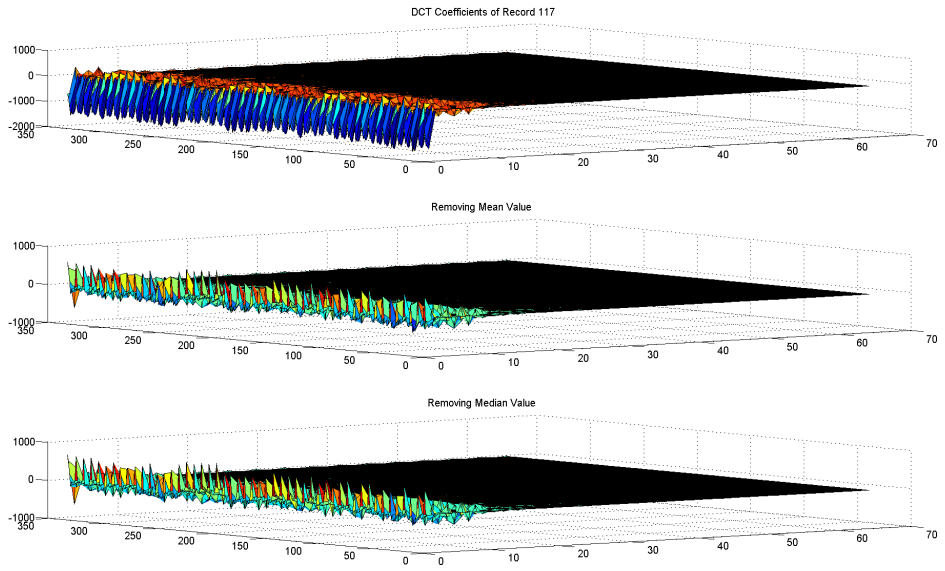


Figure 3.7: Matrices of DCT Coefficients before/after Bias Removal of Record 117; 1 minute; Channel 1

Table 3.3: Comparison of Compression Ratios of Different Methods to Optimize the Matrix of DCT Coefficients

Optimization Methods	Compression Ratio
None	4.84
Normalization	4.84
Remove Average Value	4.95
Remove Median Value	5.09

though the removing average value way has very close compression ratio. The reason is that, by removing median value, the quantized coefficients contain fewer different values and the range of values is smaller. This could reduce the encoding bits. These different values are usually defined as symbols in entropy encoding. We did check and profile the quantized result of the DCT coefficient matrix after bias removal. Table 3.4 shows the difference.

Table 3.4: Comparison of Symbols between Different Bias Removal Methods

Bias Removal	Number of Different Symbols	Range of Symbols
None	234	324
Remove Average Value	146	205
Remove Median Value	142	197

The range of symbol is calculated by:

$$range = max(symbol) - min(symbol) \tag{3.7}$$

Table 3.4 shows the subtle difference between removing average value and removing median value. They both contribute a significant reduction in power compared with original DCT coefficients. However, by removing median value the quantized symbols have lower diversity and smaller ranges. Encoder can thus build a smaller dictionary and use a little bit shorter bit-width to encode symbols. These are reasons for why removing median value can be best for compression ratio.

3.2.2 Quantization

DCT coefficients should be quantized to reduce the number of different symbols, but necessary information should still be preserved. Quantization is a technique that represents a range of values by a single quantum level. It removes trivial details from the signal so that it represents the signal by a limited number of bits. Quantization affects not only the reconstruction quality but also the compression ratio. Symmetric scalar quantization has proved successful for a variety of sources, especially for the Laplacian source [51]. The dead zone quantizer is a typical symmetric and scalar quantizer that is used widely in ECG applications. The strategy of this quantization method is that input values within the "dead-zone" are quantized to zero, whereas the remaining are uniformly and symmetrically quantized. There are different types of dead-zone quantizers [51] [53]. A simple but efficient version is used in [8] and [13] for ECG compression. It is defined as:

$$I_k = \begin{cases} (-3\delta, -T] & k = -1 \\ (-T, T) & k = 0 \\ [T, 3\delta) & k = 1 \\ [(2k-1)\delta, (2k+1)\delta) & otherwise \end{cases} \tag{3.8}$$

$$R_k = \begin{cases} 0 & k = 0 \\ \pm k\Delta & k = \pm 1, \pm 2, ... \end{cases} \tag{3.9}$$

where I is the input signal, namely the DCT coefficients in our design and k represents the quantization level, namely the output. Δ is the step size ($\delta = 0.5\Delta$) and T is the half size of the dead-zone. In principle, $\Delta \geq T$. Equation (3.8) shows the rule of quantization. The size of the dead-zone is thus 2T. Equation (3.9) shows the reconstruction from quantization coefficients, in which the reconstructed signal is obtained by multiplying Δ with the quantization level k. Figure 3.8 is the illustration of dead-zone quantization.

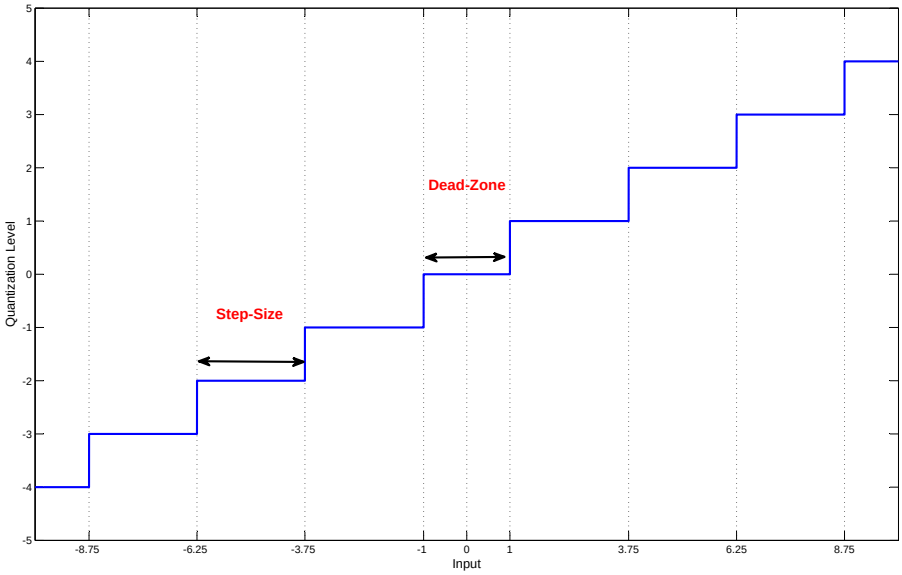


Figure 3.8: Dead-zone Quantization; T=1; *Delta*=2.5

The parameters T and Δ can affect the shape of the quantization curve, in turns the level of distortion. As discussed in section 3.1.2, quantization is usually the only lossy part in our compression scheme. The quantization parameter values are very significant for determining compression quality. In the next section we will discuss how we control the quality based on these parameters.

3.2.3 Quality Control Unit

In order to find suitable values for the parameter T and Δ, two types of search algorithm were used. The first type was the brute-force algorithm. Having defined bounds of Ts and Δs, it exhaustively searches for all possible value pairs. According to the sorted result it determines the optimal values for T and Δ. This method is simple but lacks consideration of performance. It is very time-consuming and proper bounds need to be defined manually. In practice, this method is more suitable in analysis stage, because it can illustrate all of the possibilities.
The second type of searching are mathematical optimization algorithms. The optimization problem is to find the maximum or minimum of a real function. In our case, the problem is to find the quantization parameters that minimizes the difference between our quality requirement and quality of the de-compressed signal. A number of optimization algorithms can be used to find the optimal value under the given constraints, for example, the Particle Swarm Optimization (PSO) algorithm [30], which solves the problem by simulating the process of particle movements in the search space. However, a high performance search algorithm means more complexity and resources. In our design we choose to use a binary search algorithm for its simplicity and efficiency.

Noticed by [12], there can be a linear relation between Δ and T as:

$$\Delta = \alpha T \tag{3.10}$$

Hence, the step-size Δ can be obtained by the dead-zone size T once the value of α is known. In order to search for an optimal value for α for the high compression ratio, we did tests on records from MIT-BIH database. The relation between the value of α and the compression ratio is shown in figure 3.9, when T and other parts and compression are fixed. From the figure 3.9 we observe that the compression ratio increases following the α and reaches the peak at around 1.3. In the range of 1.2 to 1.6 compression ratios are relatively higher than other values. By empirical method this value is further optimized to 1.32. How the quantizer is configured, to achieve maximal compression ratio, is highly related to the encoding approach. Because features of different encoders are diverse, for each encoding scheme the optimal value of α needs to be calculated again. This calculation can be regarded as an initialization/calibration process of our compression system.

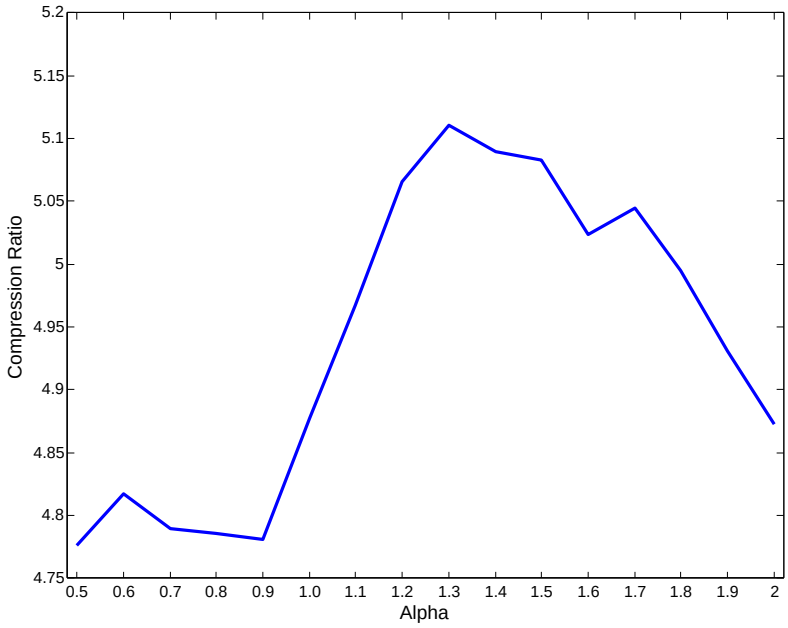


Figure 3.9: Average Compression Ratios of Different Alphas

Consequently, the problem of searching for T and Δ thus becomes a problem of searching for T given a constant value of α. A binary algorithm can be used, of which the scheme is shown in figure 3.10. The block in the yellow shows the quality control system. The quantizer sends the quantized result to the quality control system and it determines whether this result meets the quality requirement.

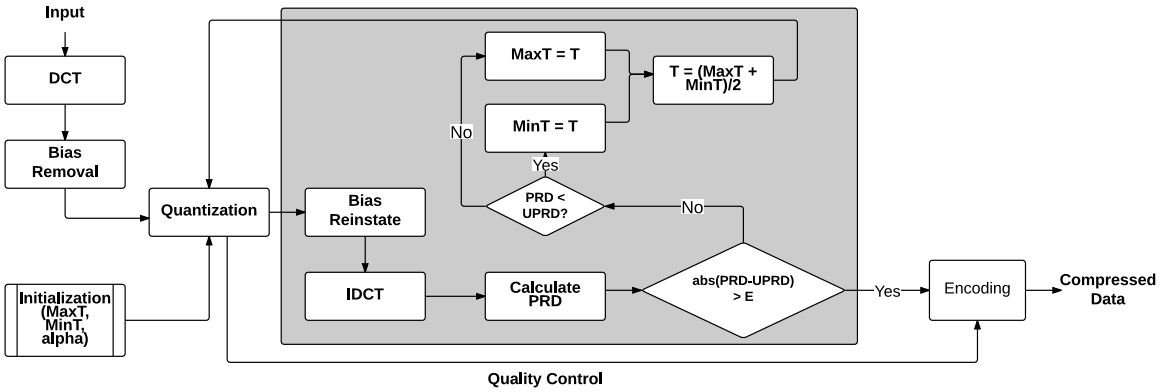


Figure 3.10: Quality Control Block with Quantization

The binary search algorithm searches for the solution value from the sorted array. In our case, the solution value should be a value of T. Since T is the half of the dead-zone size that quantizes the DCT coefficients, we initialize the dead-zone as the space between the upper-bound and the lower-bound of the DCT coefficients. The sorted array is consequently this space. T is initialized as the middle element. As indicated by "binary", searching starts from the element in the middle of the searching space, i.e., T. According to the value of T, the DCT coefficients is recovered from the quantized result and the input signal is reconstructed. If the quality of the reconstructed signal meets the criterion, the searching stops. Otherwise, this element becomes either the upper-bound or the lower-bound of the space that the searching restarts until the optimal value is obtained. In every iteration both T and the quantization result are updated. The criterion here is the quality measurement of ECG compression, namely the value of PRD/PRD2. In order to calculate PRD/PRD2, a reconstruction process is included in the quality control block to recover the signal from quantized result. This reconstruction process is also applicable to be used as part of de-compressor. Once the solution is found out, the corresponding parameter T is also saved for reconstruction. The searching speed is also acceptable. In average, 10 iterations is required to achieve the desired PRD1 of 1% for signal from MIT-BIH database. By this means, the result is guaranteed to have the desired quality.

3.2.4 Encoders

The data size is reduced by the encoder. To improve the encoding efficiency, we need to investigate the property of the encoder input. From above discussion, we know that

the input, the quantized DCT coefficients, can be regarded as a matrix. If we take the record 117 as an example, the input matrix can be plotted as the following spectrum:

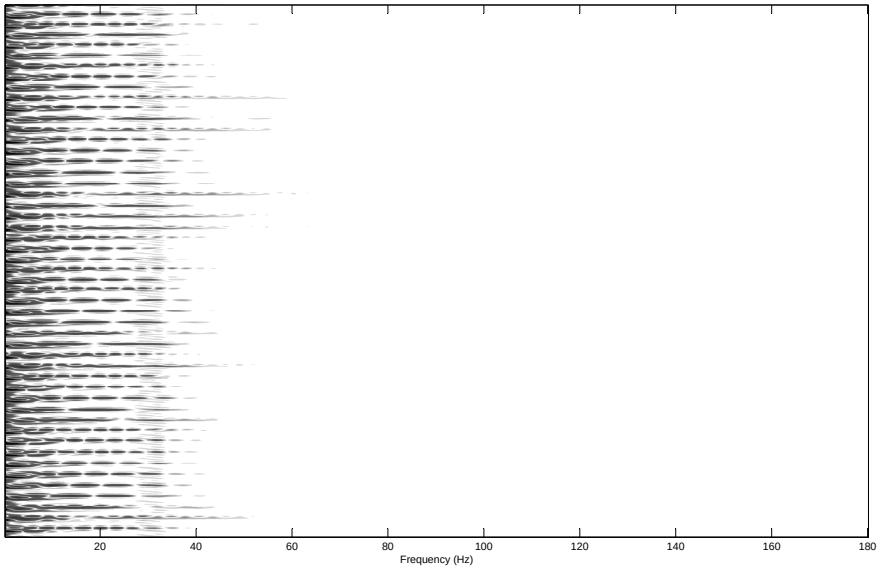


Figure 3.11: Spectrum of the Quantized DCT Coefficients Matrix after Bias Removal of Record 117

The length of DCT here is 64, which is the optimal value. Figure 3.11 is plotted by assigning different values with different color levels. Bright colors represent smaller values, while dark colors represent larger values. The width of the matrix is the DCT block length, which also represents the frequency range from 0 to Nyquist frequency, i.e., Fs/2. Figure 3.11 reflects the fact that, most of significant values are located in the left region, where low frequency components are. The closer to the left end, the darker the plot is, the more significant the values are. This plot also shows that less than the half of the whole frequency domain have significant values. In the right half, the color is far brighter than in the left, which means it is possible to use much fewer bits to represent the higher frequency part.

Based on this discussion, several encoding approaches are proposed to improve compression ratio. In the following sections we will discuss these encoding methods.

3.2.4.1 Run-length Encoding (RLE)

Run-length encoding [23] is a simple and direct way of encoding. The most basic run-length encoding is to detect the repeated part of a string and replace that with the number of repeated elements. This can reduce the size of strings of repeated symbols, especially when the input has very long repeated segments. For example, assume a random input string which is:

"jjjkllkjj"

If we encode this string directly by 8-bit ASICII encoding, the size will be 72 bits. When applying run-length encoding, the string will be as:

"j3kl2kj2"

which we can encode separately the letter and the digit. Thus 5 letters are encoded by 8 bits and 3 digits are encoded by 3 bits that in total 49 bits are used. This presents a compression ratio of 1.47.

We may further optimize that by the following string:

"j2kl1kj1"

The string "j2" can be interpreted as a j is followed by 2 js. By this means digits here can only be encoded by 2 bits, which saves 1 bit for each digit.

However, performance of the run-length encoding is affected highly by the input sequence. Especially when the input rarely contains string of repeated symbols, run-length encoding will not benefit anymore. Run-length encoding is possible to reduce storage space, but cannot guarantee its performance. For the purpose of efficient and robust encoding, we need to consider more complex encoding methods.

3.2.4.2 Huffman Encoding

Huffman code [27] is a very famous entropy encoding that is widely used for signal processing and telecommunication. It achieves the optimal encoding efficiency from a frequency-sorted binary tree, that is, Huffman tree. Given a string containing N symbols S_i and the weight W_i (usually frequency) of each symbol, the Huffman tree has the minimal weighted path length. The weighted path length is defined as:

$$WPL = \sum_{i=1}^{N} W_i \times L_i; \tag{3.11}$$

where L_i is the length of the path from root to the node S_i. A typical Huffman coding algorithm is shown in algorithm 1.

Data: String *String*
Result: Codeword C_i
Step 1: Calculate weight set Ws as frequencies of the symbol set Ss in *String*. Initialize an empty tree T.
Step 2: Find the two minimal W_is and their indexes, for example index x and y. Find their corresponding S_x and S_y.
Step 3: Make S_x and S_y as children of a binary tree in sorted order. Add a node of the weight W_x+W_y as their parents. Add this node to W and add this tree to T in sorted order.
Step 4: Delete W_x and W_y from W. Keep the tree T and go to step 2. Repeat until only one element left in W.
Step 5: The codeword C_i for each S_i is the concatenation of weights of the path from the root to the corresponding leave node.

Algorithm 1: Huffman Coding Algorithm

An illustration to the Huffman encoding is that: assume there is a string:

AAABBBCCD

We can have the symbol list and the corresponding weight list as:

Symbol: A B C D
Weight: 3 3 2 1

We notice that C and D have the minimal weight so that we build the binary tree as: The value near the line is the weight of the path. It always follow the rule that the

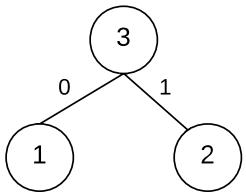


Figure 3.12: Huffman Tree Example 1

weight of left path is 0, otherwise 1. Values are assigned to the binary tree. We then delete symbols C and D and their weights from our set and add the node of weight 3. Then the weight list becomes:

Symbol: A B C+D
Weight: 3 3 3

We find that all the weights are equal. Considering we already have a node of weight 3 in the tree, we only add one node in the tree, which can be A. So that the tree becomes: Then we only have 1 symbol left. So we can update and finalize the tree as: We do step 5 from 1 that we have the codeword book as: Huffman encoding is a

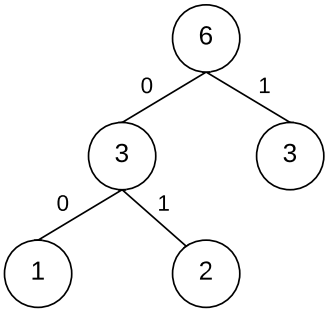


Figure 3.13: Huffman Tree Example 2

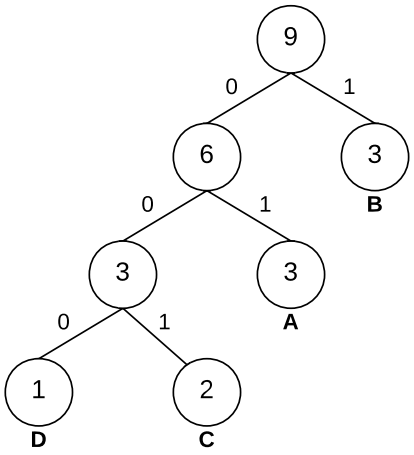


Figure 3.14: Huffman Tree Example

variable length encoding method, as shown in table 3.5. The weighted path length is 18 bits in the table. In average each symbol needs 2 bits for encoding. If we encode this by fixed-length encoding, each symbol needs 3 bits, which is clearly worse than Huffman encoding. Huffman encoding's efficiency is proved approximately optimal by Shannon's source coding theorem [48].

Table 3.5: Example Huffman Encoding Codebook

Symbol	Codeword	Length	Weight
A	01	2	3
B	1	1	3
C	001	3	2
D	000	3	1

3.2.4.3 Arithmetic Encoding

Arithmetic encoding [58] is another widely-used entropy encoding method. It also has approximate optimal coding efficiency but is possible to perform better than Huffman coding. Different from Huffman encoding, it does not make the codeword book for each symbol. Instead, it encodes the whole message into a number. Similarly, arithmetic encoding uses probabilities and symbol sets as used by Huffman encoding. However, the encoding approach is very different. A typical Arithmetic encoding algorithm is shown as algorithm 2.

Data: String *String*
Result: Encoding Result E
Step 1: Calculate and sort the probability set P of the symbols set S from the input *String*. Initialize the analysis interval as [0,1].
Step 2: According to the probability set P, divide the current interval into several small intervals. Each small interval has the length of the corresponding symbol probability.
Step 3: Check the interval for the first element from *String*. Make that interval as the current analysis interval.
Step 4: Divide and scale the current interval as step 2.
Step 5: Check the interval for the next element from *String* and make that interval as the current analysis interval.
Step 6: Repeat step 4-5 until we reach the end of the string.
Step 7: Any fraction in between the final interval can be our encoding result E.

Algorithm 2: Arithmetic Coding Algorithm

Take the same example input string as in last section, we have the probability table 3.6.

Table 3.6: Probability Table for Arithmetic Encoding

Symbol	A	B	C	D
Probability	0.333	0.333	0.222	0.111

As the first step of algorithm 2, we initialize our analysis interval as shown in figure 3.15. In the figure 3.15, each symbol occupies an interval that has the length of corresponding probability. The whole interval [0,1] is obviously the probability range for the entire input message. The first element of the input string "AAABBBCCD" is "A". We then set the current analysis interval for symbol "A" and scale this interval according to the probability set. Thus we have the figure 3.16, where the scaled interval

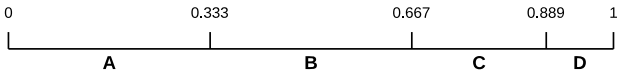


Figure 3.15: Arithmetic Encoding Interval 1

is shown in the lower plot.

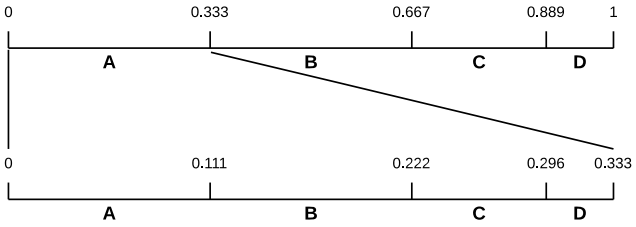


Figure 3.16: Arithmetic Encoding Interval 2

We repeat this process until we meet the last symbol, as illustrated in figure 3.17. It shows that the final interval is [0.019011, 0.019018]. Any values within this range can be our encoding result.

An issue exists in the above process. The longer the input string is, the smaller the probability could be. Since the arithmetic encoding is an accumulative process, the final interval can be extremely small values when the input is long enough. These values sometimes cannot be represented, because modern machines can usually support at maximal 64-bit floating point number, where precision is limited. Even if they can be represented, computations like multiplication and accumulation of such long numbers may not have very precise result. This will prevent the lossless decoding.

To overcome this defect, modern arithmetic encoding methods are adaptive. Among them a recursive splitting method is stable and efficient. The idea of recursive splitting is proposed by [50]. It splits the input sequence to smaller sub-arrays, so as to reduce the number of symbols, as well as the length of the encoded array. Consequently, a favorable coding efficiency can be achieved and it prevents the problem of precision, as discussed above.

There are two types of recursive splitting. Type 1 is splitting by previous symbols.

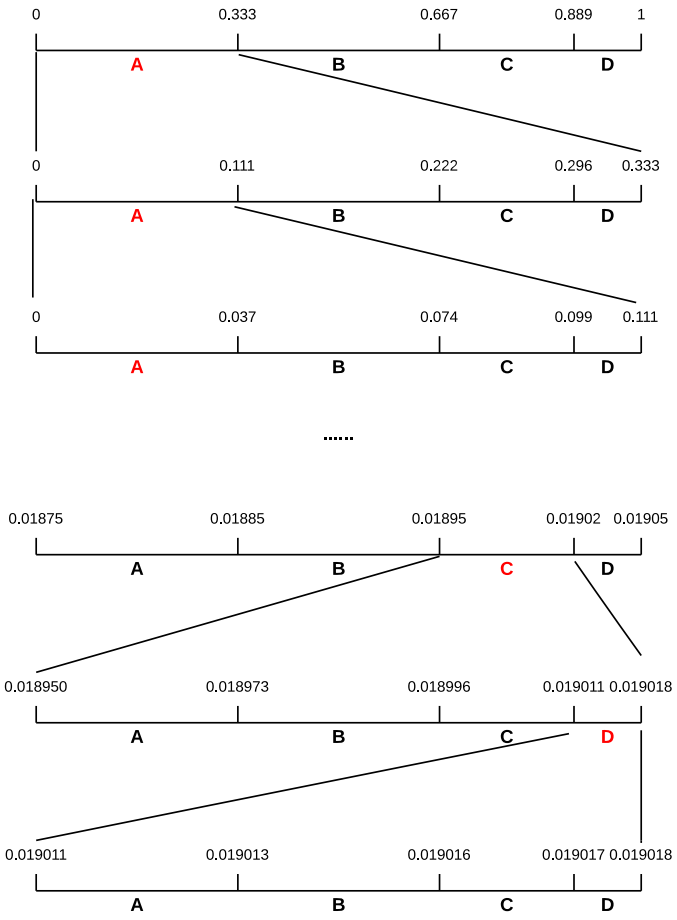


Figure 3.17: Arithmetic Encoding Interval Calculation

This method utilizes the correlation between successive symbols. By using the median value of previous symbols as the limit value of the following symbol, we are able to split a sequence to two equal-size subsequences. Type 2 is splitting in the middle of the sequence, which is rather simple but still useful. The criterion here is the length of the sequence. The benefit of splitting is obvious that we can always have smaller subsequences, so that values of the probability table of each subsequence is within a desired range.

However, the algorithm code posted by the author of [50] is very complicated for containing many conditions and branches. This improves the encoding efficiency but increases the computing complexity significantly. Besides, some branches are barely used in our application. As one block of our ECG data compressor, the encoder should not dominate the resource use. The design should make balance between the performance

and complexity. Therefore, we have modified the original method to a light-weight version, while good efficiency is maintained. Its scheme is shown in figure 3.18.

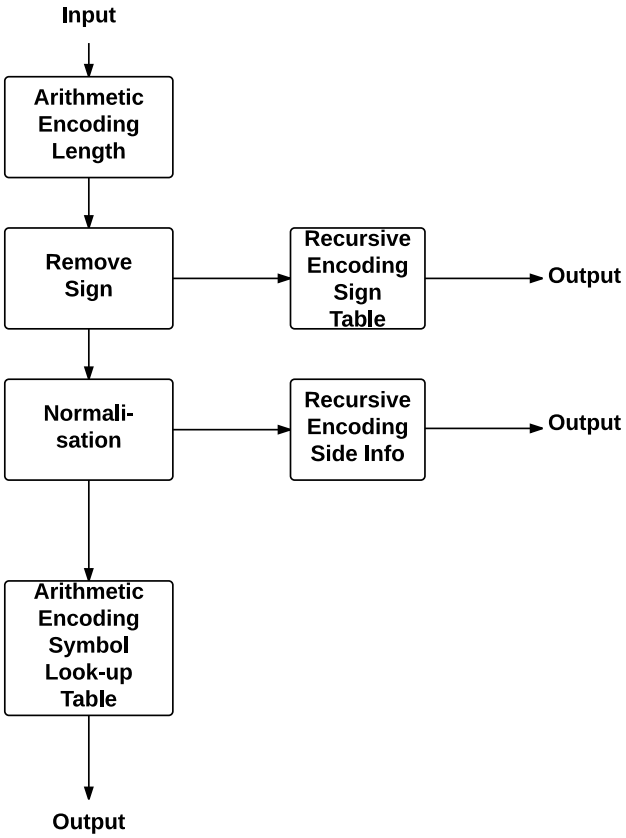


Figure 3.18: Block Diagram of Modified Recursive Arithmetic Encoding

As shown in figure 3.18, our modified encoding algorithm is made up of different blocks. The core part here is the recursive encoding (RE), which exactly does recursive splitting and arithmetic encoding. The algorithm does not direct encode the input sequence by RE, instead it starts with redundancy removal by generating side information. First it encodes the length of the input sequence by the typical arithmetic encoding. Then it removes signs of the input sequence and maps the signs to a look-up table. This table is later encoded by the RE. Next, by determining whether the value is larger than the threshold TH or not, very large non-zero entries are eliminated. The elimination follows the equations below:

$$xn = \lfloor log2(x) \rfloor \tag{3.12}$$

$$xa = x - 2^{\lfloor log2(x) \rfloor} \tag{3.13}$$

xn is used to represent the x in the next stages and xa is the side information to reconstruct x from the encoded result. This side information along with the non-zero entry look-up table is encoded by RE as well. Now the input sequence becomes an unsigned sequence with small values of symbols. Two identical look-up tables of symbols are generated. Based on those, symbols are encoded by directly using arithmetic encoder.

The recursive encoding process from figure 3.18 is shown in figure 3.19 and figure 3.20. The first step is to compute a differential sequence x3 from the input x. Since the input sequence contains only 0s and 1s, the number of bits needed can be easily derived by the number of 1s. Number of bits are stored as b0 and b2, respectively. Besides, we store position vectors of non-zero entries as I and I3 and their lengths as L11 and L31. We also store the position vectors of zero entries as J and J3. Afterwards we follow the following rules in the splitting step:

1. Determine if the length of the input x, L, is larger than T1. If not, calculate the value of b1 as the equation (3.14).

$$b1 = b0 + 1 \tag{3.14}$$

$$b3 = b2 + 1 \tag{3.15}$$

   Go to step 5. Otherwise, go to step 2.

2. Determine if any subsequence larger than T2 after splitting the input x by L11. If not, calculate the value of b3 as the equation (3.15), go to step 5. Otherwise go to step 3. Apply the same rule to x3.

3. Determine if L11 is smaller than L/2. If so, split the input x by:

$$x1 = x(I + 1) \tag{3.16}$$

$$x2 = [x(1); x(J + 1)] \tag{3.17}$$

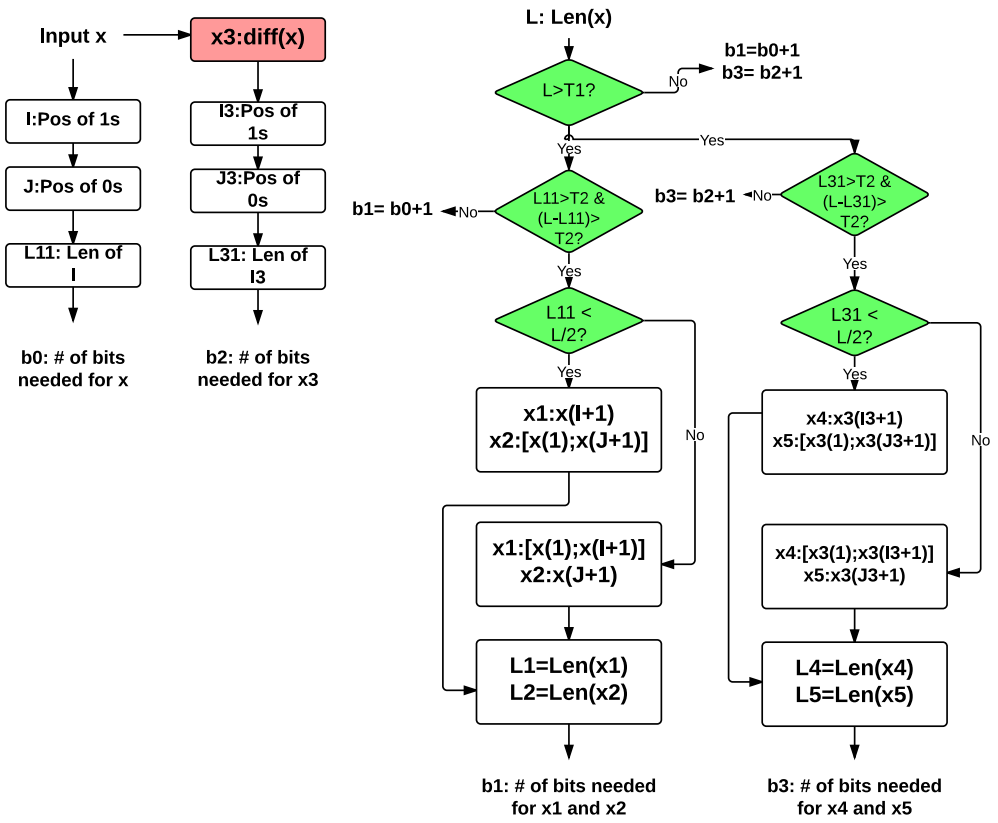


Figure 3.19: Recursive Encoding: Splitting

Otherwise, split the input x by:

$$x1 = [x(1); x(I+1)] \tag{3.18}$$

$$x2 = x(J+1) \tag{3.19}$$

Store lengths of x1 and x2 as L1 and L2, respectively. Apply the same rule to x3. Go to step 4.

4. Calculate and store the number of bits needed for x1 and x2 as b1. Calculate and store the number of bits needed for x4 and x5 as b3. Go to step 5.

5. Determine which method to use based on obtained b0, b1, b2, b3, L11, L31, L, x, x1-x5, as illustrated by figure 3.20.

Figure 3.20 shows how the recursive calls are managed. We always encode the array that has the smallest number of bits. We split the over-sized subsequences to make

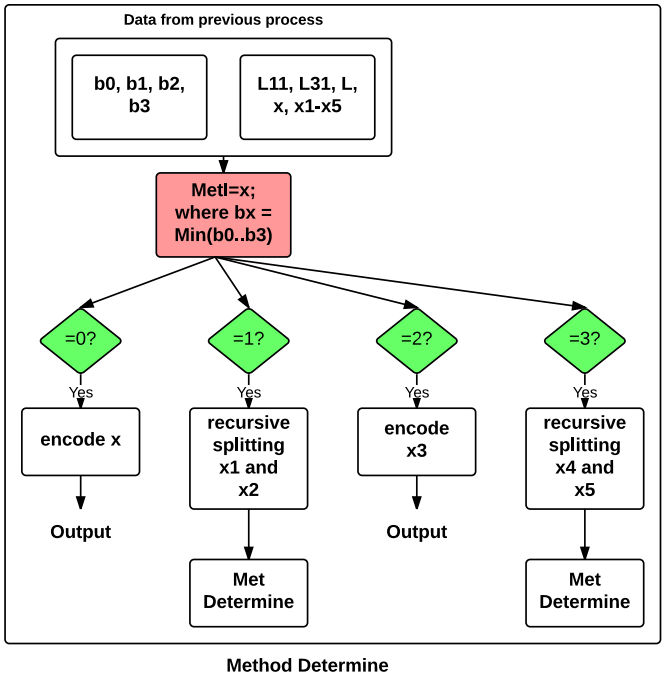


Figure 3.20: Recursive Encoding: Recursive Calling

them compliant with the arithmetic encoder. In this way the encoder is capable of lossless encoding regardless of the length of the input sequence, while maintains a high coding efficiency. This method is a great improvement to the conventional arithmetic encoding.

Up to now we have three types of encoding methods. Considering that the run-length encoding is not very efficient but relatively simple to implement, we introduce 2 combinations based on it: Huffman encoding with RLE and recursive arithmetic encoding with RLE. We tested these encoding methods on the Calgary corpus [6], a commonly used file collection for testing data compression algorithms. We did convert the binary files from ASCII representation to integers. Table 3.7 shows the comparison result in terms of compression ratio.

We observe that recursive arithmetic encoding has the best performance against any other methods/combinations. As expected, run-length encoding has very limited performance and it is possible that the compression result occupies larger storage space than original representation. We also notice that even combined with Huffman encoding and recursive arithmetic encoding, RLE does not have any better result. Comparing Huffman encoding and the recursive arithmetic encoding, in average recursive arithmetic encoding performs a bit better than Huffman encoding. For the file "pic" the recursive arithmetic coding outperforms significantly. This is because the input file is

Table 3.7: Compression Ratios of Different Encoding Methods on Calgary Corpus

File Name	Huffman Encoding	Recursive Arithmetic Encoding	RLE	Huffman with RLE	Recursive Arithmetic with RLE
trans	1.43	1.45	0.87	1.3	1.48
progp	1.62	1.64	0.94	1.46	1.66
progl	1.66	1.68	0.94	1.54	1.73
progc	1.52	1.54	0.87	1.35	1.53
pic	4.8	6.79	2.99	6.21	6.48
paper2	1.71	1.74	0.82	1.44	1.72
paper1	1.59	1.60	0.82	1.35	1.59
obj2	1.27	1.3	0.77	1.12	1.29
obj1	1.29	1.33	0.9	1.21	1.34
news	1.53	1.54	0.85	1.35	1.56
geo	1.4	1.41	0.76	1.21	1.39
book2	1.66	1.68	0.82	1.4	1.66
book1	1.75	1.77	0.82	1.47	1.75
bib	1.52	1.54	1.02	1.3	1.53
Average	1.77	1.93	1.01	1.7	1.91

a binary image, which contains a lot of regular sequences and 0s, of which the features benefit from recursive splitting. In our case, ECG signals are normally approximate periodic signals, and the quantized DCT coefficient matrix is sparse. Therefore, we expect a good compression ratio by using the recursive arithmetic encoding.

3.2.4.4 2-stage Encoding

So far the discussion is made with regarding to the general purpose techniques. As mentioned in the previous sections, the input of the encoder, the quantized DCT coefficient matrix, has some special features. As shown in figure 3.11, elements in the right region are most probably to be quantized as zeros. The darker the color is, the larger the values are. If we take the record 117 as an example, values of the quantized DCT coefficient matrix for a 1 minute segment is as:

$$\begin{pmatrix} 3 & -1 & 2 & -9 & \cdots & -1 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ -39 & 6 & 61 & 72 & \cdots & -3 & 0 & 0 & -1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 58 & -25 & 40 & -17 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 36 & -107 & 19 & 84 & \cdots & -4 & -4 & 3 & 0 & -3 & \cdots & 0 & -2 & 0 & 2 \end{pmatrix}$$

From the above matrix we see that there are many zeros in the right region. The region in the middle is relatively sparser than the left region. To make use of this property, Bendifallah[8] introduced a very efficient method, which is to re-arrange the matrix into three vectors. Those vectors are:

1. Non-Zero Element Vector: by including all the non-zero elements.

2. Last Non-Zero Position Vector: by including all last non-zero positions of each block.

3. Look-up Table Vector: By mapping zero/non-zero elements before the last non-zero positions to a binary table.

Therefore, in our above example, we will have these vectors:

- Non-Zero Element Vector: 3 -1 2 -9 ... -1 1 -39 6 61 72 ...

- Last Non-Zero Position Vector: 23 25 ... ... 19 64

- Look-up Table Vector: 1 1 1 1 ... 1 1 1 1 1 1 ... 1 0 0 1 1 ...

We only store information before the last non-zero element of each block. This reduces data size significantly. For instance, in the above example, the average last non-zero position is around 26, so that at least 38 elements per block (the block length is 64) can be saved by this method in average. Surely we have to encode side information for reconstruction, namely the last non-zero position vector (LNPV) and look-up table vector (LTV). The length of LNPV is the number of blocks from the input segment, of which the value is usually short. LTV is a very simple and regular vector, which is easy to be encoded into small size. Above all, the compression ratio should be improved by this type of data processing.

Nevertheless, the above method is just the first step of efficient encoding. To have a good encoding result, aforementioned entropy encoding methods are needed. A 2-stage encoder is thus presented, which combines the above mentioned data re-arrangement and an entropy encoding method. We compare DCT compressors with these different encoding methods on the MIT-BIH database. Results are shown in the table 3.8. It verifies our expectation of the 2-stage encoding method, where the compression ratio is improved by around 25%. In addition, the comparison is compatible with results from table 3.6. It shows the recursive arithmetic encoding also has great performance on the ECG signal compression.

Table 3.8: Compression Ratios of Different Encoding Methods on MIT-BIH Database

Methods	Compression Ratio
Huffman Encoding	3.63
Recursive Arithmetic Encoding	4.36
Run-length Encoding	1.91
Huffman with Run-length	3.61
Recursive Arithmetic with Run-length	4.03
2-stage Recursive Arithmetic	5.11
2-stage Huffman	4.57

3.3 Modifications and Optimizations

The ECG signal is made up of cardiac cycles, namely heartbeats. The cardiac activity of a normal person is usually regular and steady. In most cases cardiac cycles are similar to each other, which indicates that coding redundancy must exist. This is a property that may be utilized to improve the compression performance significantly.

To detect cycles, i.e., beats, of an ECG signal, there are many methods. In our algorithm, we prefer to detect R peaks. As shown in figure 1.1, R peak is the highest value in a cycle. We find it is possible to search for all R peaks by extracting local maxima. Thus a beat can be defined as the region in between two R peaks, instead of a complete PQRST complex. Thus by detecting R peaks, beats are easily found and tagged. For example, R peaks are detected as shown in figure 3.21.

To exploit the redundancy, a direct way of manipulating detected beats is the alignment. We can align the above beats as a matrix, as shown in figure 3.22. However, the length of each beat is not constant. Interpolation is needed to stretch or shrink sizes of each beat to the same. In latter sections, we will see benefits from this process.

Consequently, we propose an enhanced DCT-based ECG compression algorithm with beat detection and alignment. Figure 3.23 is the block diagram of the modified compression system. The part changed is the input of DCT and the reconstruction process as well. Furthermore, we add the filter option to the compressor to enhance its feature. Oversampling can thus be achieved as well. In following sections, we will discuss details of these modifications and optimizations.

3.3.1 Beat Detection

Using the correlation between ECG cycles is not a new topic in ECG compression. As mentioned above, most algorithms use R peak detection to find cycles. In [55] a QRS detection method based on Dyadic wavelet transform was proposed. In [14] a template matching algorithm is exploited, where R peak extraction is accomplished by differentiating the input signal. R peak detection is also used in [4]. This algorithm utilizes the power of the difference between the cycle and the template, so as to improve

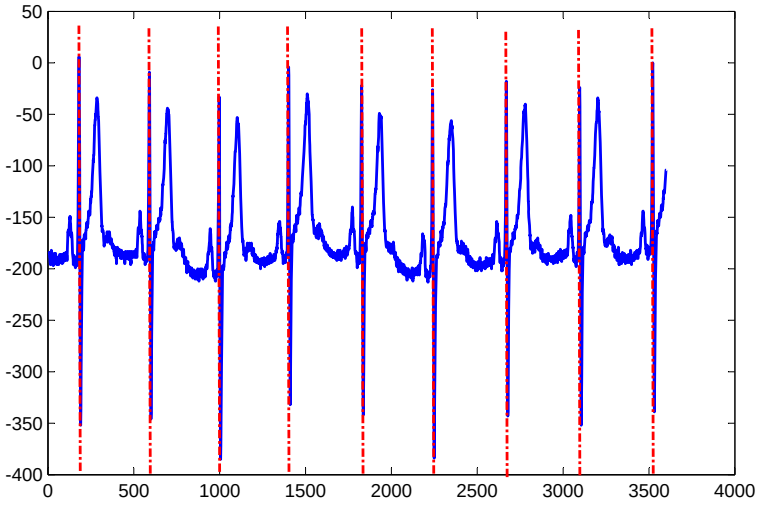


Figure 3.21: R Peaks in Record 117

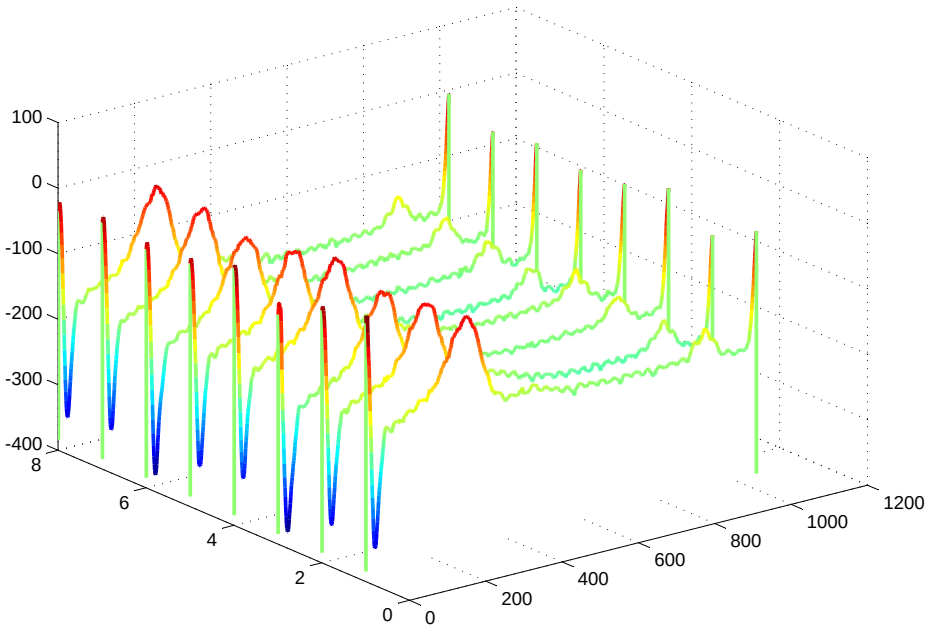


Figure 3.22: Desired Beat Alignment of Record 117

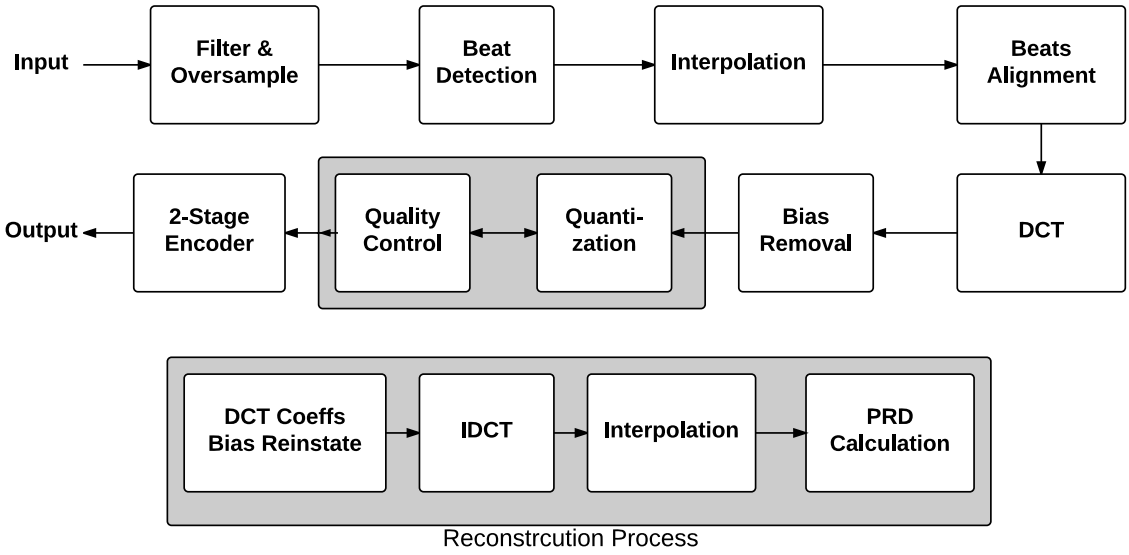


Figure 3.23: Block Diagram of the Modified Compression Algorithm

the compression ratio. In our design, we use the R peak detection as well, but we also do experiments on finer oversampling and interpolation, which makes our result superior.

The Wavelet transform modulus maxima method [37] is widely used for detecting singularity and fractal dimension of a signal. It utilizes the continuous wavelet transform (CWT) [25] to decompose a signal as a function of time. Compared to FFT, CWT not only gives information in the frequency domain but also in the time-domain. It transforms a signal into a collection of scaled wavelets. A mother wavelet $\psi(t)$ should be continuously differentiable and its Fourier transform should satisfy that:

$$\int_0^{+\infty} \frac{|\psi(\omega)|^2}{\omega} d\omega = \int_{-\infty}^{0} \frac{|\psi(\omega)|^2}{\omega} d\omega = C_\psi < +\infty$$

A wavelet transform of a function $f(x)$ is thus:

$$Wf(s,x) = f * \psi_s(x)$$

where $\psi_s = (1/s)\psi(x/s)$ and s is the scale factor.

A feature of the CWT is that the result is highly related to the mother wavelet and scaled function and/or factor. This makes it tunable to match different requirements of the application. In our R peak detection method, the difficulty is how to distinguish the R peak and T peak if their amplitude is very close. We notice that by using a Mexican hat wavelet, the R peak, which is in the middle of the PQRST complex, extends much

more significantly than the T peak. In this way peak detection is enhanced through the now large difference between R peak and T peak amplitudes. The Mexican hat wavelet has the shape as shown in figure 3.24.

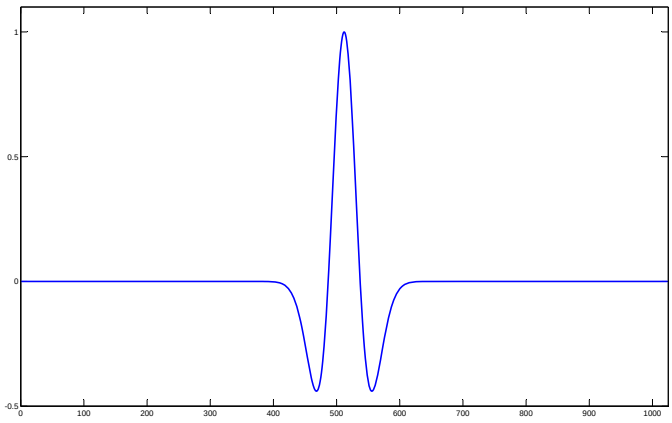


Figure 3.24: Mexican Hat Wavelet

If we take the record 228 as an example, the original signal and the wavelet transformed signal are plotted in figures 3.25 and 3.26. They show clearly that R peaks are much more distinct in the transformed signal rather than the original signal. The wavelet transform can also be interpreted as a way of filtering. Figure 3.26 shows that the baseline of the transformed signal has been removed and the transformed signal approximates a concatenation of scaled Mexican wavelets.

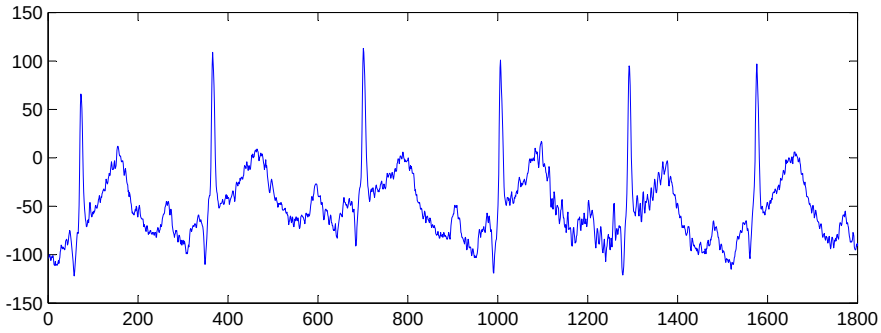


Figure 3.25: Record 228

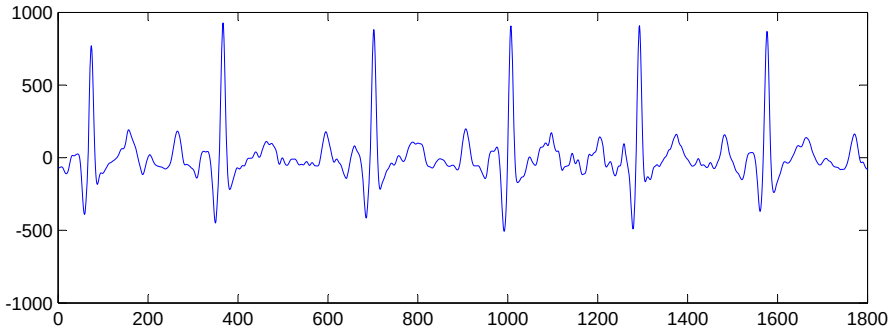


Figure 3.26: Record 228 after Continuous Wavelet Transform

Wavelet transform is the initial step of the signal processing but does not work as detection itself. We accomplish R peak detection in the following steps.

1. Edge detection: differentiate the transformed signal x and find zero-crossing points. The result is shown as figure 3.27.

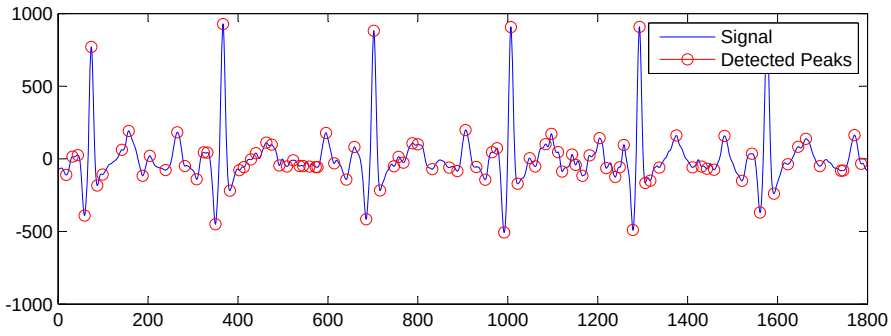


Figure 3.27: Detected Points after Edge Detection

2. Adaptive Thresholding: Initialize the threshold T to half of the maximal amplitude of the first 20 detected peaks. Heart beat rate is usually within the range of 27 and 220 beat per minute (bpm), which means there are at most 4 beats per second. Thus we initialize the window size to $fs/4$ and find the local maxima from detected peaks within the window. The value of threshold, T, is changed adaptively according to a shifting window. The following rules are applied, of which the method is derived by observing the behavior of the figure 3.29:

   - If the local maxima $lmax$ is smaller than $2T$, then T is updated as:

$$T = (1.2 * lmax + T) * \omega \tag{3.22}$$

- Else, T is updated as:

$$T = (0.15 * lmax + T) * (1 - \omega) \tag{3.23}$$

ω is a parameter of the range $0 \leq \omega \leq 0.5$. The value of 1.2 and 0.15 are arbitrary and can also be regarded as parameters. This adaptive method can avoid mis-detection when the amplitude of R peak varies too much from one part to another. The detected result from this stage is as follows:

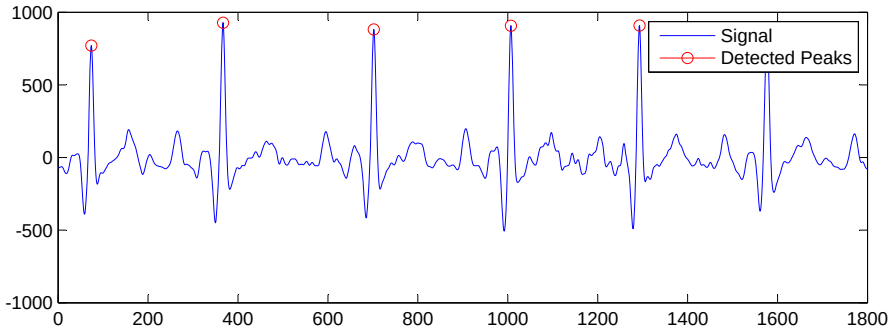


Figure 3.28: Detected Points after Adaptive Thresholding

- Elimination of False Peaks: Calculate the average peak-to-peak distance (APD) and compare each peak to find pairs that are close. Windowing the region of first close pair by APD. Remove all of the peaks that is not the maximal value within the window. Shift the window to the next pair until the end.

  As figure 3.29 shows, most peaks are detected correctly but we still have some "crowded" peaks in the right part of the upper plot in figure 3.29. We collect all of them as very close peaks and apply the elimination method, the final result is shown in the lower plot.

We have covered the basic steps of our R peak detection algorithm and the result of the example (Record 228) is shown in figure 3.30. Almost all R peaks are correctly detected and only a very few errors exist. These errors are acceptable whilst maintaining our compression quality. Compared to other R peak detection

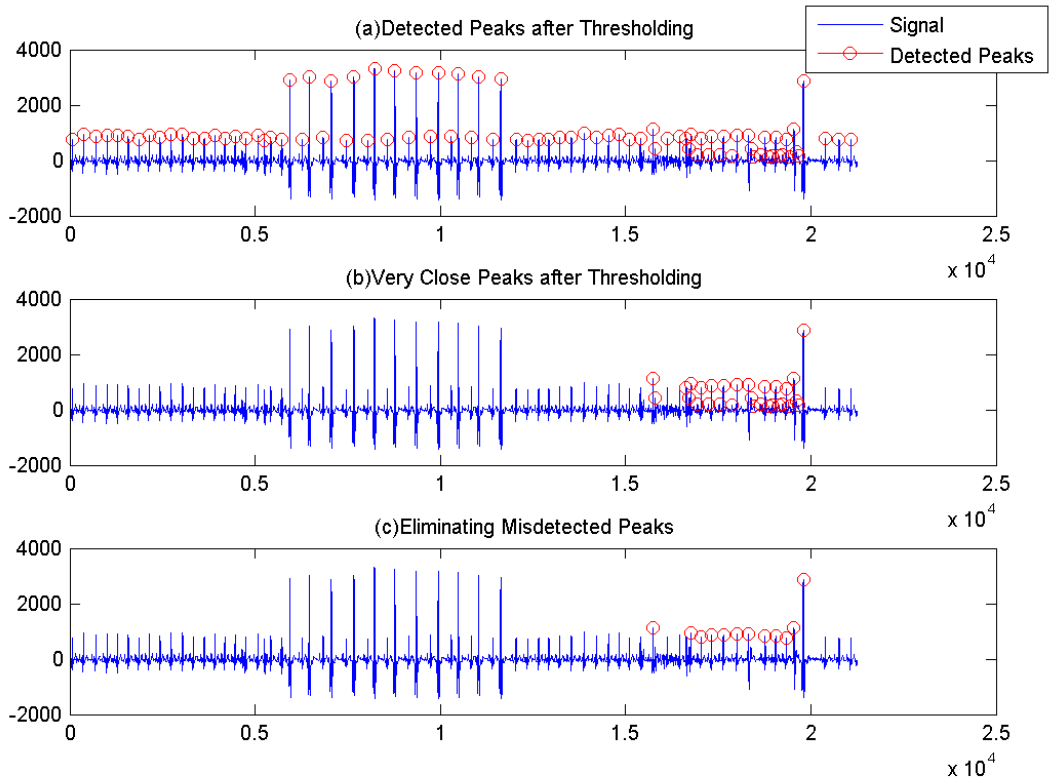


Figure 3.29: Detected Points after Elimination of False Peaks

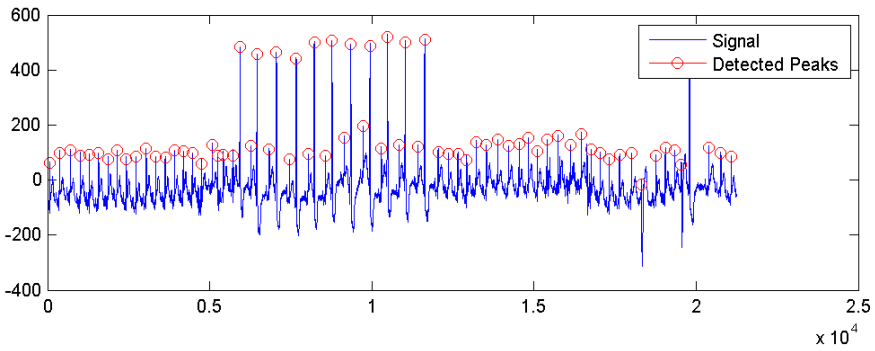


Figure 3.30: Result of R peak Detection for Record 228

algorithms, we also consider the polarity of the signal. For example, the R peaks in figure 3.31 are on the negative rather than on the positive. Using a simple positive threshold will lose this information. However, using the absolute values of the signal will enable the use of just a positive threshold and will increase

the errors in detection. In this case, the R and S parts of the PQRST complex are problematic when they have similar magnitudes. In our solution, we use the R peak detection twice, once on the positive side and once on the negative side. Then we determine which side probably contains the correct R peaks by comparing the number of peaks and their mean values on each side. Hence, we can avoid mis-detection of negative signals.

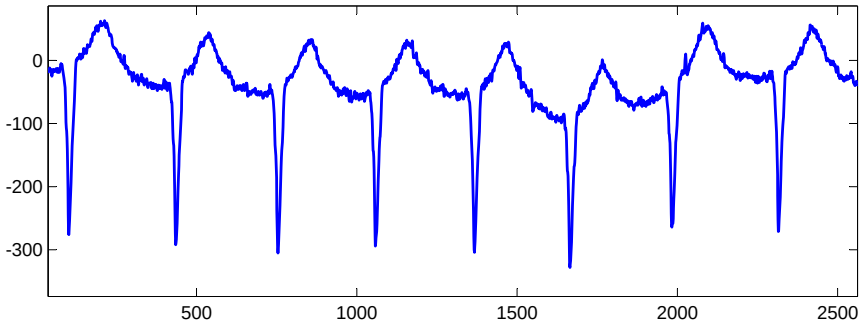


Figure 3.31: Record 207

3.3.2 Beat Alignment

We align each beat by interpolating the signal containing the beat to the same length. To avoid information loss, we define the interpolated length as:

$$L = 2^{ceil(log2(max(beat\_length)))} \tag{3.24}$$

L is at least the same length as the longest beat. So that a beat is always stretched. Moreover, the interpolated length is a power of 2, which is easier for DCT implementation.

Linear interpolation is the simplest method, but we choose quadratic interpolation. It approximates the signal as a piecewise quadratic function:

$$y = (a * x + b) * x + c \tag{3.25}$$

The solution to a, b and c can be easily calculated by using 3 successive points.

The benefit of quadratic interpolation is that it is closer to the underlying continuous signal. Figure 3.32 shows an example of the linear interpolation result and the quadratic interpolation result of a same piece of signal. The plot of quadratic interpolation is clearly smoother. To show their effects on our ECG compression algorithm, we did tests on our chosen dataset from MIT-BIH database. Table 3.9 shows the difference between reconstruction quality for linear and quadratic interpolation. We used PRD1 and PRD2 to measure the quality as usual, the result shows that quadratic interpolation causes far less quality loss than linear interpolation.

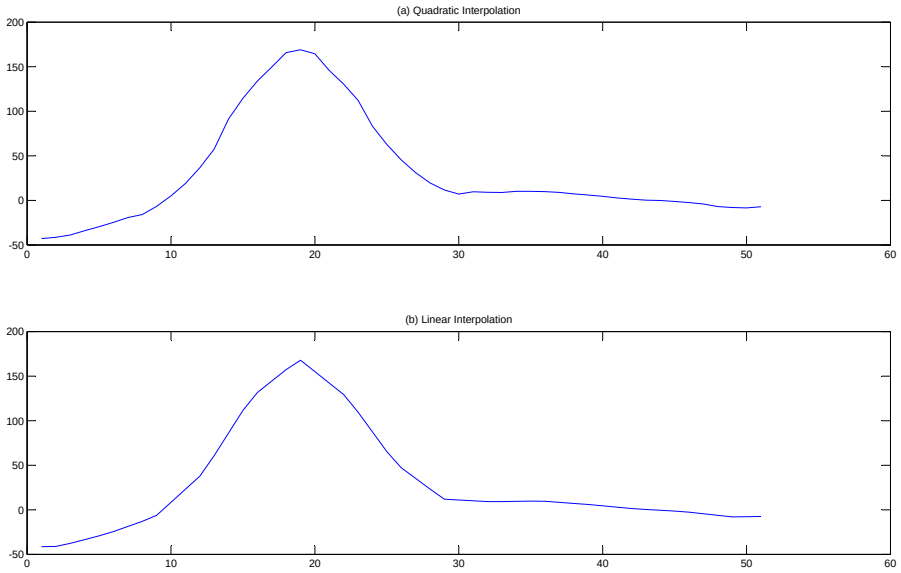


Figure 3.32: Example of the Linear Interpolation Result and the Quadratic Interpolation Result

Table 3.9: PRDs of Linear and Quadratic Interpolation

Type	PRD1	PRD2
Linear	0.2329%	0.3323%
Quadratic	0.0301%	0.0429%

By the above beat detection and alignment (interpolation), we show a matrix comparison in figure 3.33 that the input of the DCT is much more compliant to achieve a sparse output. We show the DCT coefficients matrix after bias removal in figure 3.34. Compared with figure 3.7, the range of values is decreased from around (-1000,1000) to (-500,500).

We compared the sparseness between figure 3.7 and 3.34 by using binary thresholding. The threshold value ranges from the mean value of the DCT coefficients without the beat alignment (figure 3.7) to the mean value of the DCT coefficients with the beat alignment (figure 3.34). Figure 3.35 shows the sparseness by using percentage of zeros after thresholding, where the higher the percentage is, the sparser the matrix is. It is obvious that by using the beat alignment, the sparseness of the DCT coefficient matrix is increased.

3.3.3 Filter and Oversampling

An ECG signal does not only contain information about heart activity, but contains contributions from many other sources. Some of the components are often

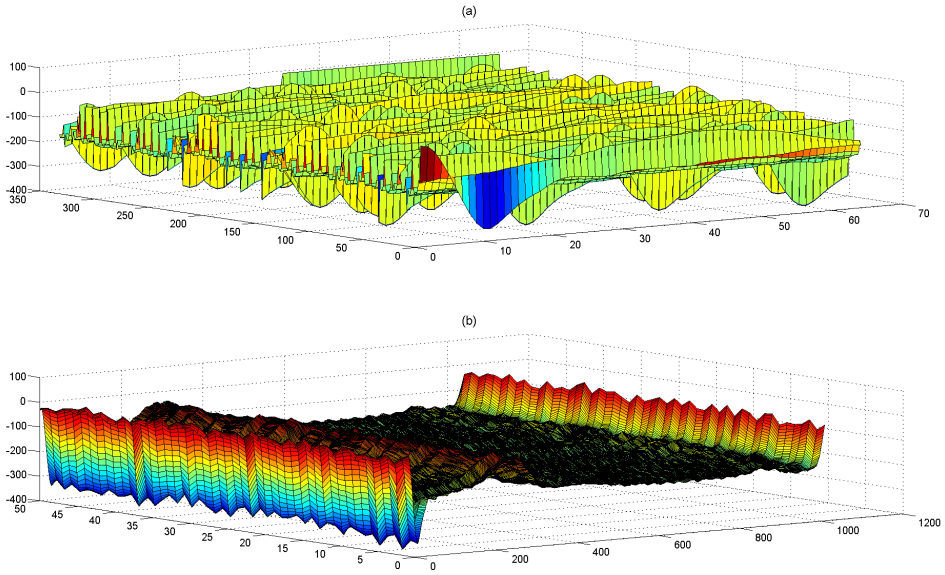


Figure 3.33: Matrix Comparison of the Record 117; channel 1; segment length = 1 min. (a)use direct DCT and the block length is 64. (b)use the beat detection and alignment

called as artifacts/noise in electrocardiography. They can be electrical interference, muscle noise and motion interference. Many types of noise can be found, but what stands out is the alternating current (AC) interference. In Europe, 50 Hz AC electricity is used as standard and will interfere with the recorded ECG signal all of the time. Removing it can be accomplished by using a band-pass filter. Filtering can affect the morphology of the ECG signal, so care must be taken not to lose or distort the features of interest.

Our filter design focuses on filtering out AC interference and the baseline as an option. The cut-off frequency and transition frequency are parameters of the filter that can be configured by the user. The implementation is relatively simple, where a FFT-based frequency-domain filter is presented, as shown in figure 3.36. A spectral mask is generated according to the defined parameters. It can be configured for eliminating higher frequency component as well as baseline. An Inverse FFT (IFFT) is used to regenerate the signal.

A few things should be considered when using an FFT. First the FFT length should be fixed and should be a power of 2. The input signal must be cut into successive segments of the length of FFT and reconstruction quality must be guaranteed. Second, the input segment is usually non-periodic and is extended periodically by repeating itself. However, we cannot ensure that the end point of the signal and the starting point is continuous. Sharp transition may exist. Usually people use windowing and overlapping to avoid this, for example, the

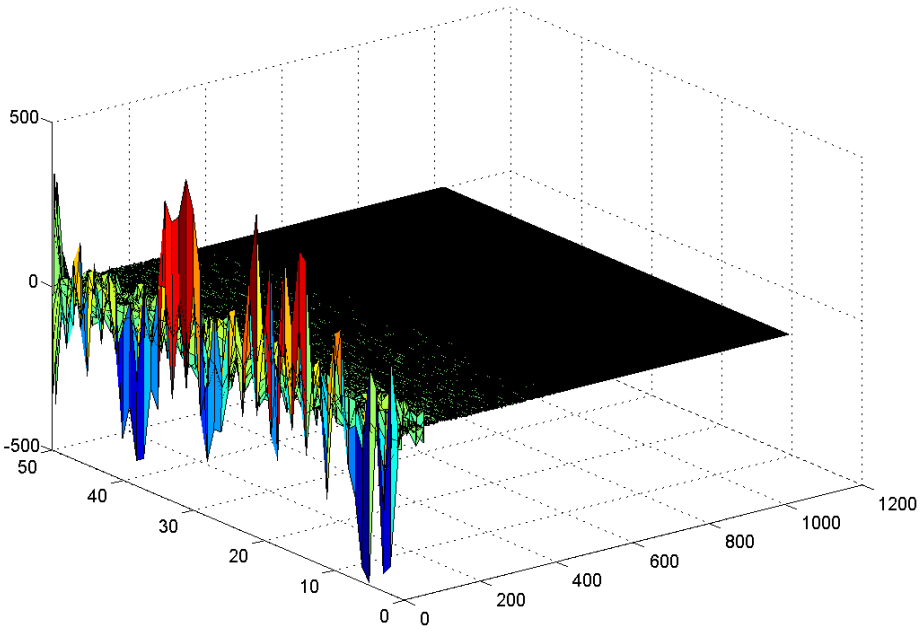


Figure 3.34: DCT Coefficient Matrix of Record 117 by Aligning Beats

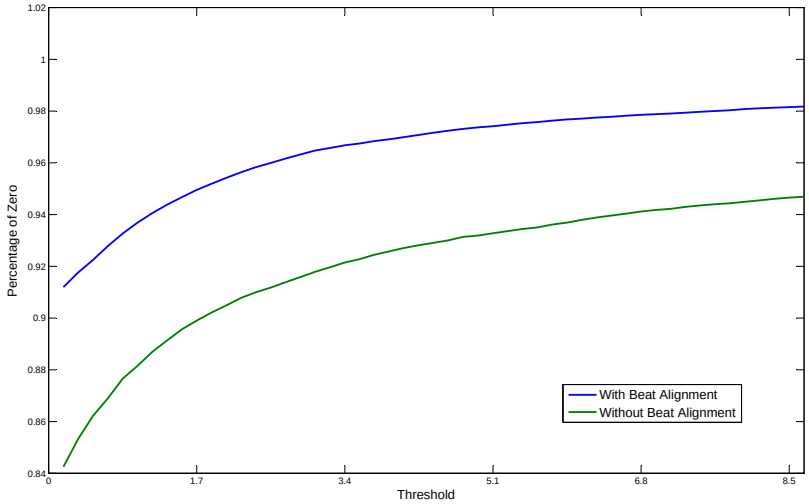


Figure 3.35: Comparison of Sparseness of DCT Coefficient Matrix with and without using Beat Alignment

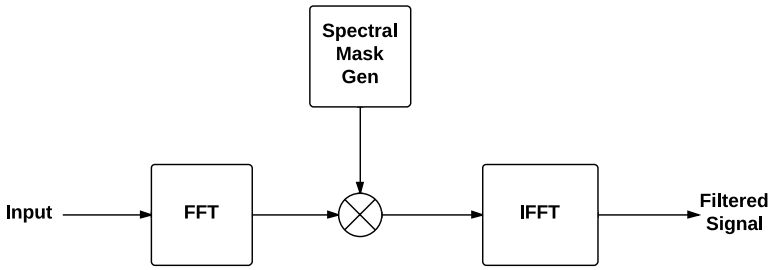


Figure 3.36: A Basic Band-pass Filter

Hanning window of 50% overlap which makes the first half and the last half of the signal connected continuously. The drawback of this method is the 50% overlap, where every sample point of the signal is affected by multiplication. In addition, to recover the signal, a compensate window is required. This adds one more multiplication. Due to these shortages, we use a new method called crossfading, to overcome those side effects.

The crossfading method improves the continuity in a more efficient and flexible way. An example signal input for FFT is shown in figure 3.37. If we directly extend the signal periodically, the extended signal clearly contains sharp transition part, as shown in the red circles of the lower plot in figure 3.37. This is not favorable for an accurate FFT.

Thus, we overload a small signal segment before our region of interest (ROI) and a signal segment after it. We name them as OV1 and OV2, as in figure 3.38, which have the same length. Then we apply the crossfading function, as shown in the lower plot in figure 3.38. OV1 and OV2 are gradually faded towards each side, respectively. The shape of the faded part is similar to the Hanning window, but we do not add any computations on ROI. This avoid errors introduced by calculation.

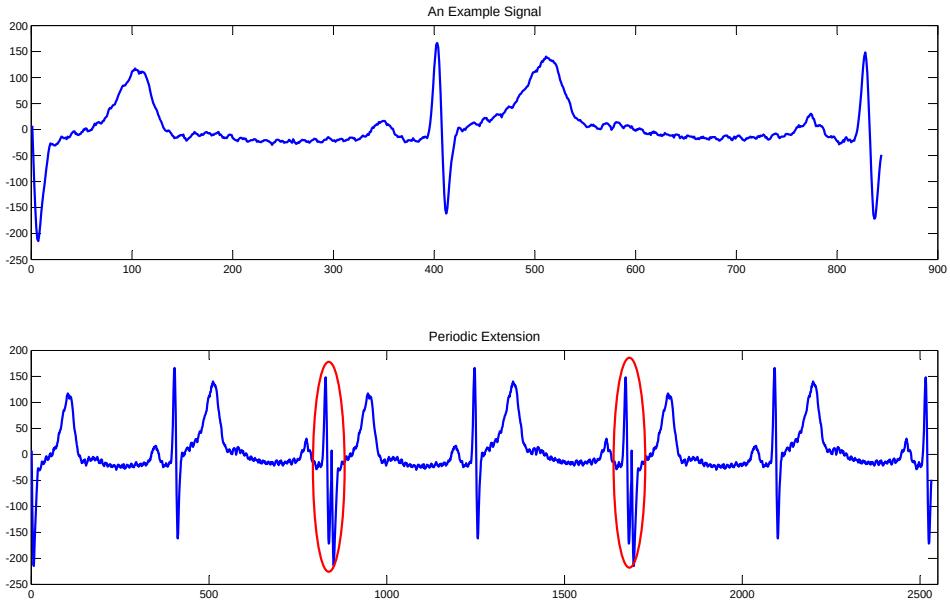


Figure 3.37: Example Signal and Periodic Extension

The final step is the overlap of OV1 and OV2, namely adding them together. Because of the fading, the first half of the OV1+OV2 result is more like OV1 but the second half is more like OV2. Since OV1 is connected with the starting point of the ROI and OV2 is connected with the ending point. If we do periodic signal transform on this FFT segment, the signal will have a much better continuity, as shown in figure 3.39. By crossfading we solve the problem of FFT calculation efficiently.

Oversampling is another option for the filter. This can be done by utilizing FFT coefficients and inserting zeros. Oversampling does influence the accuracy of peak detection and the compression performance a bit. The relation between the oversample factor and the compression ratio is shown in table 3.10. Note that the compression ratio increases following the increasing of the oversample factor.

Table 3.10: Compression Ratio of Different Oversampling Rate

Oversample Factor	CR (PRD1=1%)	CR (PRD2=1%)
1	5.8	3.5
2	5.9	3.6
4	6	3.6
8	6	3.6

Finally, we present our filter as in figure 3.40. We observe that an extra output is added for the noise output, because we need it to measure the quality of the com-

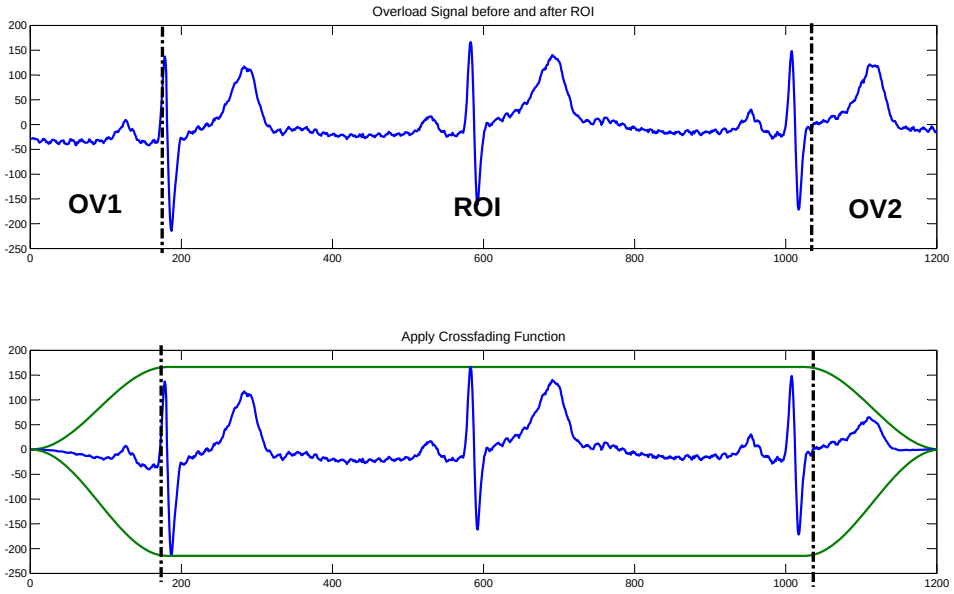


Figure 3.38: Crossfaidng Example 1

pressed signal. The filter is enabled by filteron. The signal Osr is the oversample factor.

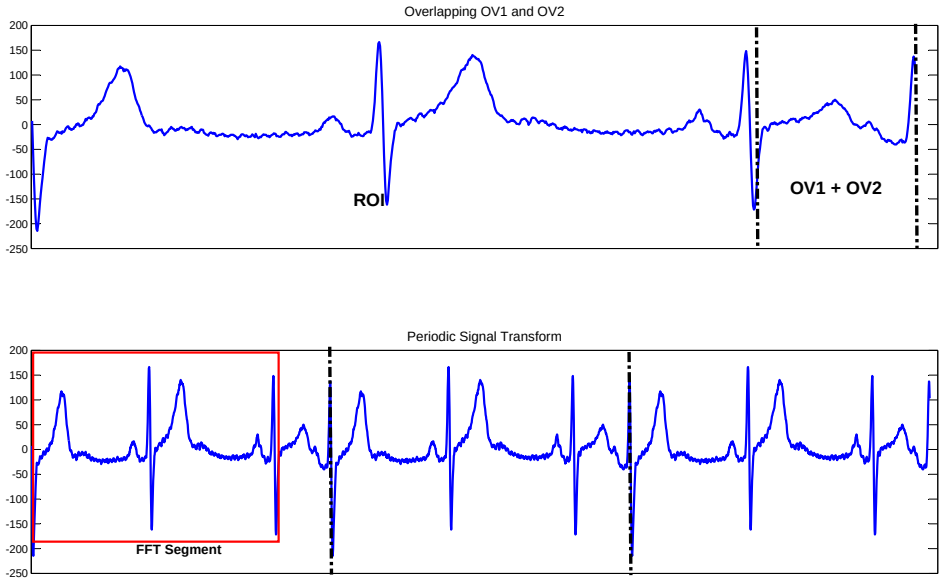


Figure 3.39: Crossfaidng Example 2

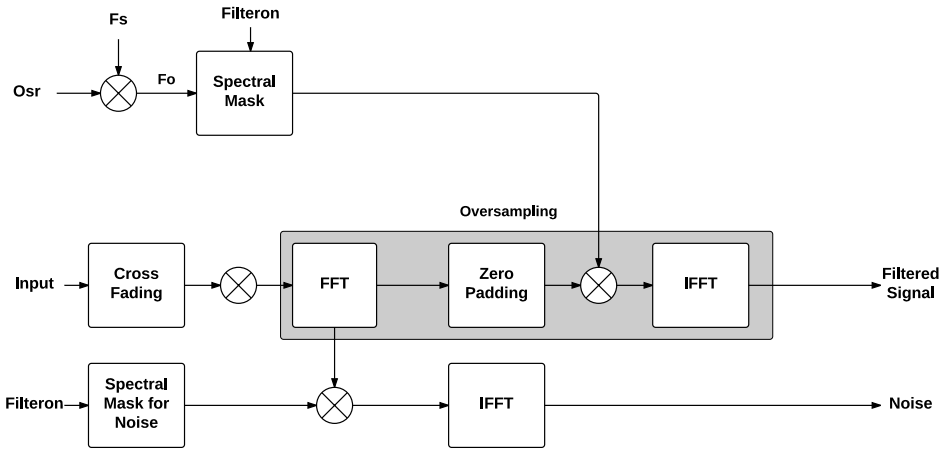
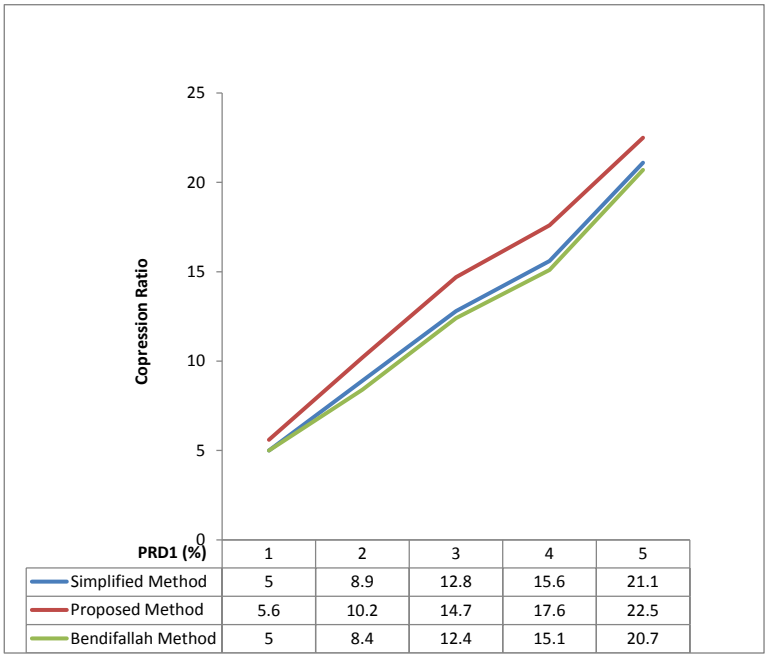


Figure 3.40: Filter with Oversampling

3.4 Results and Discussion

In previous sections, empirical findings and compression performance were discussed extensively. In this section, we focus on the comparison between our pro-

posed algorithms and existing work. The default data input is as mentioned in section 3.1.5. For each record of the chosen MIT-BIH dataset, we used the first 30 minutes of the signal on the first channel. Each time a 1-minute segment was compressed. Compression ratio results were calculated as average values. The error tolerance for quality control is 10 %.



PRD1 (%)	1	2	3	4	5
Simplified Method	5	8.9	12.8	15.6	21.1
Proposed Method	5.6	10.2	14.7	17.6	22.5
Bendifallah Method	5	8.4	12.4	15.1	20.7

Figure 3.41: Comparison of the Simplified Method, the Proposed Method and the Bendifallah Method

The proposed method is discussed in section 3.3 while the method in section 3.2 can be regarded as a simplified method, because the difference between it and the proposed method is the use of beat detection. Moreover, the simplified method has a similar structure as the method of Bendifallah [8], thus we add the Bendifallah method for comparison. A comparison of the compression ratio is shown in figure 3.41. As expected, the proposed method performed better than the simplified method and the Bendifallah method given different specification for PRD1. The compression ratio is improved by 10%-15%. There is a trade-off between complexity and compression ratio, for the proposed method, the compression ratio was increased by adding beat detection to the simplified method, where extra resources were brought in. This kind of choice can be made during the implementation.

A particular issue is that, we find in many cases, comparison of compression ratio is unfair without considering quality. For example, figure 3.42 shows the error between the input and the de-compressed data as different quality. The compression ratio increases significantly from 5 to 73, with a change in PRD2 from 2% to 20%, while the lower quality signal has far larger errors. In real

applications, we cannot ignore these quality differences because it may affect the quality of diagnosis. A very low quality signal is not acceptable, even though the compression ratio is very large. We introduce a measurement that we feel fairly combines compression ratio and quality specification (PRD) together.

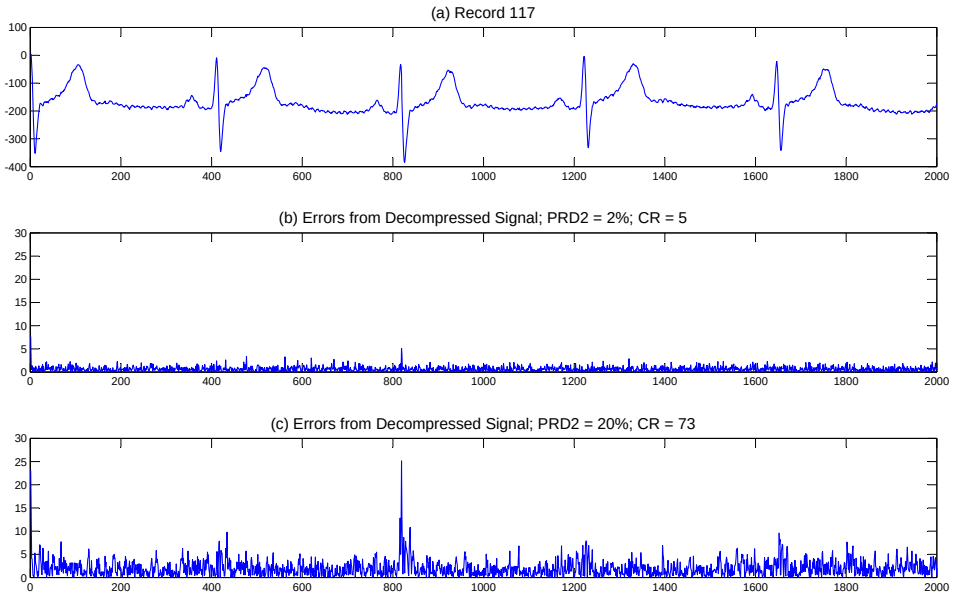


Figure 3.42: Comparison of 2 Different Quality and Compression Ratio Set

Before introducing it, experiments on our proposed algorithm with different PRD1 requirements were conducted. The relation between PRD1 and CR is shown as in figure 3.43. We observe that the relation is almost linear, where the linear function can be defined as:

$$CR = k * PRD1 + b \tag{3.26}$$

The CR equals to b when the PRD1 is zero, which means it is the compression ratio for lossless compression. In our case, we always consider the lossy compression, so that we assume there is no compression when PRD1 is zero, i.e., b = CR = 1.0. The ratio between PRD1 and CR can thus be calculated as:

$$k = \frac{CR - 1.0}{PRD} \tag{3.27}$$

Since the k is invariable with a change of PRD and CR, it can be used to evaluate or characterize the algorithm. We define this ratio as the quality score (QS). The definition is:

$$QS = \frac{CR - 1.0}{PRD} \tag{3.28}$$

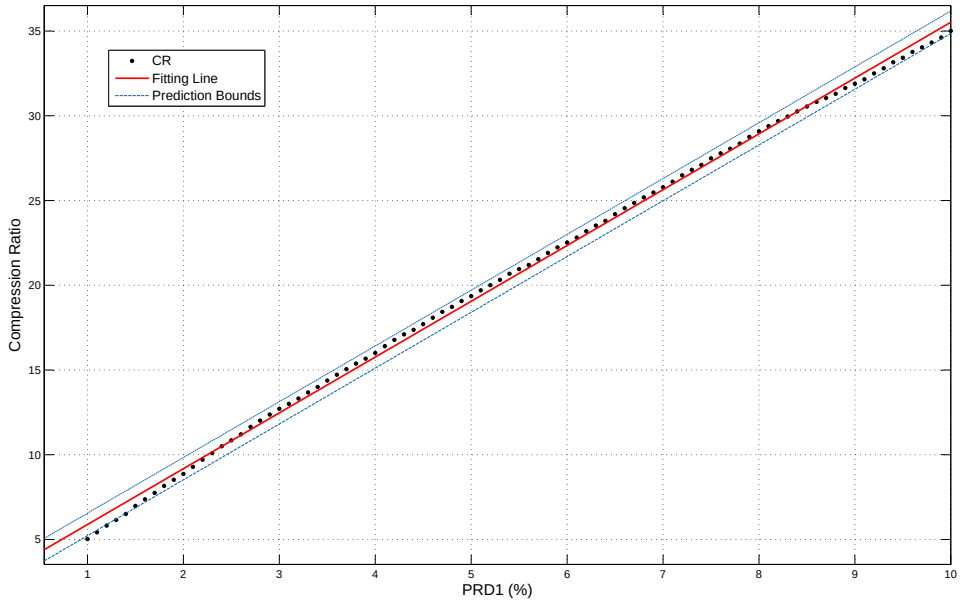


Figure 3.43: Relation between PRD1 and CR

From the definition of PRD, the larger the value is, the lower the reconstructed quality is. The compression ratio reflects the compression capability. In equation (3.28), a combination of a high PRD and a high CR may give the same result as a combination of a small PRD and a small CR. This reveals the true ability of a compression algorithm to a certain degree.

We use the QS to compare our algorithm with existing work. However, there are 3 types of PRD. To make a fair comparison, the QS is always calculated on the basis of the same PRD type.

We present our method as the scheme of section (3.3), with the oversampling factor of 4. Table 3.11 compares our result with existing work on the measurement of PRD1.

Table 3.11: Result Comparison 1

Method	Features	Year	PRD1 (%)	CR	QS
Bendifallah [8]	DCT	2011	1.0	5	4
Bilgin [9]	JPEG2000	2003	1.0	4	3
SPIHT [36]	DWT	2000	1.1	4	2.7
Alshamali [3]	DWT	2009	3.8	12.75	3.1
Simplified Method		2014	1.0	5	4
Proposed Method		2014	1.0	5.6	4.6

The simplified method has a similar architecture as Bendifallah method, accord-

ingly the compression ratio is similar. By utilizing correlations between ECG cycles, our proposed method has superior QS over others.

Quality is important and the best measurement is PRD2. That is our preferred quality measurement. There are very few examples in other work using PRD2 for evaluation, even though it gives the best quality estimate. Table 3.12 shows the comparison based on PRD2. Our work gives clearly much better result, which is the only algorithm that has the QS over 2.0.

Table 3.12: Result Comparison 2

Method	Features	Year	PRD2 (%)	CR	QS
Zigel-ASEC [59]	Synthesis Compressor	1997	4.0	6.9	1.5
Zigel-SAPA [59]	SAPA	1997	9.6	6.9	0.6
Miaou [40]	DWT	2000	6.3	9.4	1.3
Batista [7]	DCT	2001	3.3	6.9	1.8
Simplified Method		2014	1	3.4	2.4
Proposed Method		2014	1	3.6	2.6

All of the above evaluation use the chosen dataset from MIT-BIH database. For a more realistic evaluation, an IMEC dataset were available to test our method, an example is shown in figure 3.44. It includes a lot of artifacts and an obvious baseline, where utilization of appropriate filter settings is necessary. The plot below shows in figure 3.44 the filtered output, we can see the baseline is removed and high frequency noise is absent. The compression ratio result is shown as table 3.13, where the compression ratios are as high as 14.2 to have a "very good" quality of the reconstructed signal after filtering out the noise. When using the PRD2 measurement, the compression ratios are still relatively high. We preset the filter cut-off frequency as 49 Hz. The offset of the signal is 2048.

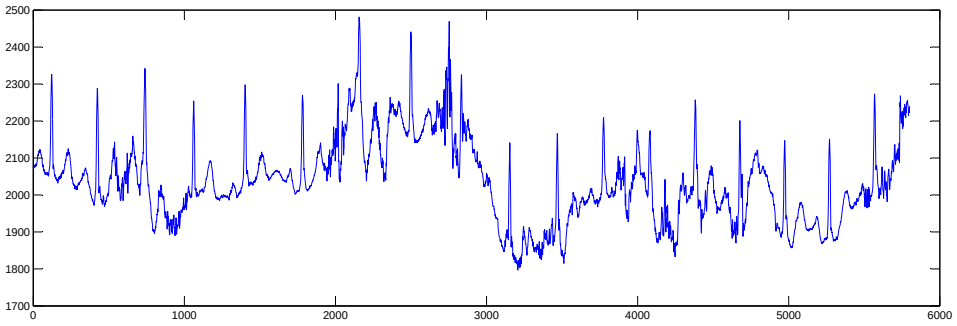


Figure 3.44: IMEC Record 1105

In summary, our proposed algorithm achieves a competitive compression ratio with a guaranteed quality.

Table 3.13: Result on IMEC Data of the Proposed Method

PRD	Filter	CR
PRD1=1%	No	4.3
PRD2=1%	No	3.8
PRD1=2%	No	7.0
PRD2=2%	No	6.6
PRD1=1%	Yes	10.8
PRD2=1%	Yes	9.3
PRD1=2%	Yes	14.2
PRD2=2%	Yes	13.7

4 Filter Implementation

This chapter discusses the filter implementation. The execution of some algorithms can be accelerated by dedicated hardware. The benefit from hardware implementation is not only on timing performance, but also on energy efficiency. Among the hardware implementation techniques FPGA and ASIC stand out. Field programmable gate arrays (FPGA) accomplishes a function by configuring its programming logic and gates. Since it already has the fixed gates and logic on it, the performance is limited both on area and power. However, it offers fast time-to-market and a simpler design cycle, due to no lower-level manufacturing steps are needed. As a compromise, it is a good choice for fast development and flexibility.

Application-specific integrated circuit (ASIC), on the other hand, is capable to have optimal performance. Different from FPGA and other techniques, the ASIC development requires the fabrication time, so it takes much more time than the FPGA development. However, the ASIC design can be very flexible because of its fully hardware customization. With the improvement of technology over the years, the complexity of an ASIC has been increased dramatically. There is no way to manually develop a 100-million-gate ASIC design by logic. Advanced and powerful tools are needed to support the design, implementation and verification of ASICs. The ASIC design is more compact, and it significantly reduces the unit cost, area and the power consumption. In our hardware implementation, we choose to use ASIC because of its benefit in lower resource requirement. Besides that IMEC has proven technology and development environment for a faster ASIC design.

Nevertheless, we still have to consider about the flow from high level design(Matlab code) to low level implementation (silicon). There is no a direct path to go from top to end. Traditionally, the design flow for a fixed-point design is shown in figure 4.1.

We see there is at least two conversions needed in our case. One is the Matlab to C. To achieve what standard Matlab library functions could perform, we have to write their C implementations on our own. This could increase the code volume drastically. Another conversion is from C to RTL, where hardware description language (HDL) is used. This stage we need to consider the logic and also the concurrent execution. If the C code itself is complex and contains a lot of controls, the re-writing could takes a lot of time. Because in HDL all the controls are handled by using registers. Timing is another issue that could slow our development. In C or Matlab execution, the code is sent to processors which data and instructions are executed serially. But in hardware simulation, functions and

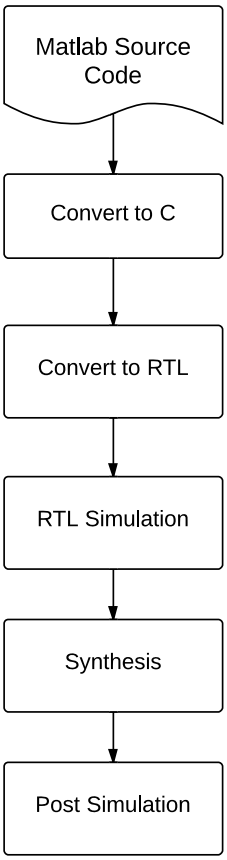


Figure 4.1: Design Flow for Matlab Design to ASIC

operations are running concurrently, which many extra controls are needed. Only when the RTL simulation is successful, the synthesis and post simulation can be done to ensure its functionality on hardware. Once there are errors or improvements made on high level design, all the lower level designs need to be modified. These iterations are obviously unfavorable in development.

Supported by Mathworks, the supplier of Matlab, we shorten our development duration by using the HDL Coder, which acts as a "direct" path from Matlab source code to RTL design. We realize our design with this coder and in latter sections we will discuss the details.

4.1 Mathworks HDL Coder

The HDL Coder generates synthesizable HDL codes, such as VHDL and Verilog, from Mathworks high level models, i.e Matlab and Simulink. It supports

both FPGA and ASIC implementations. It is able to generate the corresponding testbench. Working together with Mathworks HDL verifier, it is also capable to verify the design and do FPGA-in-the-loop cosimulation. Assisted by its workflow adviser, users can easily generate the HDL codes by simple configuration.

Furthermore, HDL coder provides options on loop optimization and streaming optimization. Multichannel designs are also supported. According to the type of the high level model, the workflow of the HDL coder has two types: Matlab workflow and Simulink workflow.

4.1.1 Matlab Workflow

Matlab, as the most popular software and language for numerical computation and programming, has been proven successful in today's industry and education. The syntax of this language is simple and the coding style is extremely flexible. It supports vast amount of libraries on various technical field and presents comprehensive example solutions and guidelines online. Researchers have widely used Matlab as the tool to solve digital signal processing problems. Due to those advantages, we also use it for development of ECG data compression.

The HDL Coder Matlab workflow is shown as the following figure: Starting with the Matlab source code, but it requires several modifications, which are:

(a) Fixed Point Conversion: Defining Fixed Point format for each variables and corresponding fixed-point operations.

(b) Loop Checking: Defining bounds for each loop. To make sure loops are correct bounded.

(c) Function Checking: Check if the function is supported by HDL coder. If not, define it manually.

An basic requirement for the Matlab design input is that both input and output of it should not be matrix or structures. Once the modified Matlab code passes all the pre-requisite, validation can be processed to verify if its function is correct. If it passed, then we can generate RTL codes.

The Matlab HDL Coder supports several types of optimizations, such as pipeline, resource sharing and loop streaming. But those require extra configurations and some more modifications. For example, a loop cannot be streamed if the index is not increased by 1 on each iterations. So the loop checking should be done once again. Only by several iterations that the code generation can succeed. The most significant drawback of Matlab HDL coder is that it is not very stable during iterations, in particular, the validation stage. The fixed-point conversion takes a lot of work, because there are a great deal of variables and operations. All in all, our experience with the Matlab HDL coder is not very pleasant. Mathworks also provides better support for Simulink workflow instead of for Matlab.

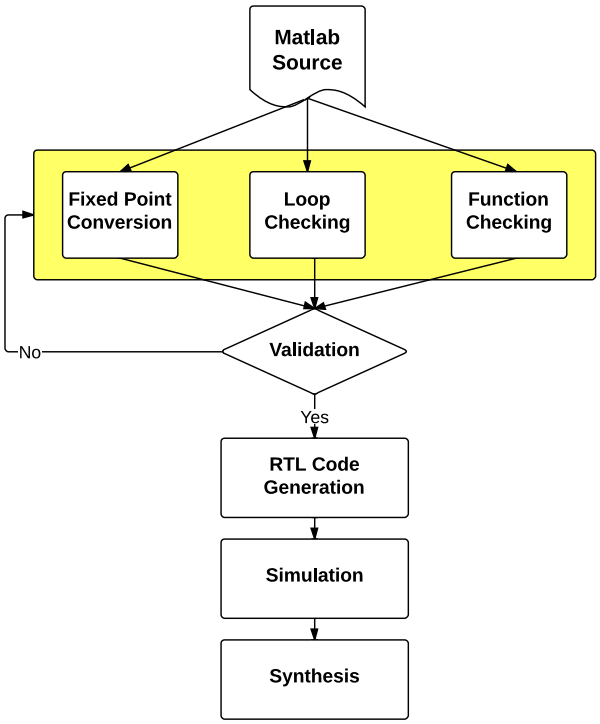


Figure 4.2: HDL Coder Matlab Workflow

4.1.2 Simulink Workflow

Simulink is a block diagram environment for model-based design. The difference between it and Matlab is that it focuses more on the system behavior rather than numerical result. Simulink supports building and simulating. Models as hardware simulations, are triggered and running by signals, which makes the Simulink closer to the realistic system. But this also means that usually a Matlab code cannot be directly converted to Simulink, because everything was executed sequentially in Matlab.

The benefit using Simulink is its ability of generating codes, especially for HDL coder. The HDL coder is initially developed for assisting Simulink for the system design. HDL code generation from Simulink is more stable and easier than from Matlab. Generally, the HDL Coder Simulink workflow is as in figure 4.3. The path is simpler than in figure 4.2. The essential part here is to build Simulink model from Matlab code. It depends on how is one accustomed to the behavior of the Matlab and how those Simulink blocks are arranged and used. Some of blocks can be customized by using Matlab functions or system objects. Block checking is also required to see if it is supported by HDL code generation. Since the design is model-based, signals are required to connect and synchronize blocks.

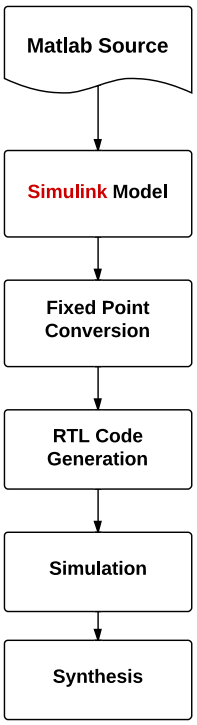


Figure 4.3: HDL Coder Simulink Workflow

This is different from Matlab code. In total some extra work should be done to successfully convert a Matlab code to a Simulink model.

Since HDL coder only supports fixed point implementation, fixed-point analysis is needed. In Simulink, the model property checking is automated by the workflow adviser. Optimization options are more practical than Matlab HDL coder. Users can optimize critical path, maximum computation latency, etc. Streaming techniques are highly supported in Simulink HDL coder. Hardware aspects, such as sample rate and RAM ports, are also considered in Simulink models. Overall, by using Simulink the code generation works better and more stable. We will also choose this coder for our implementation.

4.2 Implementation

We implement the accelerator for the filter (containing oversampling) block of our design, because it contains FFT and IFFT, which is potential to be highly optimized by hardware implementation. Also, the filter block is not so complex that we are able to finish the implementation by the thesis project duration.

As discussed above, we choose Simulink workflow over Matlab workflow for its

better support and stability. The design of the filter is discussed in section 3.3.3, which we will not repeat here. In the following sections we will discuss steps from building Simulink model to synthesis.

4.2.1 Simulink Model Design

Figure 3.40 shows the block diagram of the filter and oversampler. To ease the hardware design, we generate the spectral mask and the crossfading coefficients by software. They can be used as look-up tables in the simulink model. So the filter Simulink system is as figure 4.4.

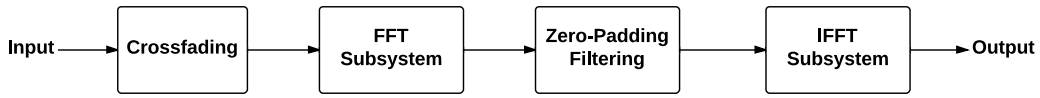


Figure 4.4: Simulink Model of Filter

Signals will be streamed in the crossfading block. It contains a lot of controllers and 2 RAMs for buffering the overlapped part. Crossfaded data will be send to the FFTSubsystem block, where a minimal resource FFT supported by HDL coder is used. The HDL coder has two streaming FFT block and one minimal resource FFT block. In our design, the minimal resource FFT block is chosen because we want to minimize area as well as energy. Zero padding is achieved by buffering and changing rate, which benefits from Simulink features. By applying spectral mask as look-up table, filtering is achieved. The IFFTSubsystem is based on the FFT block, but with conjugation. The function of the filter is thus accomplished.

Fixed point format are analyzed by Matlab. But the result is just an approximation because we use different FFT in Matlab and Simulink, of which accuracies are not the same. In the implementation we use 28-bit and 29-bit word for FFT and IFFT, respectively. Each contains 9 bits for fractional. The output for the filter system is thus 28 bits, and the quality is acceptable where PRD1 is 0.2% and PRD2 is 0.7%. Decreasing the bit-width easily increases the PRD2 to over 1.0%. Since a "very good" reconstructed signal should have PRD2 of only 0-2% [60], we cannot accept any larger quality loss in this stage.

4.2.2 RTL Code Generation and Synthesis

We follow the HDL Workflow advisor to generate HDL codes. The target is set as Generic ASIC/FPGA. All environment settings are configured to pass the model checking. We generate VHDL RTL codes and testbenches on the default settings.

The structure of the generated RTL design is exactly as figure 4.4. This benefits the debugging.

4.3 Results and Discussion

The structure of this filter looks simple. However, manual work on it still requires a lot of work. There are many controls in crossfading block and FFT block. Also the optimized RTL models for those blocks need to be studied. Fixed point operations, buffers, and testbenches need handwriting. Synchronization is another issue to be solved. It will be very difficult to achieve our goal in a very limited time. The developing speed is the essential reason for choosing to use HDL coder.

The RTL simulation is handled on Cadence NCSIM and ModelSim by using the automatically generated testbench. The testbench works in such a way that it collects the result from Simulink simulation and compares it with the RTL simulation result. If there is any difference, the testbench will report errors. The ModelSim simulation report shows no errors. The waveform of the simulation is shown in figure 4.5. The signal filterout_ref is the reference signal from Simulink simulation. The signal filterout is from the ModelSim simulation. Figure 4.5 shows that filterout_ref and filterout are the same. This proves the correctness of the RTL design.

A drawback in our Simulink model is the relatively large bit-width, which is 29 bits. This will significantly increase area usage of memory. But we cannot reduce its size because this would introduce quality loss. Moreover, Simulink models work in a streaming way, which most components will not be re-used. If there are a lot of memories in the design, the area will be extremely large.

In summary, the Mathworks HDL coder is a new method for synthesis from high level. The direct combination of hardware implementation and software development makes sense for an efficient and flexible design. But the tool still lacks of enough optimization and requires certain manual work. The training and learning process are mandatory for tactically design. The problem of time-to-market in hardware design has long influenced the industry and technology progress due to great time-consuming of low-level design. Mathworks HDL Coder tool provides a solution for faster development, but there is a lot of work to do before it can solve problem perfectly.

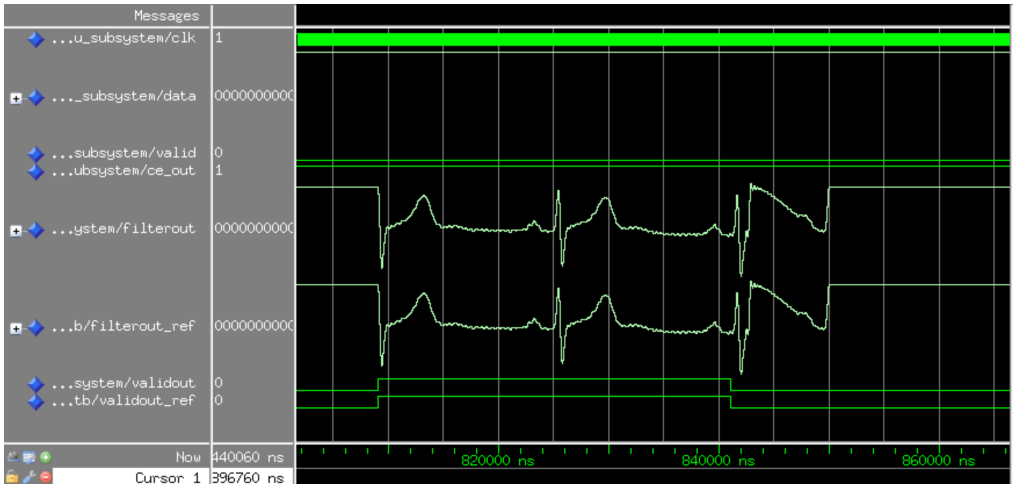


Figure 4.5: Waveform of the ModelSim Simulation for the Filter/Oversampler RTL Design

5 Conclusions and Future Work

5.1 Conclusions

This MSc project was set out to develop a practical ECG signal compression algorithm, which could maintain desired quality. Based on the transform-based compression scheme, the thesis explored different methods of each block so as to achieve a superior compression ratio while keeping signal quality. The thesis also investigated and proposed a hardware implementation for a component.

We have discussed a lot of empirical findings in Chapter 3 and Chapter 4. We synthesize our findings as follows:

- The DCT-based compression scheme is proved to be effective for ECG signal compression, which can achieve average or above average compression ratio. The scheme is also relatively simple to implement. Quality control is easily accomplished by searching the appropriate quantization parameters.
- As the major ECG signal quality measurement, three types of percentage root differences (PRDs) were investigated. The PRD2, which removes the DC component in calculation, is suggested. Because it only shows the fluctuating difference between the original signal and the reconstructed signal, so that it avoids the effect of the DC component. Nonetheless, most previous papers used PRD0 or PRD1 for showing a better compression ratio.
- The quantized DCT coefficient matrix was studied, where the 2-stage encoding approach was applied to significantly improve the compression ratio. Recursive splitting method eliminates the precision problem of the lossless arithmetic encoding.
- Correlation of each ECG cycle (beat) can be utilized to improve compression ratio, especially when the quality requirement is not so high. Beats are recognized by detecting R peaks. Thus an R peak detector was designed.
- To speed up the hardware development, Mathworks HDL coder was used. This tool reduced the development time but required a training process. It allows to do quick iterations and elaborate on different approaches. Results in terms of performance were not acceptable in our case.

To conclude, this thesis project proposed a DCT-based ECG compression algorithm by utilizing ECG cycles. It is quality controlled and has superior compression ratio. The algorithm were designed block-wise, so that by tuning or replacing method of each block, performance and complexity can be changed. This makes

the design very flexible and powerful. The implementation of hardware accelerator was carried out on Mathworks HDL coder, a high level RTL design tool. The design flow indicated a new and fast way for hardware development. And the generated RTL design proves to be correct by simulation.

5.2 Recommendations for Future Work

Following future work is recommended for improvement:

- **Computational Complexity** Numbers and types of operations of proposed algorithms can be studied. Complexity is possibly improved by future research.
- **R-peak Detector** The proposed R-peak detector was an empirical design. Due to lack of the real R-peak data of MIT-BIH database, we were not able to show the accuracy. Future research should prove the accuracy and improve the detector when possible. By looking at the other algorithms, to enhance the performance, we hope to increase the performance margin.
- **Database** So far we only investigated our algorithms on MIT-BIH database and the IMEC database. Various databases can be used in future research. Also data of different sampling frequencies can be used and investigated.
- **Segment Length** Results and examples of previous chapters are based on dividing the input signal into 1-minute segments. The length of segment can be changed flexibly.
- **Multi-Channel** In reality, ECG signal is a multichannel signal. But our assumption was always made on single channel compression. Future research could take multichannel into consideration.
- **Hardware Implementation and System Design** Future research can seek on new efficient solutions for hardware implementation, either by high level design tools or by conventional design flow.

Simulink Model

A

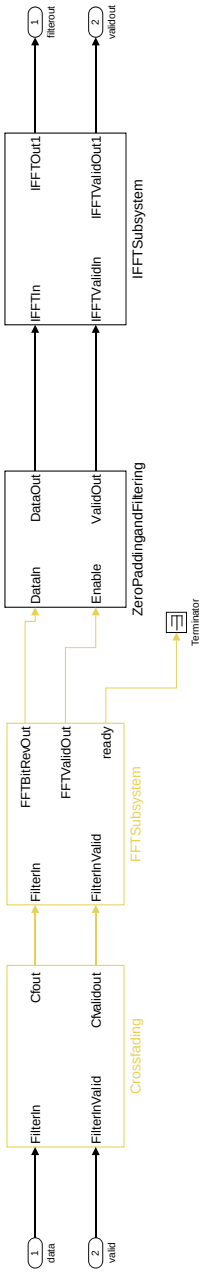


Figure A.1: Simulink Model of the Filter Subsystem

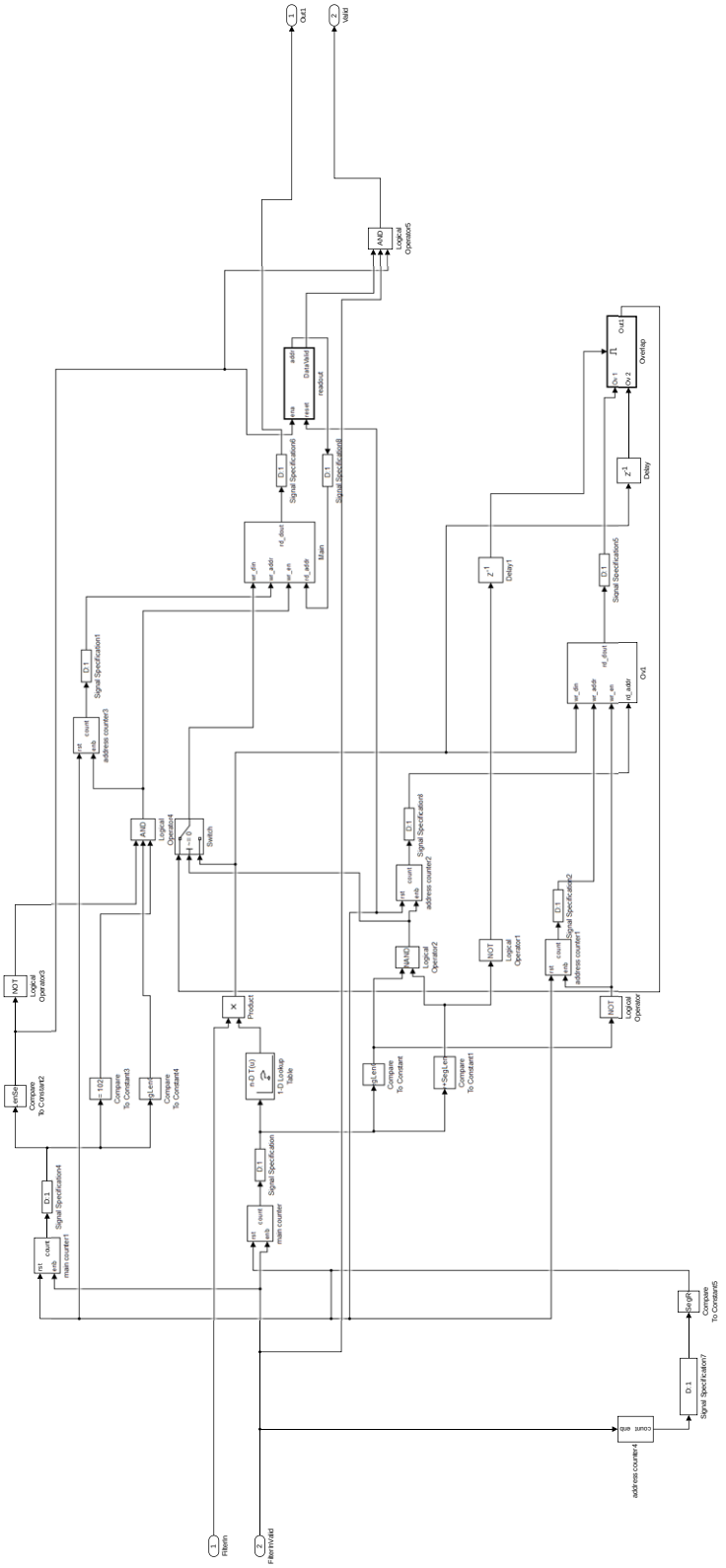


Figure A.2: Simulink Model of the Crossfading Block

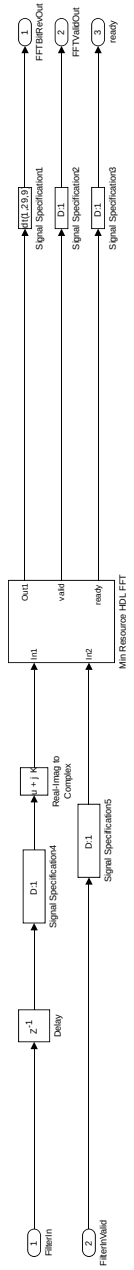


Figure A.3: Simulink Model of the FFTSubsystem Block

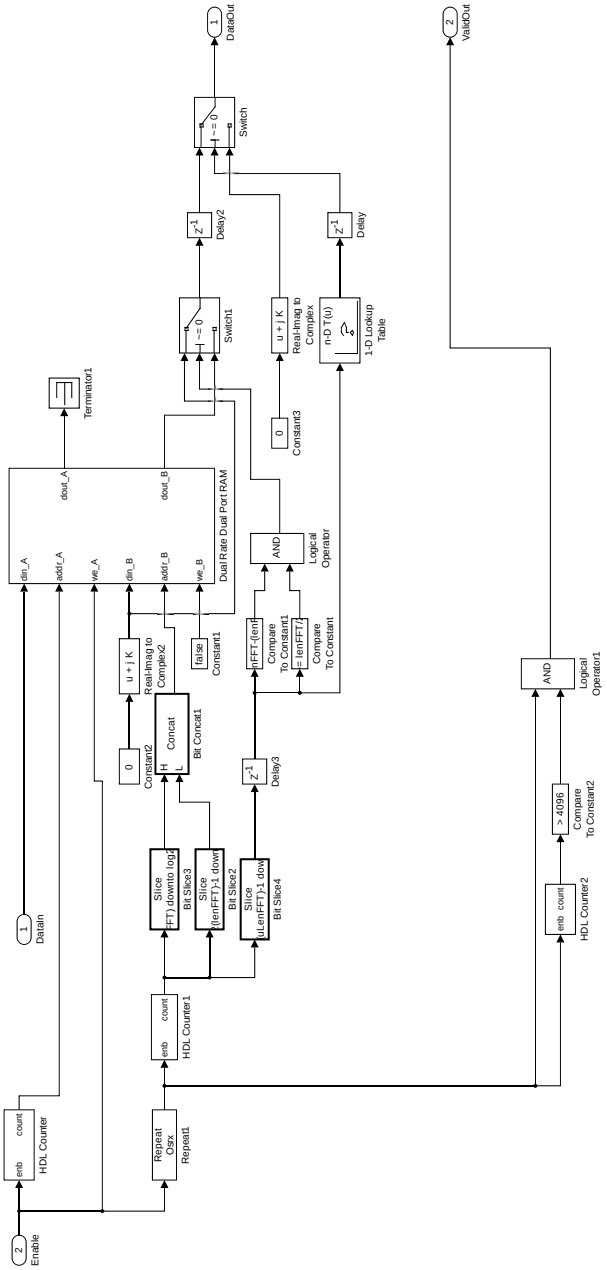


Figure A.4: Simulink Model of the ZeroPaddingandFiltering Block

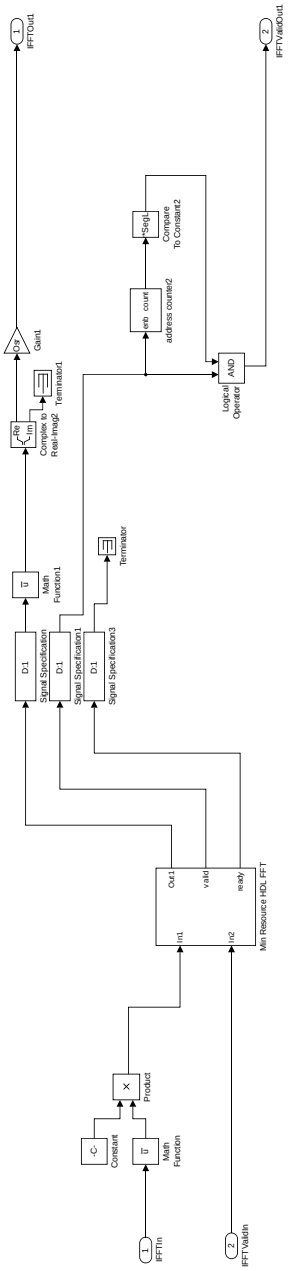


Figure A.5: Simulink Model of the IFFTSubsystem Block

Bibliography

[1] Sven Ole Aase, Ranveig Nygaard, and John Håkon Husøy. A comparative study of some novel ecg data compression techniques. 1998.

[2] Vernon A Allen and John Belina. Ecg data compression using the discrete cosine transform (dct). In *Computers in Cardiology 1992, Proceedings of*, pages 687–690. IEEE, 1992.

[3] Ahmad Alshamali and Amjed S Al-Fahoum. Comments on" an efficient coding algorithm for the compression of ecg signals using the wavelet transform". *Biomedical Engineering, IEEE Transactions on*, 50(8):1034–1037, 2003.

[4] Emran Mohammad Abu Anas, Md Iftekhar Hossain, Md Shah Afran, and Shehrin Sayed. Compression of ecg signals exploiting correlation between ecg cycles. In *Electrical and Computer Engineering (ICECE), 2010 International Conference on*, pages 622–625. IEEE, 2010.

[5] Ziya Arnavut. Ecg signal compression based on burrows-wheeler transformation and inversion ranks of linear prediction. *Biomedical Engineering, IEEE Transactions on*, 54(3):410–418, 2007.

[6] Ross Arnold and Tim Bell. A corpus for the evaluation of lossless compression algorithms. In *Data Compression Conference, 1997. DCC'97. Proceedings*, pages 201–210. IEEE, 1997.

[7] Leonardo Vidal Batista, Elmar Uwe Kurt Melcher, and Luis Carlos Carvalho. Compression of ecg signals by optimized quantization of discrete cosine transform coefficients. *Medical engineering & physics*, 23(2):127–134, 2001.

[8] A Bendifallah, R Benzid, and M Boulemden. Improved ecg compression method using discrete cosine transform. *Electronics letters*, 47(2):87–89, 2011.

[9] Ali Bilgin, Michael W Marcellin, and Maria I Altbach. Compression of electrocardiogram signals using jpeg2000. *Consumer Electronics, IEEE Transactions on*, 49(4):833–840, 2003.

[10] Bachir Boucheham, Youcef Ferdi, and Mohamed Chaouki Batouche. Incorporation of long-term redundancy in ecg time domain compression methods through curve simplification and block-sorting. *International Journal of Signal Processing*, 1(3), 2005.

[11] A Enis Cetin, H Köymen, and M Cegiz Aydin. Multichannel ecg data compression by multirate signal processing and transform domain coding techniques. *IEEE transactions on bio-medical engineering*, 40(5):495–499, 1993.

[12] J Chen, J Ma, Y Zhang, and X Shi. Ecg compression based on wavelet transform and golomb coding. *Electronics Letters*, 42(6):322–324, 2006.

[13] Jianhua Chen, Fuyan Wang, Yufeng Zhang, and Xinling Shi. Ecg compression using uniform scalar dead-zone quantization and conditional entropy coding. *Medical Engineering & Physics*, 30(4):523–530, 2008.

[14] Seong-Beom Cho, Young-Dong Lee, Do-Un Jeong, and Gi-Hyun Hwang. Implementation of novel ecg compression algorithm using template matching. In *Computing and Convergence Technology (ICCCT), 2012 7th International Conference on*, pages 305–308. IEEE, 2012.

[15] Josh Coalson. Flac-free lossless audio codec. *Internet: http://flac. sourceforge. net*, 2008.

[16] JR Cox. Compact digital coding of electrocardiographic data. In *Hawaii International Conference on System Sciences, 6 th, Honolulu, Hawaii*, pages 333–336, 1973.

[17] JR Cox, FM Nolle, HA Fozzard, and GC Oliver. Aztec, a preprocessing program for real-time ecg rhythm analysis. *Biomedical Engineering, IEEE Transactions on*, (2):128–129, 1968.

[18] Deborah A Dipersio and Roger C Barr. Evaluation of the fan method of adaptive sampling on human electrocardiograms. *Medical and Biological Engineering and Computing*, 23(5):401–410, 1985.

[19] Anna MR Dixon, Emily G Allstot, Daibashish Gangopadhyay, and David J Allstot. Compressed sensing system considerations for ecg and emg wireless biosensors. *Biomedical Circuits and Systems, IEEE Transactions on*, 6(2):156–166, 2012.

[20] David L Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289–1306, 2006.

[21] L.W. Gardenhire. Data compression for biomedical telemetry. 1965.

[22] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000.

[23] Golomb. Run-length encodings (corresp.). *Information Theory, IEEE Transactions*, 12(3):399–401, 1966.

[24] Allan VN Goodyer, Arthur J Geiger, and Willys M Monroe. Clinical fetal electrocardiography. *The Yale journal of biology and medicine*, 15(1):1, 1942.

[25] Alexander Grossmann and Jean Morlet. Decomposition of hardy functions into square integrable wavelets of constant shape. *SIAM journal on mathematical analysis*, 15(4):723–736, 1984.

[26] D Haugland, JG Heber, and JH Husøy. Optimisation algorithms for ecg data compression. *Medical and biological engineering and computing*, 35(4):420–424, 1997.

[27] David A Huffman et al. A method for the construction of minimum redundancy codes. *proc. IRE*, 40(9):1098–1101, 1952.

[28] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

[29] Masa Ishijima, Soon-Bum Shin, Gene H Hostetter, and Jack Sklansky. Scan-along polygonal approximation for data compression of electrocardiograms. *Biomedical Engineering, IEEE Transactions on*, (11):723–729, 1983.

[30] James Kennedy. Particle swarm optimization. In *Encyclopedia of Machine Learning*, pages 760–766. Springer, 2010.

[31] Byung S Kim, Sun Kook Yoo, and Moon H Lee. Wavelet-based low-delay ecg compression algorithm for continuous ecg transmission. *Information Technology in Biomedicine, IEEE Transactions on*, 10(1):77–83, 2006.

[32] Hyejung Kim, Refet Firat Yazicioglu, Sunyoung Kim, Nick Van Helleputte, Antonio Artes, Mario Konijnenburg, Jos Huisken, Julien Penders, and Chris Van Hoof. A configurable and low-power mixed signal soc for portable ecg monitoring applications. In *VLSI Circuits (VLSIC), 2011 Symposium on*, pages 142–143. IEEE, 2011.

[33] Shin-Chi Lai, Wei-Che Chien, Chien-Sheng Lan, Meng-Kun Lee, Ching-Hisng Luo, and Sheau-Fang Lei. An efficient dct-iv-based ecg compression algorithm and its hardware accelerator design. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 1296–1299. IEEE, 2013.

[34] Hanwoo Lee and Kevin M Buckley. Ecg data compression using cut and align beats approach and 2-d transforms. *Biomedical Engineering, IEEE Transactions on*, 46(5):556–564, 1999.

[35] Yong Ting Li, Xiao Yan Chen, and Yue Wen Liu. Multi-channel ecg signals compression algorithm using simultaneous orthogonal matching pursuit. *Advanced Materials Research*, 457:1305–1309, 2012.

[36] Zhitao Lu, Dong Youn Kim, and William A Pearlman. Wavelet compression of ecg signals by the set partitioning in hierarchical trees algorithm. *Biomedical Engineering, IEEE Transactions on*, 47(7):849–856, 2000.

[37] Stephane Mallat and Wen Liang Hwang. Singularity detection and processing with wavelets. *Information Theory, IEEE Transactions on*, 38(2):617–643, 1992.

[38] Hossein Mamaghanian, Nadia Khaled, David Atienza, and Pierre Vandergheynst. Compressed sensing for real-time energy-efficient ecg compression on wireless body sensor nodes. *Biomedical Engineering, IEEE Transactions on*, 58(9):2456–2466, 2011.

[39] Shaou-Gang Miaou and Heng-Lin Yen. Multichannel ecg compression using multichannel adaptive vector quantization. *Biomedical Engineering, IEEE Transactions on*, 48(10):1203–1207, 2001.

[40] Shaou-Gang Miaou, Heng-Lin Yen, and Chih-Lung Lin. Wavelet-based ecg compression using dynamic vector quantization with tree codevectors in single codebook. *Biomedical Engineering, IEEE Transactions on*, 49(7):671–680, 2002.

[41] SK Mukhopadhyay, M Mitra, and S Mitra. Ecg signal processing: Lossless compression, transmission via gsm network and feature extraction using

hilbert transform. In *Point-of-Care Healthcare Technologies (PHT), 2013 IEEE*, pages 85–88. IEEE, 2013.

[42] Salvador Olmos, Mar MillAn, Jose Garcia, and Pablo Laguna. Ecg data compression with the karhunen-loeve transform. In *Computers in Cardiology, 1996*, pages 253–256. IEEE, 1996.

[43] Davis Pan. A tutorial on mpeg/audio compression. *IEEE multimedia*, 2(2):60–74, 1995.

[44] Luisa F Polania, Rafael E Carrillo, Manuel Blanco-Velasco, and Kenneth E Barner. Compressed sensing based method for ecg compression. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 761–764. IEEE, 2011.

[45] Urs E Ruttimann and Hubert V Pipberger. Compression of the ecg by prediction or interpolation and entropy encoding. *Biomedical Engineering, IEEE Transactions on*, (11):613–623, 1979.

[46] Amir Said and William A Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *Circuits and Systems for Video Technology, IEEE Transactions on*, 6(3):243–250, 1996.

[47] PJ Schwartz, AJ Moss, GM Vincent, and RS Crampton. Diagnostic criteria for the long qt syndrome. an update. *Circulation*, 88(2):782–784, 1993.

[48] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[49] SinusRhythmLabels.svg. Schematic diagram of normal sinus rhythm for a human heart as seen on ecg.

[50] Karl Skretting, John Håkon Husøy, and Sven Ole Aase. Improved huffman coding using recursive splitting. In *Proceedings of Norwegian Signal Processing, NORSIG*, 1999.

[51] Gary J Sullivan. On embedded scalar quantization. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 4, pages iv–605. IEEE, 2004.

[52] Shen-Chuan Tai, Chia-Chun Sun, and Wen-Chien Yan. A 2-d ecg compression method based on wavelet transform and modified spiht. *Biomedical Engineering, IEEE Transactions on*, 52(6):999–1008, 2005.

[53] Bo Tao. Deadzone quantization method and apparatus for image compression, June 18 2002. US Patent 6,408,026.

[54] Kristian Thygesen, Joseph S Alpert, Allan S Jaffe, Harvey D White, Maarten L Simoons, Bernard R Chaitman, Hugo A Katus, Fred S Apple, Bertil Lindahl, David A Morrow, et al. Third universal definition of myocardial infarction. *Journal of the American College of Cardiology*, 60(16):1581–1598, 2012.

[55] K Uyar and YZ Ider. Development of a compression algorithm suitable for exercise ecg data. In *Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE*, volume 4, pages 3521–3524. IEEE, 2001.

[56] Mahes Visvalingam and J Duncan Whyatt. The douglas-peucker algorithm for line simplification: Re-evaluation through visualization. In *Computer Graphics Forum*, volume 9, pages 213–225. Wiley Online Library, 1990.

[57] Gregory K Wallace. The jpeg still picture compression standard. *Consumer Electronics, IEEE Transactions on*, 38(1):xviii–xxxiv, 1992.

[58] Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[59] Y Zigel, A Cohen, A Abu-Ful, A Wagshal, and A Katz. Analysis by synthesis ecg signal compression. In *Computers in Cardiology 1997*, pages 279–282. IEEE, 1997.

[60] Yaniv Zigel, Arnon Cohen, and Amos Katz. The weighted diagnostic distortion (wdd) measure for ecg signal compression. *Biomedical Engineering, IEEE Transactions on*, 47(11):1422–1430, 2000.