Data Compression for Implantable Medical Devices

LA Koyrakh

St. Jude Medical, St. Paul, MN, USA[1]

Abstract

Introduction: Implantable devices have limited memory, computational and battery power resources, while collecting, processing and transmitting out information from potentially many sensors. These limitations require that information within the devices be efficiently compressed. Such data compression presents a challenging task, as it must provide high fidelity of the waveform reproduction and high compression ratios on limited size data frames. Also, it must efficiently run on ultra low power hardware, and allow flexible configuration, based on the type of data to be compressed. Methods: The new compression algorithm was implemented as a bit accurate Matlab simulation, consisting of the following major steps: 1. Integer wavelet transform. 2. Quantization coupled with filtration. Two selectable quantization schemes could be utilized based on the signal properties: linear and dead-zone. Data filtration is performed on bit boundaries, which simplifies hardware implementation. The filtration thresholds are made different in different wavelet sub-bands, controlled by a single parameter. 3. Original adaptive data encoding. Our approach only requires basic logical operations such as bit counters and shifts, and is highly optimized for implementation in implantable device hardware. For high reliability each compressed data frame contains all information needed for decompression. Results: The algorithm was applied to data from the PhysioNet ECG compression test database. 40 *ECG frames of* 1024 *samples from* 4 *patients were sampled at* 250 *Hz,* 12 *bit resolution, and compressed with distortions below* 8%*. Compression ratios were* 9.3 ± 2.5*, consistently exceeding* 85% *of the theoretical limit based on bit entropy for each individual data frame. Conclusions: The compression algorithm is efficient on data collected by implantable devices, and could be used in various applications in both microprocessor and ASIC implementations, helping to reduce memory requirements and the battery energy spent on the information transmission to and from the implantable device.*

[1]The work was performed by the author prior to joining the St. Jude Medical. The IP belong to the author.

1. Introduction

Modern implantable medical devices could be viewed as complex data acquisition, processing and communication agents, designed to work for many years, powered by the original battery. Data storage and transmission represents one of the major challenges in designing such devices, requiring dedication of substantial memory and power resources. To meet this challenge, data within the device must be sufficiently compressed. There exists a large body of work dedicated to compression of cardiac signals (see, for example, [1]-[4]). However, designing a compression algorithm that is substantially efficient for an implantable device implementation represents a major challenge. The compression algorithm must be computationally inexpensive while providing high compression ratios and high fidelity of the compressed waveform. In many applications the problem becomes even more challenging due to limitation imposed on the size of the data framed that must be compressed and reliably transmitted. Because of these limitations, the known efficient compression algorithms, such as Huffman and arithmetic encoding [5], can not be directly used [4]. These algorithms require relatively complex statistical computations performed on the signal, and have limited efficiency on small data frames.

The most efficient modern compression approaches are wavelet transform-based. The first wavelet-based algorithm implemented in an implantable device used Haar transform to effectively compress the EGM morphology information [6]. Here we propose a wavelet-based approach to compression, which is optimized for implantable and other low power devices implementation. It requires only very basic arithmetic and logical operations, such as additions, counters and shifts. While being adaptive to the information contained in the waveform, all statistical decisions are based on simple number and bit counters. All information needed for decompression is included with every data frame, and compression is already efficient with frames as small as 16 samples. The decompression algorithm is also very efficient. While different variations of the proposed approach could be used with different types of waveforms, in this paper we report applications of the compression algorithm to ECG signals.

2. Compression algorithm

The flowchart of the compression algorithm is shown in Fig. 1. The waveform data is usually available in a buffer of a certain length, and the samples could be viewed as integers at a certain bit resolution. The data is processed in the following major steps:

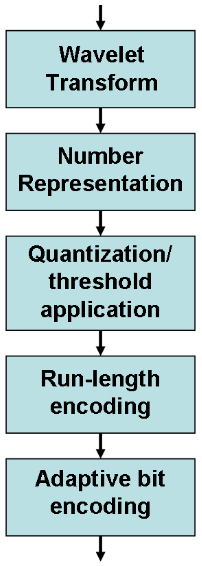

Figure 1. The flowchart of the compression algorithm.

1. Wavelet transformation
2. Change of number representation
3. Quantization coupled with threshold application
4. First stage compression: number level
5. Second stage compression: bit level

Let us describe the steps listed above.

1. The wavelet transform used in this work is an integer to integer transform known as the Integer or Linear Wavelet Transform or CDF(2,2) [5, 7]. In the lifting scheme this transform uses very efficient integer arithmetics: additions, subtractions and bit shifts (divisions by 2 and 4):

$$\begin{aligned} d_k &= x_{2k+1} - (x_k + x_{2k+2})/2, \\ s_k &= x_{2k} + (d_{k-1} + d_k)/4, \end{aligned}$$

where x_k are the original waveform or smooth wavelet coefficients of the current scale, d_k are the detail and s_k are the smooth wavelet coefficients of the next, coarser, scale. Rounding errors of the direct transform are exactly compensated by the rounding errors of the inverse one. Our simulations showed that the Linear Wavelet Transform, while being very computationally inexpensive, provides sufficient smoothness of the compressed waveforms and higher compression ratios than those of the Haar transform [5]. This wavelet transform increases the bit bandwidth of the signal by a single bit. For example, if the signal is sampled with 12 bits resolution, then in the wavelet domain each sample will be represented by 13 bits. Our simulations showed that compression ratios saturate at about 5 to 6 transform stages. This observation allows data frame size to be multiples of 64 or 128 correspondingly, which is a much less restrictive condition than the power of 2 length, and thus allows more flexibility in the system design.

2. So far all data samples, both in the time and transform domains, were assumed to be integers, with negative numbers in the standard twos complement representation. If one assumes that wavelet coefficients could be negative or positive with equal probability of 0.5, and absolute values of the majority of the coefficients are expected to be small, then the counts of one and zero bits in this representation are going to be somewhat similar. In order for bit one to carry information about the smallness of the number in which it is present, the wavelet coefficients are converted into the following representation: The most significant bit represents the sign, which is followed by the bits representing the absolute value of the number. For example, number -5 in 8 bit representation will look as 10000101. By dramatically cutting the probability of bit 1 in the transform domain, this representation facilitates the bit compression algorithm proposed in this work.

3. Quantization and threshold application. Quantization is the process of assigning integer numbers to the values in the wavelet domain. Since the wavelet transform used in this work is already integer, the linear scalar quantization is trivial: no action taken. It becomes non-trivial if a threshold is applied to the wavelet coefficients, which effectively performs the mapping of integers to integers. Another method known as dead-zone quantization, which is also coupled with the threshold application, might provide some additional computational and compression efficiency. With this method the least significant bits of the signal in the transform domain which are below the threshold are set to zero, effectively reducing the dynamic range of the signal. To reduce distortion due to compression losses, different thresholds are used in different wavelet subbands. In this work, for greater computational efficiency, the thresholds are controlled by a single parameter: the threshold in the finest scale wavelet domain. In each adjacent domain containing the wavelet coefficients of the next time scale, the corresponding threshold is chosen to be half of the current one. If we denote the threshold of a wavelet subband j as th_j, then the threshold in the next subband is given by the expression:

$$\mathrm{th}_{j+1} = \mathrm{th}_j/2.$$

We have chosen the threshold in the subband corresponding to the finest scale to be a power of 2, so that thresholds in all other subbands are also powers of 2, making computation and application of such thresholds very efficient, as all comparisons happen at bit boundaries of wavelet coefficients.

4. The next two steps of the algorithm perform compression of the information contained in the quantized (threshold applied) wavelet coefficients. The first compression stage is the standard run-length encoding of numbers [5]. In order to further increase the compression ratio, the encoding is modified by replacing single isolated zeros with ones. This substitution significantly increases compression ratio without any noticeable deterioration of the signal, since after application of a threshold that is greater than one, the one and zero levels in the signal become indistinguishable.

5. The second compression stage is performed on bit strings corresponding to the same bit levels across the data frame which is being compressed. The only statistical measure of such bit strings used in this work is the probability of bit one in it. Each bit string is adaptively run-length encoded, with the length of the field containing the number of consecutive zeros based on the probability of 1 in the string. The advantage of the chosen number representation becomes evident at this stage: since most numbers are expected to have small absolute values, this representation insures that ones are sparse and on most levels the bit strings are mostly comprised of zeros, thus enabling efficient run-length encoding. The length of the field encoding the number of consecutive zeros is very critical to the code efficiency and is determined for each string according to the following formula:

$$k = \text{ceil}(\log_2(1/p_0)),$$

where k is the length of the encoding filed, p_0 is the frequency of zeros, and the ceil() function denotes the nearest integers greater than or equal to its argument. This function could be efficiently implemented as a look up table, based on bit counts in the string. The value of k is stored in a fixed-length preamble to the encoded bit string. For example, a 4-bit preamble can encode the maximal number of 16, meaning that 16 bits can be available to encode the number of zero bits, which potentially could be used to efficiently encode rather long strings. The length of the preamble depends on the size of the data frame and is added to the length of the compressed bit string, creating an additional overhead and reducing the compression efficiency for short strings.

To illustrate this encoding, let us consider a toy example of a frame consisting of 12 samples, each containing 5 bits. Let us say they are $0, -3, 8, -1, -2, 0, 7, 3, -2, 0, -6, 1$, then in the number representation used, we have:

0	:	00000
-3	:	10011
8	:	01000
-1	:	10001
-2	:	10010
0	:	00000
7	:	00111
3	:	00011
-2	:	10010
0	:	00000
-6	:	10110
1	:	00001

The following bit strings will have to be encoded:

010110001010, containing all most significant bits
001000000000
000000100010
010010111010
010100110001, containing all least significant bits

The first string has about the same number of ones as zeros, so the bit run-length encoding will not be applied. In the second string the number of zeros is sufficient to use the encoding, provided one allocates 4 bits for the encoding field. The second string is encoded as follows:

$$00010101001,$$

which is one bit shorter than the original string. A fixed length preamble (in this case 3 bits containing 100) with information about the length of the zero encoding field has to be added to each string, however. Simulations show that compression is efficient on data frames starting at length of 16.

The described run-length encoding of bit strings is supplemented with another method, utilizing adaptive use of fixed look up tables. The look up tables map combinations of fixed numbers of bits into variable length bit fields, which is similar to using Huffman encoding trees. For each bit string a fixed encoding table is used. The particular table used with a bit string is determined by the frequency of ones p_1 in that string. In our approach no statistical information about different bit combinations is measured, and probabilities of all sequences containing the same numbers of ones are considered the same. The following bit probabilities for using different tables were determined in simulations. If the frequency of ones is $0.21 < p_1 \leq 0.29$, then the following table is used:

$$[00] \to [0];\ [10] \to [10]; [01] \to [110]; [11] \to [111].$$

If $0.16 < p_1 \leq 0.21$, then each three consecutive bits are encoded in a similar manner using the table containing 8 entries , and if $0.06 < p_1 \leq 0.16$, each four bits are encoded using the table containing 16 entries. If $p_1 \leq 0.06$

then the adaptive bit run-length encoding described above is used. The frequency ranges provided here were found to be optimal in numerical simulations. The compression method used is indicated in the preamble to each bit string. The only statistical measure required in this procedure is the bit count, so that no complex symbol statistical information is computed. The compression ratio is saturated by just three encoding tables for three distinct values of probabilities of ones. The total memory required by all tables is just 223 bits. This approach resulted in slightly greater overall compression ratios for bit strings than using just the adaptive bit run-length encoding, while having similar computational efficiency.

3. Results

A bit-accurate simulation of the new compression algorithm was implemented in Matlab and applied to ECG data taken from the Physionet compression test database. The ECG data was sampled at 250 Hz rate and encoded with 12 bit resolution. The algorithm performance was measured on 40 frames from 4 different patient records.

The compressed waveform distortion was defined as the Percent Root-mean-square Difference, PRD $= 100\% * \left(\sum_{i=1}^{n}(x_i - \widetilde{x}_i)^2 / \sum_{i=1}^{n} x_i^2\right)^{1/2}$, where x_i are samples of the original waveform, $\widetilde{x}_i$ are samples of the compressed waveform, and n is the number of samples in the data frame. With PRD kept under 8%, the compression ratios, defined as ratios of total numbers of bits in the original and compressed waveforms, were 9.3 ± 2.5, consistently exceeding 85% of the theoretical limit determined by the bit entropy of the original data frames. Example frames with the original and compressed waveforms are shown in Fig. 2. These examples show the high fidelity of the compressed waveform and demonstrate the dependence of compression ratios on the content of the frames.

4. Discussion and conclusions

The new wavelet-based compression algorithm is computationally inexpensive, while being versatile and efficient, especially when applied to relatively small data frames, which lack symbol statistics needed for effective encoding by other methods. All information needed for decompression is included with every data frame. Bit accurate simulations showed that the algorithm achieves compression ratios close to those of it's more sophisticated and considerably more computationally expensive counterparts. The algorithm could be used in various applications in both microprocessor and ASIC implementations, helping to reduce memory requirements and the battery energy spent on the information transmission to and from the implantable or other low power device.

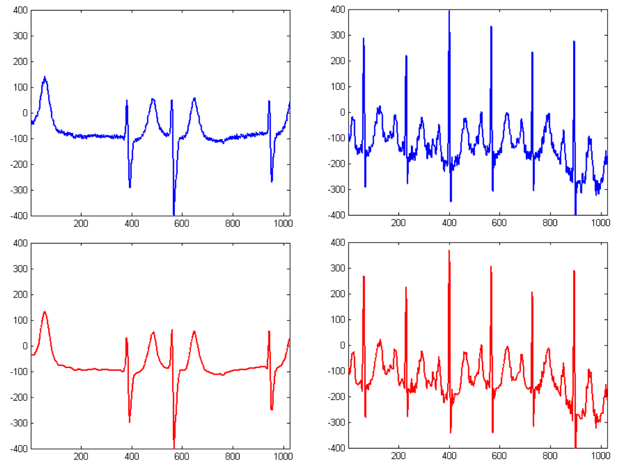

Figure 2. The original (above) and compressed (below) ECG frames sampled at 12 bits, 250 samples per second. Combined adaptive fixed-table and bit run-length encoding was used. Left: Rec No: 08730-02, compression ratio = 13.1, PRD = 6.1%. Right: Rec No: 11442-01, compression ratio = 5.13, PRD = 6.4%.

Acknowledgements

I would like to thank my family and friends for all their support during the work on this project.

References

[1] K Nagarajan, E Kresch. Constrained ECG Compression Using Best Adapted Wavelet Packet Bases. IEEE Signal Processing Letters, 1996;3:273-275

[2] A. Molina, A. Urbaszek, J. Huber, M. Schaldach. A novel, low-complexity method for intracardiac signal compression in implantable devices. Proceedings of the 19th Annual International Conference of the IEEE.1997;30:95-96

[3] P Rossi, A Casaleggio, M Chiappalone, M Morando, G Corbucci, M Reggiani, G Sartori, E Borgo. Low Complexity Methods for Intracardiac Atrial Electrogram Compression. Computers in Cardiology 2000;27:565-568.

[4] HL Chan, YC Siao, SW Chen, SF Yu. Wavelet-based ECG compression by bit-field preserving and running length encoding. Computer Methods and Programs in Biomedicine. 2008;90:1-8 .

[5] David Salomon. Data Compression. Third edition. Springer, 2004.

[6] LA Koyrakh, JM Gillberg, NM Wood. Wavelet Transform Based Algorithms for EGM Morphology Discrimination for Implantable ICDs. Computers in Cardiology 1999;26:343-346.

[7] Wim Sweldens, Peter Schröder. Building your own wavelets at home. In: Wavelets in the Geosciences. Springer, 2000.

Address for correspondence:

Lev Koyrakh
4595 Forestview Lane North
Plymouth, MN 55442
lev.koyrakh@gmail.com