Bufferless Compression of Asynchronously Sampled ECG Signals in Cubic Hermitian Vector Space

T. Marisa*, T. Niederhauser, A. Haeberlin, R. A. Wildhaber, R. Vogel, M. Jacomet, and J. Goette

***Abstract*—Asynchronous level crossing sampling analog-to-digital converters (ADCs) are known to be more energy efficient and produce fewer samples than their equidistantly sampling counterparts. However, as the required threshold voltage is lowered, the number of samples and, in turn, the data rate and the energy consumed by the overall system increases. In this paper, we present a cubic Hermitian vector-based technique for online compression of asynchronously sampled electrocardiogram signals. The proposed method is computationally efficient data compression. The algorithm has complexity $O(n)$, thus well suited for asynchronous ADCs. Our algorithm requires no data buffering, maintaining the energy advantage of asynchronous ADCs. The proposed method of compression has a compression ratio of up to 90% with achievable percentage root-mean-square difference ratios as a low as 0.97. The algorithm preserves the superior feature-to-feature timing accuracy of asynchronously sampled signals. These advantages are achieved in a computationally efficient manner since algorithm boundary parameters for the signals are extracted *a priori*.**

***Index Terms*—Asynchronous sampling, buffer-less compression, cubic Hermitian basis.**

I. Introduction

ASYNCHRONOUS analog-to-digital converters (ADCs) in electrocardiogram (ECG) signal acquisition reduce power and mean sampling rate as compared to constant sampling rate ADCs [1], [2].[1] Asynchronous ADCs attain these advantages by offering signal-dependent sampling, which avoids capturing samples when there is low-to-no-signal activity. However, as the required threshold voltage is lowered, the number of samples and, in turn, the energy consumed by the overall system increases [3]. There have been attempts to reduce the amount of data generated by these asynchronous ADCs. These attempts thus far have mostly been methods that achieve data reduction by varying the level crossing threshold voltage during flat and steep signal slopes [4], [5]. These methods are directly coupled to a particular ADC design and, thus, offer power advantages. The idea with these methods is to increase the threshold voltage of the ADC when the signal is changing, thus reduce number of samples in these areas. These attempts have shown great potential. However, the direct coupling of these methods with the ADC architecture means that the methods are hardware architecture specific. Furthermore, the strategy can easily present itself as a limitation in situations where the most interesting signal features are in these areas of high-signal activity, because there all the interesting features will be acquired with lower quality than the less important portions of the signal.

We propose a portable compression methodology for asynchronous sampling that attempts to solve this dilemma. Our efforts in developing our compression methodology have not been in isolation. There have indeed been several others. These developments have seen different kinds of methods; these can be broadly classified into direct and transformation data compression methods [6]. Examples of direct methods are in [6]–[12]. Examples of transformation compression methods are in [8], [13]–[22]. The algorithms in both classes offer different advantages. Direct methods are typically lightweight with a lower computational burden than the transform methods [7], [9], [11]. Transform methods, on the other hand, offer better compression ratios (CRs) and reconstructed signal quality [20]. However, most of the developments in both direct and transform-based methods have targeted synchronous equidistant sampling. Our compression algorithm targets asynchronously sampled signals. The proposed method may be classified as a transformation compression method. The proposed approach shares a common attribute with compressed sensing [23]: even though compressed sensing approaches can be considered as transformation-based approaches, compressed sensing is not computationally intensive for compression since compressed sensing systems are designed to incoherently sample data in a sparse transform domain. All the computational intensive work is carried out during reconstruction. In our approach, compression is carried out by acquiring the signal in cubic Hermitian vector space, without any computationally intensive transformation task, as in wavelets or other transformation methods [18], [20], [24]. This advantage results since the major step in the transformation is done by the level crossing ADC. However, unlike compressed sensing, we do not rely on incoherent sampling; indeed, level crossing ADCs are signal dependent. Furthermore, since we capture the signal in cubic Hermitian vector domain and reconstruct using

Manuscript received January 3, 2015; revised June 12, 2015; accepted June 16, 2015. Date of publication June 25, 2015; date of current version November 20, 2015. This work was supported in part by the Commission for Technology and Innovation CTI (10717.1-PFLS-LS), Switzerland. *Asterisk indicates corresponding author*.

*T. Marisa is with the Institute for Human Centered Engineering—microLab, Department of Engineering and Information Technology, Bern University of Applied Sciences, CH 2501 Biel, Switzerland, and also with the ARTORG Center for Biomedical Engineering Research, University of Bern, 3012 Bern, Switzerland (e-mail: thanks.marisa@bfh.ch).

T. Niederhauser is with the Institute for Human Centered Engineering—microLab, Department of Engineering and Information Technology, Bern University of Applied Sciences, and also with the ARTORG Center for Biomedical Engineering Research, University of Bern.

A. Haeberlin is with the ARTORG Center for Biomedical Engineering Research, University of Bern, and also with the Department of Cardiology, Inselspital, Bern University Hospital.

R. A. Wildhaber, M. Jacomet, and J. Goette are with the Institute for Human Centered Engineering—microLab, Department of Engineering and Information Technology, Bern University of Applied Sciences.

R. Vogel was with the University of Bern. He is now with the Department of Cardiology, Bürgerspital Solothurn.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TBME.2015.2449901

[1]Note that in asynchronous ADC, there is no sampling rate since the samples are not equidistant.

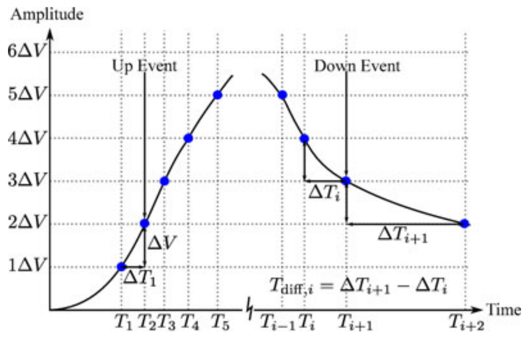

Fig. 1. Level crossing sampling.

cubic Hermitian interpolation, the reconstruction is less computationally intensive than in compressed sensing. The method is highly suited for asynchronous signal acquisition systems since:

1) The algorithm's complexity is $O(n)$ and compresses the samples as they arrive from the asynchronous ADC maintaining the signal activity dependence advantage. Therefore, the algorithm also does not require signal buffering memory. This is shown by the algorithm in Section IV.
2) The inputs to the algorithm are time change information and signal events that are directly generated by the asynchronous ADC. This is discussed in more detail in Section II.
3) A cubic Hermitian vector is a 4-D vector with two position and two derivative axis, and both these axis pairs are simply obtained from the asynchronous ADC output. This is discussed in Section III.
4) Using the cubic Hermitian basis for compression means that we also have efficient reconstruction methods available [25]–[27].

In this paper, we explain the development of the compression method. In Section II, we analyze level crossing ADC sampled data that are generated from simulations based on all the data records in the MIT-BIH arrhythmia database [28]. Section III discusses the relationship between asynchronously sampled data and cubic Hermitian basis data transformation and representation. In Section IV, we present a Monte–Carlo simulation model, which we used to extract system parameters. Section V presents the performance analysis and hardware verification. We draw conclusions in Section VI.

II. Asynchronously Sampled Data Analysis

Level crossing ADCs asynchronously transform signals into high accurate time information [1], [2]. Fig. 1 shows an illustration of asynchronous sampling: the ΔTs are the change in time between two level crossing events, and $T_{\mathrm{diff},i}$ is the difference between two consecutive ΔTs.

A. Data Analysis

The aim of our data analysis is to investigate the statistical characteristics of asynchronously sampled data. Based on the analysis, we develop a compression algorithm that exploits the data characteristics. The data are generated by asynchronously resampling all the signals from the MIT-BIH arrhythmia database. The asynchronous ADC has a 10-bit timer. We performed resampling for ΔV resolutions of 4.0, 5.0, 6.0, and 7.0 bits per millivolt.[2] The selected resolution range is common for asynchronous ADCs [2], [3], [29], [30].

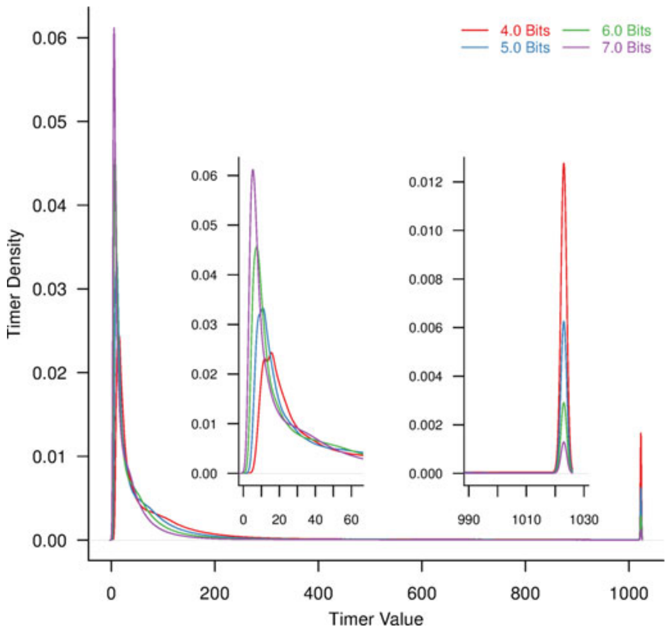

Fig. 2. Timer values probability density.

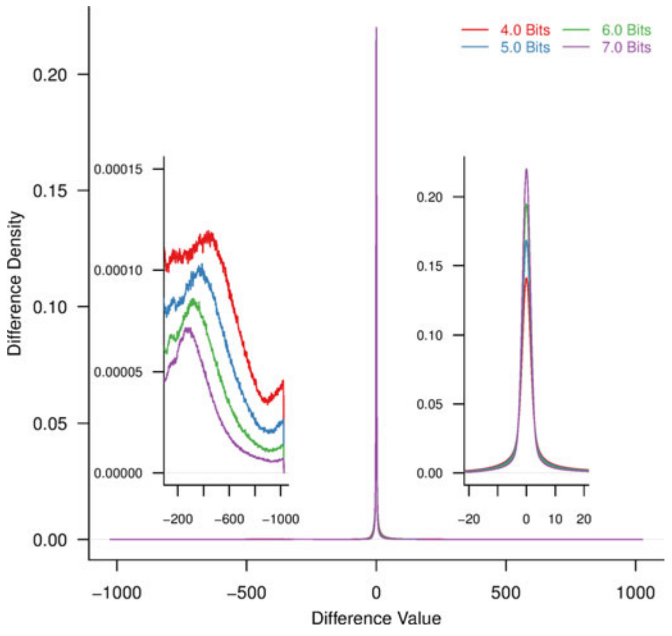

Fig. 3. Timer difference values probability density.

Figs. 2 and 3 show the probability density (mass) functions of the timer values and differences, respectively.

Fig. 2 shows the distribution of the timer values for four different levels of ΔV. The plot also shows two peaks; the

[2]The full range of the signals in the MIT-BIH database is ± 10 mV.

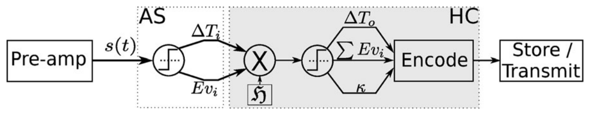

Fig. 4. System block diagram. AS is the level crossing asynchronous ADC and HC is the Hermitian coder.

first and larger peak accounts for most of the samples. Thus, most of the samples are small timer values. The second peak is at timer count 1023, which is the overflow value for our 10-bit timer. Fig. 3 shows the distribution of the timer difference values for consecutive samples in the sequence. The plot shows the timer differences for four different levels of ΔV. The plot has one peak in the neighborhood of timer value zero. This shows that most of the differences between consecutive samples are also small. Small timer values and small differences in timer values tell us that within the observed time and ΔV scales, there is a smooth transition in the signal's amplitude and their first derivative with low values for first and second derivatives. With the information gathered this far, we move toward our goal of designing a computationally efficient compression method with minimal data buffering.

III. Signal Transformation and Representation

Our goal is to efficiently reduce the amount of data captured while ensuring that the important diagnostic information contained in the signal is maintained. The level crossing ADC captures the changes in time ΔT it takes for the signal to make a predefined change ΔV. These values can be used to infer the first derivative within the ΔT time interval as follows:

$$dV_{\text{in}}/dt \approx \Delta V/\Delta T\ . \tag{1}$$

Since ΔV is fixed, and our intention is not to have an accurate value of the point derivative but just an estimate over a short time period for inference purposes, we may use ΔT to infer the derivative. Fig. 4 gives the block diagram showing the essentials of our algorithm in the shaded area. Our algorithm is the HC block. The algorithm converts and encodes acquired time and event information into more compact cubic Hermitian basis representation. Ev_i and ΔT_i are the event type (up or down crossing), and timer value information generated by the level crossing asynchronous ADC, AS block. ΔT_o, $\sum Ev_i$, and κ describe our Hermitian segment. ΔT_o is the initial ΔT value of the Hermitian represented signal section, $\sum Ev_i$ is the accumulated event count, and κ is the accumulative effect of higher derivatives in the time interval. The algorithm can be pictorially represented by a projection part, which converts the signal events into the cubic Hermitian basis representation, the thresholding part, which compacts the vectors, and the encoding part, which packs the data for storage or transmission.

A. Cubic Hermitian Representation

The cubic Hermitian basis $H(t) = \{\phi_h(t)\}$ is made up of four orthogonal functions. A polynomial (segment) $f(t)$ can be approximated in cubic Hermitian basis domain by

$$\tilde{f}(t) = w_0\phi_{00}(t) + w_1\phi_{01}(t) + w_2\phi_{10}(t) + w_3\phi_{11}(t) \tag{2}$$

where w_i are components of the Hermitian geometry vector associated with the function $f(t)$. The subscripts on the Hermitian basis functions have been specifically chosen to represent the orthogonality property of the functions; orthogonality is important since it offers computational simplicity and efficiency [31]. We can extract the geometry vector from a monotonic segment of $f(t)$ as follows: with no loss of generality, we can normalize the time internal of $f(t)$ to $t \in [0, 1]$, we have at the left end of $f(t)$, $f(0) = P_0$ and $f^{'}(0) = T_0$, and for the right end $f(1) = P_1$ and $f^{'}(1) = T_1$; here, P_0 and P_1 are the amplitudes, and T_0 and T_1 are the derivatives at these ends. $(P_0,\ P_1,\ T_0,\ T_1)$ is the Hermitian geometry vector which we have extracted from the function $f(t)$. The cubic Hermitian basis polynomials in vector form are

$$\begin{pmatrix} \phi_{00}(t) \\ \phi_{01}(t) \\ \phi_{10}(t) \\ \phi_{11}(t) \end{pmatrix} = \begin{pmatrix} (2t^3 - 3t^2 + 1) \\ (-2t^3 + 3t^2) \\ (t^3 - 2t^2 + t) \\ (t^3 - t^2) \end{pmatrix} \hat{=}\ \boldsymbol{\phi}(t)\ . \tag{3}$$

We can represent an approximation of $f(t)$ as $\tilde{f}(t)$ based on its associated geometry vector as

$$\tilde{f}(t) = (P_0,\ P_1,\ T_0,\ T_1)\,\boldsymbol{\phi}(t). \tag{4}$$

If we generalize our time interval to $t \in [t_o, t_n]$, then the instantaneous error for this approximation is given by

$$|f(t) - \tilde{f}(t)| = \frac{f^{''''}(c)}{384}(t - t_o)(t - t_o)(t - t_n)(t - t_n) \tag{5}$$

where $c \in [t_o, t_n]$. This formula is important since it tells us that the error can be regulated by relating the fourth derivative ($f^{''''}(c)$) to the interval length. The error can be kept small if we keep $[t_o, t_n]$ small. The error is zero on $t = t_o$ and $t = t_n$. The error bounds are given by

$$||f(t) - \tilde{f}(t)|| \leqslant \frac{||f^{''''}||_\infty}{384}\, d^4 \tag{6}$$

where $d = t_n - t_o$, and the norm is the infinity norm [32].

Having laid down the basics of cubic Hermitian vector representation, we now connect this to the data coming out of the asynchronous ADC and show how the data are handled and assembled by our algorithm into Hermitian geometry vectors.

B. Time to Geometry Vector

Our algorithm constructs Hermitian geometry vectors from data generated by the asynchronous ADC. Equation (4) shows how a signal segment is represented in cubic Hermitian vector space by geometry vector $(P_0,\ P_1,\ T_0,\ T_1)$. In cubic Hermitian space, the signal is split into segments and each of these segments is represented by a 4-D Hermitian vector. Fig. 5 shows a function/signal segment f, which can be represented in the

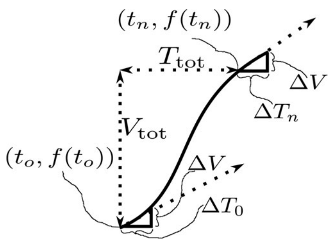

Fig. 5. Hermitian representation.

Hermitian basis as

$$\tilde{f}(t) = f(t_o)\phi_{00}(t) + f(t_n)\phi_{01}(t) + \left(\frac{\Delta V}{\Delta T_o}\right)\phi_{10}(t) + \left(\frac{\Delta V}{\Delta T_n}\right)\phi_{11}(t)\,. \tag{7}$$

In Fig. 5, $V_{\text{tot}} = (n+1)\Delta V$, an integer multiple of ΔV, and T_{tot} is a time interval composed of the sum of all ΔTs from the asynchronous ADC during the interval. Since we know that ΔV is fixed, we can represent the two tangent-based coordinates using only the values of ΔT_0 and ΔT_n. The function f has points whose coordinates $(t_i, f(t_i))$ can be represented by

$$f(t_n) = \sum_{i=0}^{n} Ev_i * \Delta V\,, \quad \text{with } t_n = \sum_{i=0}^{n} \Delta T_i\ . \tag{8}$$

Here, we let the event-type operator Ev_i have values $\{-1, 0, +1\}$, where -1 represents a signal decreasing event, 0 represents a timer overflow event, and $+1$ represents a signal increasing event. The simplest events to handle for our algorithm are the timer overflow events. Timer overflows are merely time-keeping events; thus, we can add the timer overflows to the ΔT of the next signal increase or decrease event in the series of events without introducing any error in the signal representation. This operation turns Ev_i in (8) into a sign operator with values $\{-1, +1\}$. Since ΔV is fixed and known for the asynchronous ADC, we can determine amplitude values $f(t_n)$ by simple event counts as shown in (9). Since Ev_i is a sign operator, we can fully represent the signal with a series of ΔT_is with a single bit for event type (sign):

$$f(t_n) = \Delta V \sum_{i=0}^{n} Ev_i\ . \tag{9}$$

If we replace the tangent axis with asynchronous ADC generated data, (4) becomes

$$\tilde{f}(t) = \sum_{i=0}^{1} P_i\phi_{0i}(t) + \sum_{i=0}^{1} \frac{\Delta V}{\Delta T_i}\phi_{1i}(t)\,. \tag{10}$$

By expressing P_0 and P_1 in terms of function values according to (9), we obtain

$$\tilde{f}(t) = \phi_{00}(t)P_0 + \phi_{01}(t)\left(P_0 + \Delta V \sum_{i=0}^{n} Ev_i\right) + \Delta V\left(\phi_{10}(t)\frac{Ev_0}{\Delta T_0} + \phi_{11}(t)\frac{Ev_n}{\Delta T_n}\right) \tag{11}$$

where P_0 is the left-end amplitude of the previous interpolation piece. If we let $P_0 = m\Delta V$, where m is the cumulative count of level crossing events up to the current interpolation segment, and restrict interpolation segments to derivatives of the same sign, to meet monotonicity requirements, we restrict interpolation segments to derivatives of the same sign; then, (11) simplifies to

$$\tilde{f}(t) = \Delta V\left(m\phi_{00}(t) + (m \pm (n+1))\,\phi_{01}(t) + \frac{\phi_{10}(t)}{\Delta T_0} + \frac{\phi_{11}(t)}{\Delta T_n}\right)\,. \tag{12}$$

Since the event type Ev is a sign operator, we have the $\pm$ in (12). The Hermitian geometry vector is given by

$$\left(m, (m \pm (n+1)), \frac{1}{\Delta T_0}, \frac{1}{\Delta T_n}\right)\,. \tag{13}$$

We do not yet have information on the time interval length. Therefore, we introduce the variable κ defined as

$$\kappa = \sum_{i=1}^{n} (\Delta T_i - \Delta T_o)\,. \tag{14}$$

The time interval is determined as

$$T_{\text{tot}} = (n+1)\Delta T_o + \kappa\,. \tag{15}$$

We have thus far shown how a signal segment can be represented by a Hermitian geometry vector, and also how the geometry vector can be represented by the variables n, ΔT_o, ΔT_n, and κ.

IV. Algorithm Design and Implementation

We built a data compressor that maintains the activity-dependent low-energy advantages of asynchronous sampling systems. Such an algorithm should:

1) have low working memory requirements. This keeps static energy requirements low.
2) have an order of performance at most linear $O(n)$. This ensures that the compression algorithm can be driven by the signal events from the asynchronous ADC [33].

A. Modeling and Parameter Estimation

To achieve the above requirements, our algorithm must be adaptive. It dynamically makes decisions on the error level by observing data generated by the asynchronous ADC and use the information to infer boundary conditions for Hermitian geometry vector construction. The error bound can be found by evaluating (6). However, calculation of reliable values of the infinity norm of the fourth-order derivative, $||f''''||_\infty$, given the nonuniform sampling grid, is computationally intensive [34], and we may not have enough data points for the calculation during periods of low signal activity. To overcome these challenges, we developed an error inference method that does not rely on the evaluation of (6).

In Fig. 5, T_{tot} is the Hermitian geometry vector time interval, $T_{\text{tot}} = t_n - t_o = d$; with (6) and (15), this gives

$$||f(t) - \tilde{f}(t)|| \leqslant \frac{||f''''||_\infty}{384}(n\Delta T_o + \kappa)^4. \tag{16}$$

Equation (16) shows us that for a function $f(t)$ in the interval $t \in [t_o, t_n]$, we can estimate the relative change in the error bounds of cubic Hermitian vector representation by observing the variables n, ΔT_o, and κ, since $||f^{''''}||_\infty$ is a constant in the interval $[t_o, t_n]$. Furthermore, since we know that at $t = t_o$ and $t = t_n$, the instantaneous error is zero according to (5), we can control the error in the interval $t \in [t_o, t_n]$ by tracking n, ΔT_o, and κ throughout the interval. Therefore, we need to define the parameters for decision making and their boundaries. Our findings in Section II guide us in developing a decision making strategy. Figs. 2 and 3 give the following.

1) Most of the timer values generated are small. This suggests that most of our samples are generated in a burst.
2) The time differences between consecutive events are small, meaning that most of our signal segments have small second derivatives.
3) There is a smooth transition from low to high timer values and their associated differences. This means that we can break the timer value space into zones and be guaranteed of a smooth transition between the zones.
4) These findings are more pronounced on lower ΔV values and are consistent on all the simulated levels.

From the exploratory data analysis and signal transformation equations, we need to develop a strategy for packing the related sample values into geometry vectors. We introduce another variable τ that defines the ΔT values allowed within a Hermitian geometry vector. The variable τ appears as

$$\kappa = \sum_{i=1}^{n} (\Delta T_i - \Delta T_o) \qquad \text{for all } |\Delta T_i - \Delta T_o| < \tau. \tag{17}$$

The variable τ prohibits sudden large jumps in ΔT values. In order to simplify the implementation, we let τ, κ, and n have dyadic boundaries. The boundaries of τ span the timer bit-width B. For example, if we let the $Q\text{th}$ boundary of τ be τ_Q, then $Q\text{th}$ boundary of τ be τ_Q, then

$$\tau_Q = \tau_1 \cdot 2^{Q-1}, \quad \max \tau_Q < 2^B \tag{18}$$

where τ_1 is the first τ boundary. For each τ boundary, there are corresponding boundaries for κ and n, κ_Q and η_Q, respectively:

$$\kappa_Q = \kappa_1 \cdot 2^{Q-1}, \quad \max \kappa_Q < 2^{B+1} \tag{19}$$

where κ_1 is the first κ boundary

$$\eta_Q = \eta_1 \cdot 2^{-(Q-1)}, \quad \eta_Q > 0 \tag{20}$$

with η_1 being the first η boundary.

We use a Monte–Carlo simulation to find the boundaries of our compression-steering parameters from large amounts of data. We executed 1000 Monte–Carlo runs while sweeping the system variables τ, κ, and n. Monte–Carlo simulations require a large pool of ECG signals with characteristics covering a wide range. To generate these signals, we use ECGSYN [35]. For ECGSYN to be useful for our Monte–Carlo simulations, we need to have underling distributions for its parameters that cover a wide range of heart rate variations and noise conditions. Our Monte–Carlo model generates the parameters for ECGSYN by sampling the distributions shown in Table I. In Table I, RSA means Respiratory Sinus Arrhythmia.

TABLE I
ECGSYN SIGNAL-SYNTHESIZER PARAMETERS*

ECGSYN Parameter	Source/Value
Resolution	Floating point
Sample rate (f_{ecg}) [Hz]	2 000 000
Heart rate mean (h_{mean}) [beats/min]	Weibull
Low-frequency RSA (f_L) [Hz]	Uniform
High-frequency RSA (f_H) [Hz]	Uniform
Amplitude of additive uniform noise (A)	Gaussian

*The high sampling frequency of 2 MHz is used to emulate an analog signal that is asynchronously sampled by an asynchronous ADC model.

For various heart rate variations, our Monte–Carlo simulation samples an inverse function based on a Weibull distribution, which is nonzero only for nonnegative values [36]. According to [36], the parameter settings cover ECG signals for pediatrics, young adults, and the elderly. It also covers effects of signal phenomena such as RSA [36]. The low-frequency f_L and high-frequency f_H for RSA are sampled from a uniform distribution with values in (0.04–0.15 Hz) and (0.15–0.4 Hz) for low- and high-frequency bands, respectively. These variables also model artifacts like baseline wander. The noise level is sampled from a Gaussian distribution with zero mean and 0.1-mV standard deviation.

We present Monte–Carlo simulation results for two widely used performance metrics, namely the percentage root-mean-square difference (PRD) [6], and CR, with relation to our system variables, ΔV in bits/mV, $\log_2(\tau)$, $\log_2(\kappa)$, and $\log_2(\eta)$.[3] Interpretation of the PRD measure from a medical point of view has already been discussed in [37]. The CR is defined as

$$\text{CR} = \frac{\texttt{bitrateO} - \texttt{bitrateC}}{\texttt{bitrateO}} \times 100 \tag{21}$$

where `bitrateO` and `bitrateC` represent the number of bits required for the original and compressed signals, respectively. For this paper, unless stated otherwise, the value of `bitrateO` is taken from the bit rate used in the MIT-BIH arrhythmia database, calculated based on a sample rate of 360 Hz by 11 bits per sample. The MIT-BIH bit-rate definition was used to facilitate comparison with other works.

Figs. 6–9 show the relationship of our system variables κ, τ, and η with respect to PRD and compression at ΔV level of 5 and 7 bits. The variables are simply base-2 logarithms of the first boundaries shown as τL, κL, and ηL. As expected, we notice that both PRD and CR levels are higher for higher values of κ. We show that there is an improvement in PRD and a reduction in CR when the ΔV resolution is increased. However, the decrease in the CR rate shown for the high resolution may be misleading since the CR rate with reference to the data coming out of the asynchronous ADC is higher than the one presented

[3]The timer frequency variable has weak influence on performance for timer frequency values greater than 6 kHz. For our experiments, timer frequency is at 16 kHz.

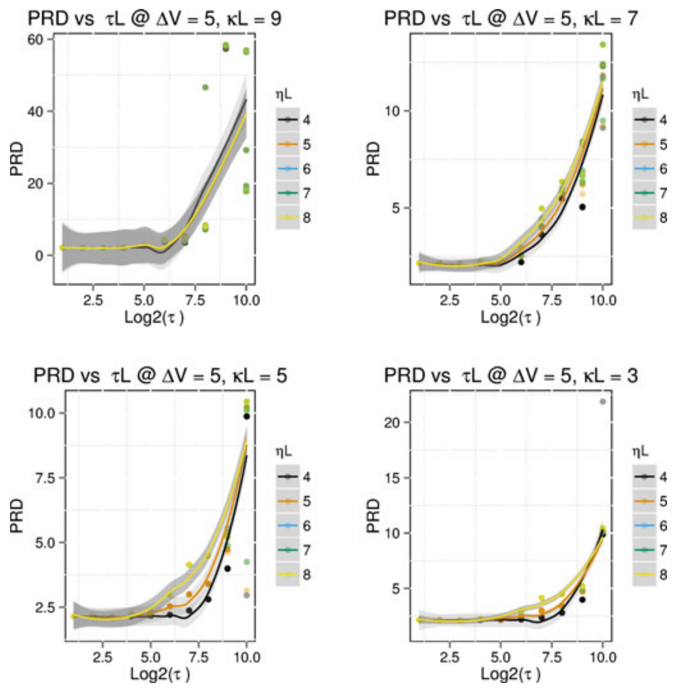

Fig. 6. Signal PRD versus $\log_2(\tau)$ for each value of ΔV level.

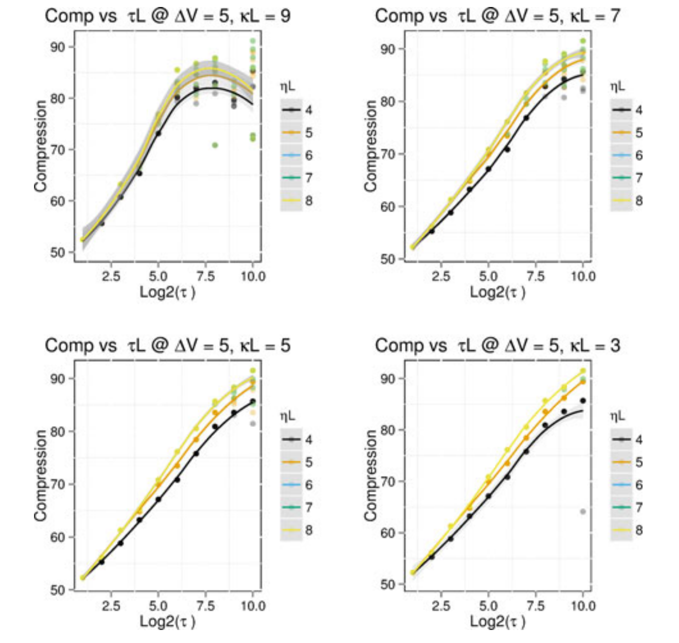

Fig. 7. Signal compression versus $\log_2(\tau)$ for each value of ΔV level.

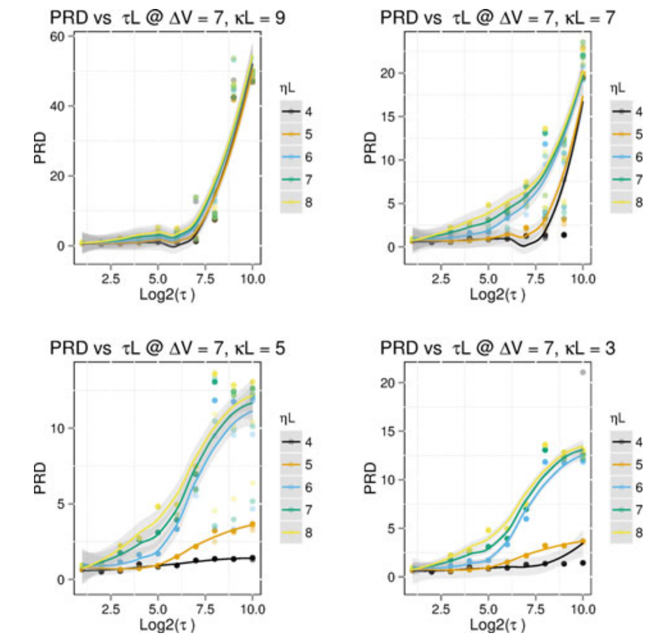

Fig. 8. Signal compression versus $\log_2(\tau)$ for each value of ΔV level.

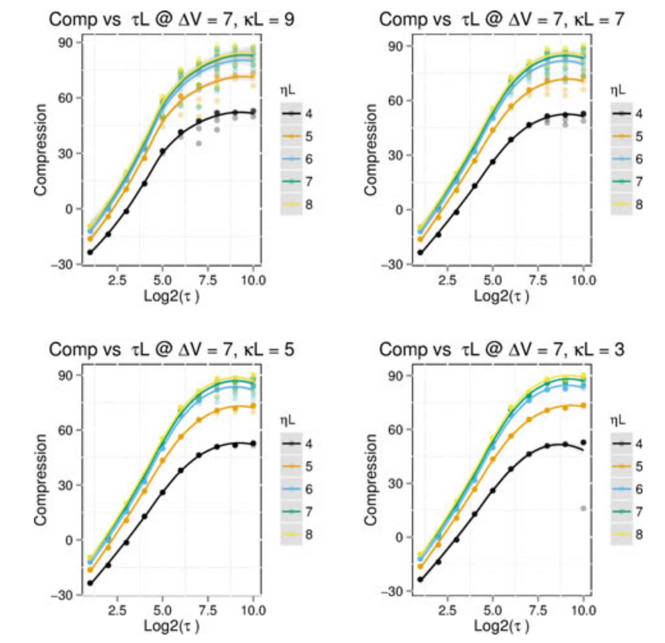

Fig. 9. Signal compression versus $\log_2(\tau)$ for each value of ΔV level.

here, which is calculated with reference to the original classical data to enable readers to compare with other works. We also see that at ΔV resolution of 5 bits, the event count boundary ηL has less effect on the PRD than ΔV resolution of 7 bits. This is because event counts are low at this low ΔV resolution; thus, event count boundaries are not breached. We see that the PRD has a higher variance for high values of κL and ηL. This is because high values of κL and ηL mean that we are reducing the rate of response of our algorithm to changes in the incoming samples, thereby allowing larger variances in our performance metrics. Furthermore, higher values of κL allow larger deviation of subsequent values of ΔT from the initial value. From the figures, we are able to choose boundary values for our algorithm and use these boundaries in the implementation.

B. Implementation

The algorithm adaptively constructs cubic Hermitian geometry vectors from a sequence of time intervals and events. The adaptive construction of compact cubic Hermitian geometry vectors reduces the data, while ensuring that the error remains sufficiently small. This operation is used to assimilate samples of little significance (those with little to no new information on the signal) as determined by the initial timer value ΔT_o, segment event counter n, zone parameter τ, and accumulative effect of higher derivatives in the time interval parameter κ. If the rate of new information is low, the algorithm combines the incoming vector with the existing one by addition. It then checks κ. If κ is below the threshold, the algorithm continues to construct the Hermitian geometry vector from the events and ΔTs as they are generated by the asynchronous ADC. This means that for rising/falling slopes in the signal, where asynchronous ADCs generate a burst of samples, but where changes per sample are small, each consecutive sample contains redundant information when viewed from the Hermitan basis perspective as in (12). Hence, most of the events and ΔTs generated will be compressed into one Hermitian geometry vector. All we have to store or transmit is the ΔT_o, n, and κ. Here is the resulting compression algorithm, which uses two auxiliary functions, `StoreHV()` and `CheckBoundaries()`, and which we name `AsyncHermitian`:

```
StoreHV($\Delta T_0, \Delta T_i, Ev_0, Ev_i, n, \kappa_i$)
   **if** $n == 0$ **then**
      store($\Delta T_0, Ev_0$)
   **else**
      store($T_0, Ev_0, n, \kappa_i$)
   $\Delta T_0 \leftarrow \Delta T_i,\ Ev_0 \leftarrow Ev_i,\ n \leftarrow 0,\ \kappa_i \leftarrow 0$
CheckBoundaries($\Delta T_0, \Delta T_i, Ev_0, Ev_i, n, \kappa_i$)
   get $Q$ from $\Delta T_0$
   **if** $(Ev_i \neq Ev_0)$ **or** $\left((\Delta T_i - \Delta T_0) \notin [\tau_{Q-1}, \tau_Q)\right)$ **or**
   $(\kappa_i \geq \kappa_Q)$ **or** $(n \geq \eta_Q)$ **then**
      StoreHV($\Delta T_0, \Delta T_i, Ev_0, Ev_i, n, \kappa_i$)
   **else**
      $n \leftarrow n + 1,\ Ev_0 \leftarrow Ev_i,\ \kappa_i \leftarrow \kappa_i + (\Delta T_i - \Delta T_0)$
AsyncHermitian
   $n \leftarrow 0,\ \kappa_i \leftarrow 0,\ T_i \leftarrow 0$
   **while true do**
      **wait until** ($\Delta V$ event **or** $\Delta T_i$ timer overflow)[4] **do**
      capture asynchronous ADC values: $(\Delta T_i,\ Ev_i)$
      **if** $n = 0$ **then**
         $\Delta T_0 \leftarrow \Delta T_i,\ Ev_0 \leftarrow Ev_i,\ \kappa_i \leftarrow 0$
      CheckBoundaries($\Delta T_0, T_i, Ev_0, Ev_i, n, \kappa_i$)
```

The presented algorithm has been programmed in software in procedural form for simulations and implemented in two forms, asynchronous and synchronous logic, on field-programmable gate array (FPGA) hardware. Since the maximum event rate is sufficiently low for ECG signals [3], there is no need for parallelization on the signal processing level in our hardware implementation.

C. Encoding

In order to ensure buffer-less compression suitable for asynchronous ADCs, we developed a simple method of packing

[4]For timer overflow, the captured value from the ADC is $\Delta T_i = 0$.

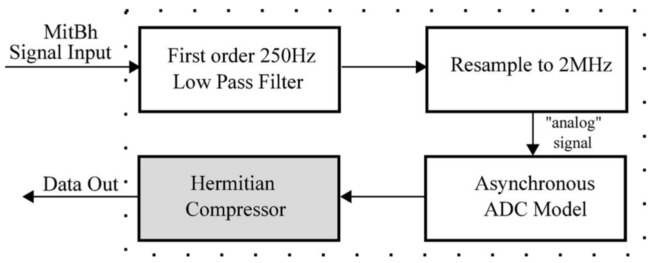

Fig. 10. Test-bench layout algorithm evaluation.

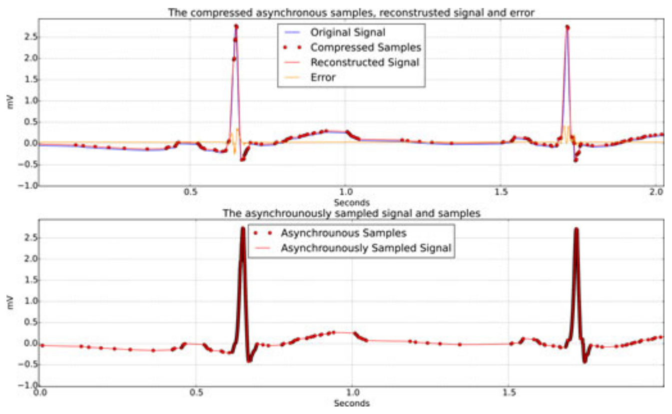

Fig. 11. Compressed samples and asynchronous ADC samples for an ECG signal.

the bits for storage or transmission. Our algorithm gives out four types of data chunks; thus, we use 2 bits to represent the data chunk-type: increment (01), decrement (10), timer overflow (11), compact vector (00). For example, an increasing signal geometry vector would be represented by a code 01 followed by the ΔT_o, then the n, and then κ. Since the n value does not fill the full bit width, we do not need extra bits for the 00 code. An uncompressed increment event that did not fit any geometry vector would simply be represented by a 01 followed by the ΔT_i; value and overflow values are simply represented by 11 followed by the count of overflows since these also occur in bursts in regions of low activity. Huffman encoding or other methods can still be used on top of the simple encoding proposed to gain additional compression at the expense of needing buffering memory and Huffman tree storage memory.

V. Results

We evaluated our algorithm based on all the signals in MIT-BIH arrhythmia database. We also did an FPGA-based implementation of our algorithm. We tested the implementation with a real ECG signal acquired with an asynchronous ADC application specific integrated circuit ASIC. Fig. 10 shows the layout of our test bench for using the MIT-BIH arrhythmia database. In the signal flow block diagram shown in Fig. 10, the signal from the MIT-BIH arrhythmia database is passed through a low-pass 250-Hz filter to band-limit the signal [28]. The signal is then resampled to 2 MHz. This step is necessary since this signal is taken to the asynchronous ADC model that requires input

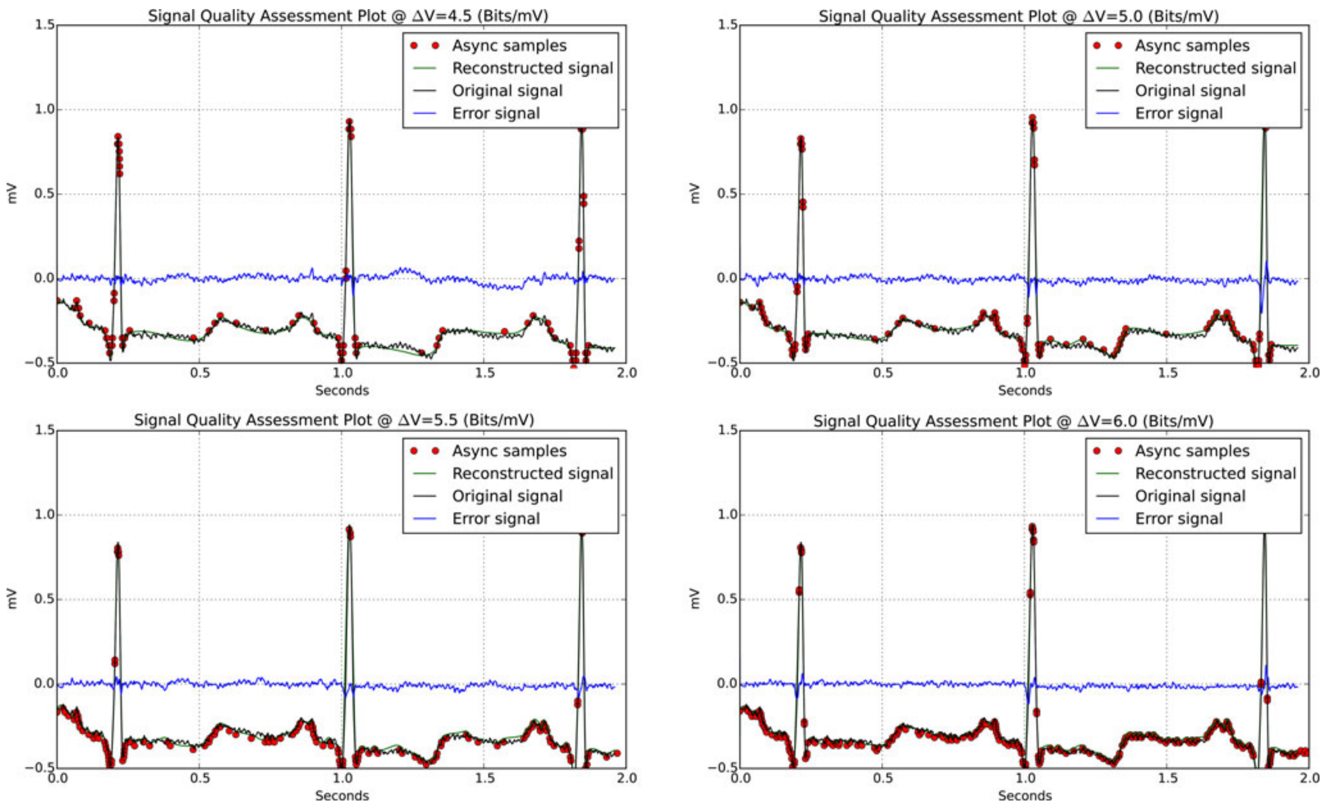

Fig. 12. Signal reconstruction quality assessment.

signals with a very fine time axis (representative of an analog signal).

A. Hardware Implementation

We have implemented our algorithm on 500k gates Xilinx Spartan3E FPGA. We implemented the algorithm in two unbuffered forms: a synchronous version clocked at 32 kHz (32 768 Hz), and an asynchronous version fully driven by the events from the asynchronous ADC. For both implementations, the time measurement resolution is $1/32\,768$ s. For practical reasons, we chose the above 32-kHz clock for running the synchronous implementation of the algorithm to match the time measurement resolution. We tested the implementation with an asynchronous ADC ASIC. The ΔV resolution of the asynchronous ADC is 5 bits/mV. The test signal was an ECG signal band limited to 250 Hz. The number of sample events from the asynchronous ADC doubles with every 1 bit increase in ΔV [3]. Therefore, to keep the synchronous implementation of the algorithm working, the clock rate should be doubled with every 1-bit resolution increase of the asynchronous ADC.

In terms of compression and signal quality, both the synchronous (clocked) and asynchronous versions are identical. However, the asynchronous implementation is 30% more efficient in hardware-resource resource usage and also offers the advantage of not requiring a clock that has to be set based on the input signal and the ΔV. Fig. 11 shows the performance of our algorithm on a real ECG signal captured with an asynchronous ADC and then compressed by our algorithm. The top plot in the figure shows the compressed samples, with ΔV resolution of 5 bits/mV, $\tau L = 6$, $\eta L = 5$, and $\kappa L = 5$, reconstructed signal, and error signal with resulting performance of PRD $= 2.6$ and CR $= 79$. The bottom plot in the figure shows the asynchronously sampled signal with no compression.

B. Discussion

Figs. 6–9 show that our algorithm is robust and can be used at different threshold resolutions. In Fig. 9, we notice that the CR is negative for low values of the first τ boundary. This is because the asynchronous ADC generates a higher number of samples than the original equidistant sample rate on which the compression calculations are based. However, as the τ boundary is increased, the CR rate increases allowing designers to use asynchronous sampling ADCs in applications which would require a high ΔV resolution. These particular applications are normally not good candidates for asynchronous sampling due to the high amount of data generated at low ΔV values.

For asynchronous ADCs, the number of samples increases with increasing ΔV resolution. This phenomenon is inherent in all asynchronous ADCs. This effect is worsened when the ΔV value approaches the noise floor. Fig. 12 shows a signal segment from the MIT-BIH arrhythmia database showing our algorithm's results for different ΔV resolutions. Fig. 12 shows that at a resolution of 6 bits, our asynchronous ADC is approaching the noise floor, hence increasing the number of samples even in regions of low activity. Most of these samples that are close to the noise floor are not compressed by our algorithm since it thinks that

TABLE II
PERFORMANCE COMPARISON FOR DIFFERENT COMPRESSION METHODS

Parameter	This work	CSBL[22]	CSBP [21]	WDB10 [21]
CR%	50–85	20–65	20–74	75–86
PRD%	0.97–9.61	— #	1.58–9.0	1.1–7.52
Memory Bytes	16	6500*	6500	4600
Additions	312–420	768–5888	5888*	11 272
Multiplications	0	0	0	11 784

The # means the paper [22] gives mean square errors only. We can, however, assume the values are close to those in [21]. The * means that the value was derived from the related work.

these samples have valuable information. In an implementation where there is no need for buffer-less compression, Huffman encoding can be carried out on the Hermitian packed vectors. Our simulations indicate that this would result in less than 10% increase in compression.

We compare the performance results of our compression method to wavelet compressed equidistantly sampled signals and to compressed sensing techniques; Table II presents the results.

Table II shows that our method required less computational effort compared to wavelets and digital compressed sensing approaches. The table also shows that our method also requires much lower working memory with eight (16 bit) registers. These are not for data buffering but rather for holding our parameters and state bits. Our method also achieves high compression rates.

VI. CONCLUSION

We have proposed a buffer-less and noncomputationally intensive compression method for asynchronous level crossing ADCs. Our approach offers dynamic error tracking by watching the accumulative effect of first and higher derivatives of the signal. This ability to dynamically monitor the error level enables our approach to offer signal compression with reconstructed-signal quality and CRs comparable to more computationally intensive methods such as wavelets-based approaches [24]. Our compression and performance results are compared to others in Table II. The buffer-less operation is beneficial for asynchronous sampling systems and allows efficient implementations. Additionally, hardware implementations would lead to low-area and low-power implementations. We also shown that the algorithm can operate in conditions of noise even though there is an increase in the number of samples. We went further to implement and verify the algorithm on hardware in two configurations a synchronous version with a low clock of 32 kHz (32 768 Hz) and an asynchronous version which we found to be 30% more hardware-resource efficient. For both implementations, the time measurement resolution is $1/32\,768$ s.

We have presented our algorithm for the compression of ECG signals. However, the algorithm can be used for other signals by adapting the parameters to suit the characteristics of signal families in question. Other possible signals include accelerometer signals and other biomedical signals such as EEGs.

REFERENCES

[1] N. Sayiner *et al.*, “A level-crossing sampling scheme for A/D conversion,” *IEEE Trans. Circuits Syst. II, Analog Digital Signal Process.*, vol. 43, no. 4, pp. 335–339, Apr. 1996.

[2] E. Allier *et al.*, “A new class of asynchronous A/D converters based on time quantization,” in *Proc. 9th Int. Symp. Asynchronous Circuits Syst.*, May 2003, pp. 196–205.

[3] Y. Li *et al.*, “A sub-microwatt asynchronous level-crossing ADC for biomedical applications,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 7, no. 2, pp. 149–157, Apr. 2013.

[4] M. Kurchuk and Y. Tsividis, “Signal-dependent variable-resolution quantization for continuous-time digital signal processing,” in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2009, pp. 1109–1112.

[5] M. Trakimas and S. Sonkusale, “An adaptive resolution asynchronous ADC architecture for data compression in energy constrained sensing applications,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 5, pp. 921–934, May 2011.

[6] S. M. Jalaleddine *et al.*, “ ECG data compression techniques—A unified approach,” *IEEE Trans. Biomed. Eng.*, vol. 37, no. 4, pp. 329–343, Apr. 1990.

[7] J. R. Cox *et al.*, “ AZTEC, a preprocessing program for real-time ECG rhythm analysis,” *IEEE Trans. Biomed. Eng.*, vol. 15, no. 2, pp. 128–129, Apr. 1968.

[8] N. Ahmed *et al.*, “Electrocardiographic data compression via orthogonal transforms,” *IEEE Trans. Biomed. Eng.*, vol. BME-22, no. 6, pp. 484–487, Nov. 1975.

[9] W. C. Mueller, “Arrhythmia detection program for an ambulatory ECG monitor,” *Biomed. Sci. Instrum.*, vol. 14, pp. 81–85, Apr. 1978.

[10] J. P. Abenstein and W. J. Tompkins, “A new data-reduction algorithm for real-time ECG analysis,” *IEEE Trans. Biomed. Eng.*, vol. BME-29, no. 1, pp. 43–48, Jan. 1982.

[11] S. C. Tai, “SLOPE—A real-time ECG data compressor,” *Med. Biol. Eng. Comput.*, vol. 29, no. 2, pp. 175–179, Mar. 1991.

[12] S. C. Tai, “ECG data compression by corner detection,” *Med. Biol. Eng. Comput.*, vol. 30, no. 6, pp. 584–590, Nov. 1992.

[13] B. R. Reddy and I. S. Murthy, “ECG data compression using fourier descriptors,” *IEEE Trans. Biomed. Eng.*, vol. 33, no. 4, pp. 428–434, Apr. 1986.

[14] H. A. al Nashash, “ECG data compression using adaptive fourier coefficients estimation,” *Med. Eng. Phys.*, vol. 16, no. 1, pp. 62–66, Jan. 1994.

[15] H. A. al Nashash, “A dynamic fourier series for the compression of ECG using FFT and adaptive coefficient estimation,” *Med. Eng. Phys.*, vol. 17, no. 3, pp. 197–203, Apr. 1995.

[16] Z. Lu *et al.*, “Wavelet compression of ECG signals by the set partitioning in hierarchical trees algorithm,” *IEEE Trans. Biomed. Eng.*, vol. 47, no. 7, pp. 849–856, Jul. 2000.

[17] M. Kyoso and A. Uchiyama, “ECG data reduction method for medical telemetry systems,” *Frontiers Med. Biological Eng.: Int. J. Japan Soc. Med. Electron. Biol. Eng.*, vol. 11, no. 2, pp. 131–152, 2001.

[18] S.-G. Miaou and S.-N. Chao, “Wavelet-based lossy-to-lossless ECG compression in a unified vector quantization framework,” *IEEE Trans. Biomed. Eng.*, vol. 52, no. 3, pp. 539–543, Mar. 2005.

[19] G. Tohumoglu and K. E. Sezgin, “ECG signal compression by multi-iteration EZW coding for different wavelets and thresholds,” *Comput. Biol. Med.*, vol. 37, no. 2, pp. 173–182, Feb. 2007.

[20] H.-L. Chan *et al.*, “Wavelet-based ECG compression by bit-field preserving and running length encoding,” *Comput. Methods Programs Biomed.*, vol. 90, no. 1, pp. 1–8, Apr. 2008.

[21] H. Mamaghanian *et al.*, “Compressed sensing for real-time energy-efficient ECG compression on wireless body sensor nodes,” *IEEE Trans. Biomed. Eng.*, vol. 58, no. 9, pp. 2456–2466, Sep. 2011.

[22] Z. Zhang *et al.*, “Compressed sensing for energy-efficient wireless telemonitoring of noninvasive fetal ECG via block sparse Bayesian learning,” *IEEE Trans. Biomed. Eng.*, vol. 60, no. 2, pp. 300–309, Feb. 2013.

[23] E. J. Candès *et al.*, “Stable signal recovery from incomplete and inaccurate measurements,” *Commun. Pure Appl. Math.*, vol. 59, no. 8, pp. 1207–1223, 2006.

[24] J. Chen and S. Itoh, “A wavelet transform-based Ecg compression method guaranteeing desired signal quality,” *IEEE Trans. Biomed. Eng.*, vol. 45, no. 12, pp. 1414–1419, Dec. 1998.

[25] F. N. Fritsch and R. E. Carlson, “Monotone piecewise cubic interpolation,” *SIAM J. Numerical Anal.*, vol. 17, no. 2, pp. 238–246, 1980.

[26] A. Ahmadian *et al.*, "An efficient piecewise modeling of ECG signals based on Hermitian basis functions," in *Proc. 29th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, 2007, pp. 3180–3183.

[27] R. Gopalikrishnan and D. H. Mugler, "The evolution of Hermite transform in biomedical applications," in *Intelligent Medical Technologies and Biomedical Engineering: Tools and Applications.*, Hershey, PA, USA: IGI Global, 2010, pp. 260–278.

[28] A. L. Goldberger *et al.*, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, Jun. 13, 2000.

[29] M. Trakimas and S. Sonkusale, "A 0.8 V asynchronous ADC for energy constrained sensing applications," in *Proc. IEEE Custom Integrated Circuits Conf.*, Sep. 2008, pp. 173–176.

[30] K. Kozmin *et al.*, "Level-crossing ADC performance evaluation toward ultrasound application," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 8, pp. 1708–1719, Aug. 2009.

[31] T. Chihara, *An Introduction to Orthogonal Polynomials*. New York, NY, USA: Gordon and Breach, 1978.

[32] C. Moler, *Numerical Computing with MATLAB*. Philadelphia, PA, USA: SIAM, 2004.

[33] Y. P. Tsividis, "Event-driven data acquisition and digital signal processing—A tutorial," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 8, pp. 577–581, Aug. 2010.

[34] X. Z. Ratnesh, K. Shukla, "Derivation of high-order compact finite difference schemes for non-uniform grid using polynomial interpolation," *J. Comput. Phys.*, vol. 204, no. 2, pp. 404–429, 2005.

[35] P. E. McSharry and G. D. Cifford, "Open-source software for generating electrocardiogram signals," *arXiv:physics/0406017*, Jun. 2004.

[36] S. J. Mandrekar *et al.*, "Statistical modelling of the differences between successive RR intervals," *Statist. Med.*, vol. 24, no. 3, pp. 437–451, Feb. 2005.

[37] Y. Zigel *et al.*, "The weighted diagnostic distortion (WDD) measure for ECG signal compression," *IEEE Trans. Biomed. Eng.*, vol. 47, no. 11, pp. 1422–1430, Nov. 2000.

Authors' photographs and biographies not available at the time of publication.