An Energy-Efficient Biomedical Signal Processing Platform

Joyce Kwong, Anantha P. Chandrakasan
Massachusetts Institute of Technology, Cambridge, MA USA
email: {jyskwong, anantha}@mtl.mit.edu

***Abstract*—This paper presents an energy-efficient processing platform for wearable sensor nodes, designed to support diverse biological signals and algorithms. The platform features a 0.5V-1.0V 16b microcontroller, SRAM, and accelerators for biomedical signal processing. Voltage scaling and block-level power gating allow optimizing energy efficiency under applications of varying complexity. Programmable accelerators support numerous usage scenarios and perform signal processing tasks at 133 to 215× lower energy than the general-purpose CPU. When running complete EEG and EKG applications using both CPU and accelerators, the platform achieves 10.2× and 11.5× energy reduction respectively compared to CPU-only implementations.**

I. Introduction and System Overview

Advances in sensor and circuit technologies are fueling new possibilities in medical monitoring, where wearable devices can monitor a subject's vital signs. In such devices, a local processor can extract key information from raw biological signals, thus greatly reducing the amount of data to be transmitted or stored. In this context, a processor offers some advantages over custom ASICs: it can be used across many applications and allows ongoing algorithm improvement. This paper presents a processing platform which supports algorithms of varying complexity in an energy-efficient manner, through voltage scaling, power gating, and hardware accelerators.

The platform pictured in Fig. 1 leverages a 16b CPU based on [1], but here the CPU has been extended with logic to support software debugging as well as a direct memory access (DMA) block. In contrast to [1], [2], we further integrate custom accelerators to reduce energy of biomedical signal processing. The platform also includes a custom SRAM, timers, serial ports, and a hardware multiplier. The CPU initializes these modules by writing to their register interfaces, while the modules drive an interrupt or DMA trigger at task completion. To accommodate different performance demands, the system is voltage-scalable from 1V down to 0.5V. Moreover, each module can be dynamically clock- and power-gated when not in use. Power gating of 15 domains is achieved with on-chip, high-V_t switches controlled by the power management unit.

II. Accelerators for Biomedical Applications

A survey of biomedical signal processing algorithms in literature reveals several common operations that can benefit from hardware acceleration. Filtering is a prevalent task since noise in the acquired signals must be removed. The Fast Fourier Transform (FFT) is employed to analyze the frequency content of various physiologic signals, for example in [3]. In addition, many applications make use of adaptive thresholds, which often require the largest or smallest sample in a window of data. Similarly, median filtering, which is helpful for removing noise spikes without degrading signal edges, involves finding

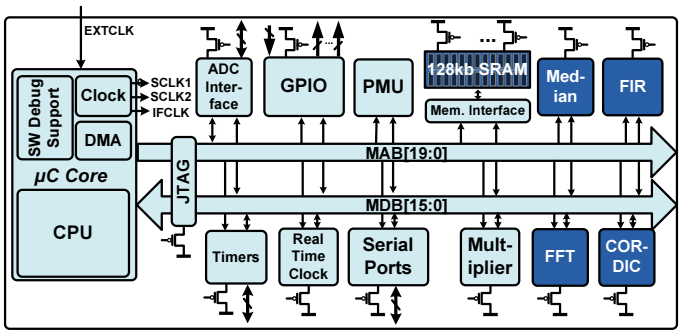


Fig. 1. Block diagram of signal processing platform. Darkly shaded blocks are custom-designed for biomedical applications and low-voltage operation.

the median of a window of data. Many algorithms utilize basic mathematical functions such as e^x [4] and $\sqrt{x}$ [5] which typically involve expensive software emulation in fixed point microcontrollers. To address these needs, the platform includes four hardware accelerators: FFT, CORDIC, FIR filter, and median filter.

Many systems employ dedicated hardware to speed up specific tasks; it has also been shown (e.g. [6]) that accelerators can reduce energy in sensor processors. Measurements reported in Sec. IV show that accelerators can reduce the energy of biomedical signal processing by two orders of magnitude. We will now describe the accelerator architectures and discuss techniques to lower power and extend usability.

A. FFT with Switching Activity Reduction

The FFT accelerator pictured in Fig. 2(a) centers around a radix-2 butterfly datapath, supported by a dedicated SRAM, a lookup table to store twiddle factors (constants used in the FFT), and control logic. The radix-2 serial architecture is widely used and our implementation is based on [6]. However, here we propose a control technique to reduce datapath power.

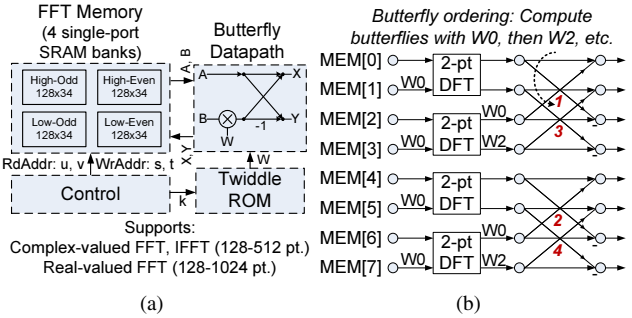


Fig. 2. (a) FFT accelerator architecture. (b) The proposed FFT butterfly ordering to reduce switching activity.

In this serial architecture, the datapath computes one butterfly per clock cycle, and hence the control logic must manage

a sequence of butterflies to complete an FFT. Various control schemes have been proposed (e.g. [7], [8]), but here we focus on reducing the datapath switching activity and power. Specifically, we present a scheme that orders the butterflies according to two constraints: 1) butterflies with the same twiddle factor are performed consecutively, thus reducing datapath switching activity, and 2) two reads and two writes occur on four different SRAM banks in each clock cycle, allowing the use of simpler single-port SRAM. The ordering is shown conceptually in Fig. 2(b) for an 8-point FFT, but the scheme is easily scalable to different FFT sizes.

Details of the control scheme are as follows. Let u, v be the two addresses being read from memory, k the twiddle factor address, and j a butterfly counter going from $0..\frac{N}{2}-1$. In the first $i = 0..(n-2)$ iterations of an N-point FFT ($n = log_2 N$), they can be generated by:

$$m = \{j[n-2:1], 0\} \quad (1)$$

$$u = \{\mathrm{j}[0],\ \mathrm{ROL}_{n-1}(m, i)\} \quad (2)$$

$$v = \{\mathrm{j}[0],\ \mathrm{ROL}_{n-1}(m+1, i)\} \quad (3)$$

$$k = j \text{ with } (\mathrm{n}-1-\mathrm{i}) \text{ LSBs set to } 0 \quad (4)$$

where $j[0]$ indicates bit 0 of j, and $\{a, b\}$ denotes a concatenated with b. $\mathrm{ROL}_{n-1}(a, b)$ involves rotating an $n-1$ bit number, a, by b bits to the left.

In the last iteration ($i = n-1$), u is generated with an $(n-1)$-bit gray code counter, while $v = u$ with MSB set to 1. The above scheme lends itself to a compact circuit realization with two bit-rotators and a gray code counter as shown in Fig. 3. To accommodate different FFT sizes, the bit-rotators are designed to support variable bit widths.

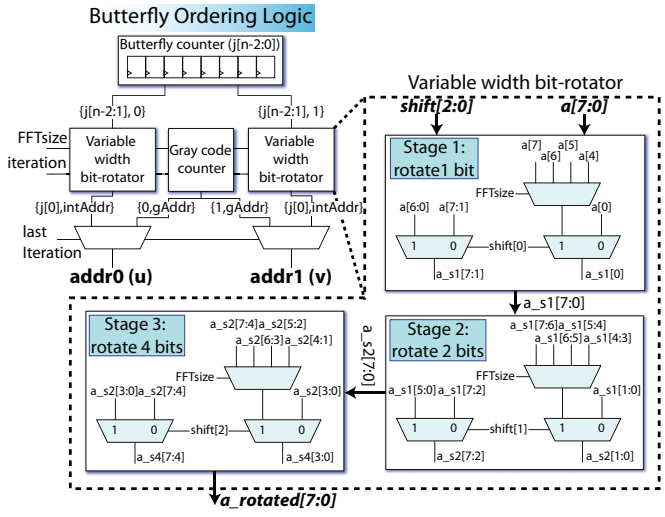


Fig. 3. FFT control logic designed to reduce datapath switching activity.

Fig. 4(a) shows the simulated waveform of the proposed control scheme and a reference design. Switching activity of the twiddle factor (w_r and w_i) is significantly reduced, especially in the early iterations when only several distinct twiddle factors are in use. This is reflected in Fig. 4(b), which compares the simulated power of a reference FFT design with the proposed accelerator, including wiring parasitics extracted from layout. The proposed control scheme reduces the datapath power by 50%, leading to an overall power reduction of 29% in a 128-point complex-valued FFT.

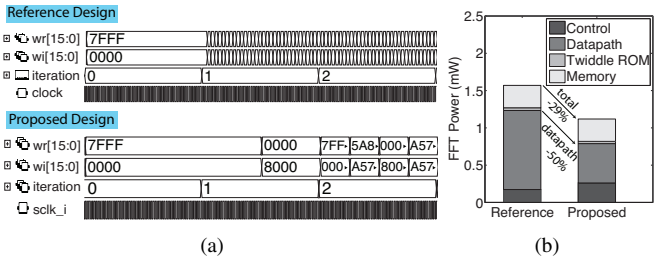


Fig. 4. (a) Simulated waveform of the twiddle factor w_r and w_i in the reference and proposed design. (b) Power comparison (at 1V, 10MHz) of a reference design and the proposed design with switching activity reduction.

B. CORDIC Engine

Biomedical applications employ mathematical functions that must be emulated in software on typical microcontrollers, often requiring several thousand clock cycles to complete. Fortunately, the CORDIC algorithm (COordinate Rotation DIgital Computer) [9] can compute these functions efficiently at relatively low hardware cost. In this work, we modify the classic CORDIC architecture in order to support the wide range of use cases in biomedical applications.

1) Overview: The left portion of Fig. 5 illustrates how $\sin(\alpha)$ and $\cos(\alpha)$ can be computed with CORDIC. We begin with a unit vector v_0 and rotate it successively by θ_i until it makes the desired angle α with the x-axis. Now, by restricting θ_i such that $\tan\theta_i$ equals $\pm 2^{-i}$, the rotation can be realized by shifting and addition. To determine the direction of rotation, an angle accumulator is first initialized with α, then each rotation is taken in the direction that decreases the magnitude of the angle accumulator. Mapping this algorithm onto hardware leads to the classic architecture consisting of the solid shaded blocks in Fig. 5. Although CORDIC is highly efficient, it does involve several shortcomings described below.

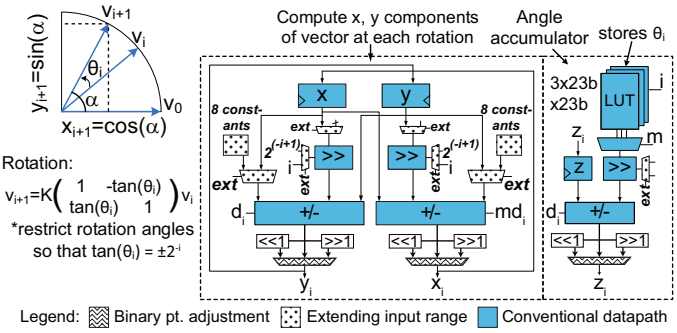


Fig. 5. CORDIC datapath showing blocks in the conventional architecture plus proposed additions to extend input range and reduce quantization error.

2) Extending Input Range: In conventional CORDIC, the range of inputs for which CORDIC gives valid results is extremely limited for certain functions, as reported in the second column of Fig. 6. We propose architectural improvements to implement an extended algorithm described in [10].

For large inputs, several extra rotations are performed to quickly reduce the input down to the range supported by conventional CORDIC. By reusing existing hardware, our architecture achieves this with a few datapath additions as indicated by the dotted blocks in Fig. 5. With this, the input range is now greatly extended to cover the full range in 16-bit fixed point processing. The improvement in $\sqrt{x}$ is critical to reducing the energy of an EKG application discussed in Sec. IV.

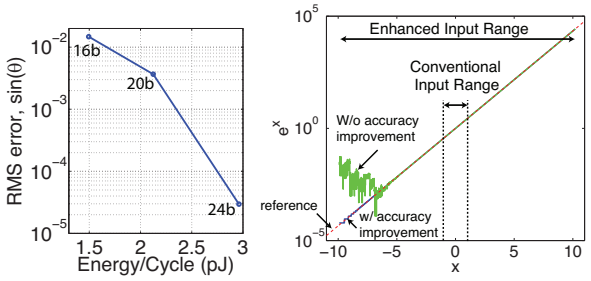
Operation	Conventional CORDIC	Proposed Enhancements
$\sqrt{x}$	$0.0267 \le x \le 2.339$	$0 \le x \le 2^{15}-1$
$\ln x$	$0.1068 \le x \le 9.360$	$2^{-15} \le x \le 2^{15}-1$
e^x	$-1.118 \le x \le 1.118$	$-10.39 \le x \le 10.39$ $(e^{10.39} \approx 2^{15}-1)$

Fig. 6. Convergence range of conventional CORDIC and our implementation. The range has been extended to cover the full range required in 16-bit fixed point processing.

3) Reducing Quantization Error: The CORDIC algorithm contains two main sources of quantization errors from 1) finite precision arithmetic and 2) approximating the desired rotation angle as the sum of elementary angles stored in the lookup table. In this work we improve the accuracy by:

1) widening datapath and increasing number of iterations
2) dynamically adjusting binary point, shifting right for overflow and shifting left to increase precision
3) computing e^x with alternate method

To quantify the impact of these techniques, Fig. 7(a) shows that widening the datapath reduces the RMS error exponentially at the cost of a linear increase in energy. Fig. 7(c) summarizes how the binary point adjustment technique improves the RMS error in different modes of operation. Lastly, straightforward computation of $e^x, x < 0$ results in large inaccuracies, which we remove by performing $e^{|x|}$ then inverting the result, also with CORDIC. The associated input range extension and accuracy improvement are shown in Fig. 7(b).



(a) (b)

Mode of Operation	No adjustment	With adjustment
Circular (sin(θ))	$5.75\text{x}10^{-5}$	$2.95\text{x}10^{-5}$
Hyperbolic (sinh(θ))	0.8136	0.002

(c)

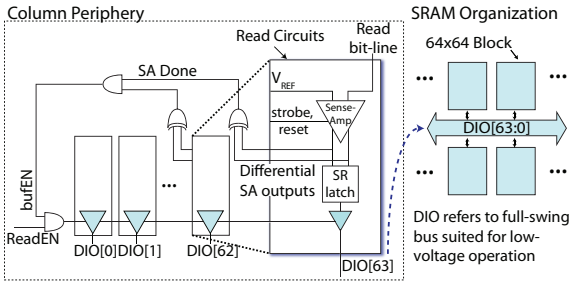
Fig. 7. (a) RMS error in $\sin(\theta), -\pi < \theta < \pi$ vs. CORDIC energy/cycle at different datapath widths. (b) Input range and accuracy improvement in e^x. (c) Improvement in RMS error from binary point adjustment.

C. Accelerators for Filtering

Since filtering is critical to many applications, this platform includes FIR and median filter accelerators. The FIR filter contains one multiply-accumulate unit and employs a synthesized latch-based local memory that reduces overall FIR power by 31% compared to a flip-flop-based memory, in simulation with extracted parasitics. The median filter provides the median, min. and max. of a sliding window of data. The filter keeps a window of data in sorted order; when a new sample arrives, the oldest sample is removed from the window, then the new sample is compared with the window to find the correct position for insertion. While the design in [11] employs 2 comparators per entry in the window, the median filter in this work shares 1 comparator between 4 entries to reduce area.

III. SRAM with Glitch Reduction

This chip employs a 0.5V to 1V 8T SRAM based on [1] as main memory. Here we add a self-timed mechanism to reduce glitches on the data bus connecting 16 SRAM blocks. Although self-timing for this purpose has been shown before, we specifically consider leakage overhead and add minimal logic to prevent most glitches in the average case, rather than attempting to remove all glitches at high leakage cost. During a read, the sense-amplifier (SA) in the column periphery (Fig. 8) compares the read bitline against a reference. The result is then stored in an SR latch and driven onto the data bus (*DIO*) by tri-state drivers. If a new read occurs on a different SRAM block than the previous read, control of *DIO* is transferred to tri-state drivers of the new block. However, if the new drivers are enabled too early, they will first output data previously held in the SR latches before outputting the newly accessed data, causing glitches on the large 64b data bus. To prevent glitches, we can wait until each differential SA has resolved before enabling the tri-state drivers, but adding logic for this at every SA imposes excessive leakage overhead. Instead, we add logic to two SAs and wait until both have resolved before enabling all 64 drivers in the block. Accounting for local variation, on average this prevents glitches on 43 out of 64 bits on *DIO* with much less leakage overhead (see table in Fig. 8).



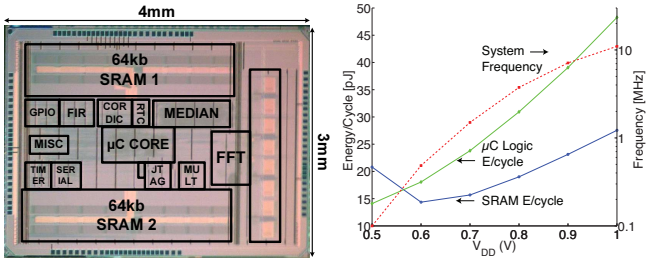
V_{DD}	Straightforward approach		Reduced leakage approach	
	Avg. glitch energy savings	Leakage power overhead	**Avg. glitch energy savings**	**Leakage power overhead**
0.5V	7.0pJ	81.9nW	**4.7pJ**	**2.6nW**
1.0V	28.1pJ	155nW	**18.8pJ**	**4.8nW**

Fig. 8. Low-leakage self-timing scheme to reduce glitching on data bus, with simulated glitch reduction and leakage overhead for a 128kb SRAM.

IV. Measurement and Application Results

The processing platform pictured in Fig. 9(a) was fabricated in a low-leakage $0.13\mu m$ CMOS process. Fig. 9(b) plots the frequency for all blocks and the energy versus V_{DD} for a 64kb SRAM macro and microcontroller (μC) core which

includes the 16b CPU, DMA, and software debug support logic. Since the CPU is compatible with a commercial μC instruction set, we can use the platform to quantify how the accelerators reduce energy for signal processing over conventional microcontrollers. Accordingly, we measure the energy required to perform common tasks 1) using the CPU and hardware multiplier to execute compiled C software and 2) with the accelerators. Silicon measurements in Fig. 10 show that the accelerators provide 133 to 215× energy savings in the listed processing tasks.



(a) (b)

Fig. 9. (a) Die photo of test-chip. (b) Frequency of all components and energy of SRAM and μC core plotted against V_{DD}.

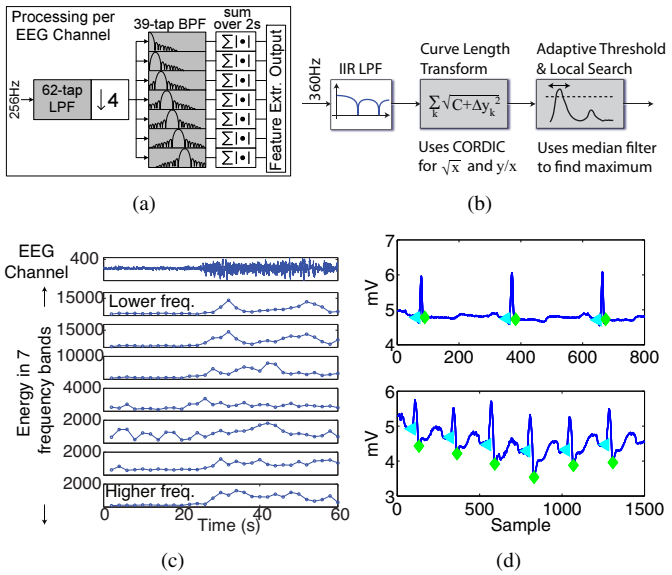
Operation	CPU + Multiplier	Accelerator	Energy Reduction
32-tap FIR	176nJ	1.22nJ	144.4x
512-point FFT	82.1µJ	0.616µJ	133.3x
sin(x)	279nJ	1.30nJ	215.2x
65-point Median	114nJ	0.79nJ	144.9x

Fig. 10. Energy savings afforded by accelerators in executing common signal processing tasks. Measurements performed at 1V.

We illustrate the impact of accelerators in the context of two applications. The first is the feature extraction stage of a machine learning algorithm for real-time epileptic seizure detection [12]. Fig. 11(a) shows the processing on an EEG channel to estimate the energy in 7 frequency bands, via an FIR filter bank followed by magnitude summation; results are then used in a classification stage to detect seizures. Fig. 11(c) plots one EEG input channel and feature extraction results.

The second algorithm finds the onset and duration of the QRS complex in an EKG, which extracts more useful information for medical monitoring than algorithms that find the heart rate only [5]. The algorithm shown in Fig. 11(b) leverages the CORDIC accelerator in computing a curve length transform of the EKG signal, and the median filter in finding the maximum of the transform. Fig. 11(d) shows the test-chip correctly locating the start and end of the QRS complexes in two EKG records from the MIT/BIH Arrhythmia Database.

The platform's versatility allows us to implement these two algorithms from different domains while leveraging the accelerators to save energy. We compare the energy of executing the complete applications 1) solely on the CPU and multiplier versus 2) using accelerators for key signal processing and the CPU for the remaining tasks. Since the accelerated version finishes computation in fewer clock cycles, the platform can operate at a lower V_{DD} while achieving the same latency as the CPU-based version. In Fig. 12, measurements show that the accelerators reduce the total energy to complete the EEG and EKG applications by 10.2× and 11.5× respectively.



(a) (b) (c) (d)

Fig. 11. (a) EEG and (b) EKG applications, with portions that can leverage accelerators shaded in gray. (c) EEG channel and computed energy in 7 frequency bands, (d) Two EKG records with markers showing the computed onset and offset of the QRS complex.

Algorithm	CPU-based version		Accelerated version		Energy reduction
	V_{DD}	Energy	V_{DD}	Energy	
EEG	1.0	(per 512 samples) 198µJ	0.7	(per 512 samples) 19.3µJ	10.2x
EKG	1.0	(per beat) 188µJ	0.7	(per beat) 16.4µJ	11.5x

Fig. 12. Energy savings provided by accelerators in two applications.

ACKNOWLEDGMENT

The authors thank TI for funding and chip fabrication. The authors are grateful to M. Koesler, M. Goel, M. DiRenzo, N. Ickes, and M. E. Sinangil for their valuable support, as well as to A. Shoeb and E. Shih for assistance with the EEG analysis algorithm. J. Kwong is supported by an NSERC fellowship.

REFERENCES

[1] J. Kwong, *et al.*, "A 65nm Sub-V_t microcontroller with integrated SRAM and switched-capacitor DC-DC converter," *ISSCC*, Feb. 2008, pp. 318–319.

[2] S. C. Jocke, *et al.*, "A 2.6-μW sub-threshold mixed-signal ECG SoC," *Symp. on VLSI Circuits*, June 2009, pp. 60–61.

[3] M. Tsuruoka, *et al.*, "Analysis of 1/f fluctuations of heart rate response while walking or listening to sounds," *EMBC*, Aug. 2007, pp. 5915 –5918.

[4] A. Shoeb, *et al.*, "Detecting seizure onset in the ambulatory setting: Demonstrating feasibility," *EMBC*, Jan. 2005, pp. 3546–3550.

[5] W. Zong, *et al.*, "A robust open-source algorithm to detect onset and duration of QRS complexes," *CInC*, Sept. 2003, pp. 737–740.

[6] N. Ickes, *et al.*, "A 10-pJ/instruction, 4-MIPS micropower DSP for sensor applications," *ASSCC*, Nov. 2008, pp. 289–292.

[7] D. Cohen, "Simplified control of FFT hardware," *IEEE Trans. Acoustics, Speech, and Sig. Proc.*, vol. 24, no. 6, pp. 577–579, Dec 1976.

[8] M. Hasan and T. Arslan, "A coefficient memory addressing scheme for VLSI implementation of FFT processors," *IEEE International Symposium on Circuits and Systems*, vol. 4, Aug. 2002, pp. 850–853.

[9] J. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Elect. Comp.*, vol. EC-8, pp. 330–334, Sept. 1959.

[10] X. Hu, *et al.*, "Expanding the range of convergence of the CORDIC algorithm," *IEEE Trans. Comp.*, vol. 40, no. 1, pp. 13–21, Jan. 1991.

[11] S. B. Leeb, *et al.*, "Applications of real-time median filtering with fast digital and analog sorters," *IEEE/ASME Trans. Mech.*, vol. 2, no. 2, pp. 136–143, June 1997.

[12] A. Shoeb, *et al.*, "Patient-specific seizure onset detection," *Epilepsy & Behavior*, vol. 5, no. 4, pp. 483–498, 2004.