A Robust Compressed Sensing IC for Bio-Signals

Jun Kwang Oh

Technical Report No. UCB/EECS-2014-114
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-114.html

May 19, 2014

University of California, Berkeley College of Engineering

MASTER OF ENGINEERING - SPRING 2014

Electrical Engineering and Computer Science

Integrated Circuits

Compressed Sensing ICs for Bio-signals

Junkwang Oh

This **Masters Project Paper** fulfills the Master of Engineering degree requirement.

Approved by:

1. Capstone Project Advisor:

Signature: ___________________________ Date ____________

Print Name/Department: David Allstot / EECS

2. Faculty Committee Member #2:

Signature: ___________________________ Date ____________

Print Name/Department: Jan Rabaey / EECS

A Robust Compressed Sensing IC for Bio-Signals

Master of Engineering Capstone Project

UC Berkeley, Electrical Engineering

Master of Engineering

Jun Kwang Oh

Abstract

Energy-efficient application-specific integrated circuits (ASICs) are necessary in severely energy-constrained sensors. Compressed sensing is a signal-processing algorithm for data compression which exploits the nature of the sparseness in typical bio-signal. However, compressed sensing algorithms are not stable in the presence of noise, so data is often reconstructed with errors. Most existing designs only provide either spike detection or compressed sensing for data compression to achieve energy-efficient ASICs. In this capstone project, we demonstrate a design for combining spike sorting and compressed sensing algorithms in a DSP chip that can perform data compression by achieving accurate data reconstruction with injected noise. The resulting circuit architecture is implemented in a 32nm CMOS process. Clocked at 20 kHz, the total power dissipation is 30.462 uW at 0.78 V for a single channel with a 10bit/sample input stream.

I. INTRODUCTION

Over the past two decades, the cost of the microelectronics has decreased, allowing for relatively cheap wearable sensors. In turn, wearable sensors are highly developed and employed in a broad range of applications from agriculture to health care, where high energy efficiency is essential. For example, the battery lives of implantable medical devices for patients should be long enough to reduce the costs of replacing them. In order to improve the energy efficiency of the system, the overhead by compressing the data cannot outweigh the energy savings from data compression. Thus, compressed sensing is a practical solution for wireless sensors, and an energy efficient hardware implementation must be realized.

In a traditional bio-sensor, analog input signals are sampled based on the Nyquist sampling theorem, which requires that the sampling rate must be twice greater than the maximum frequency of the sampled input signal. However, compressed sensing requires the sampling rate to be based on the information content rather than the frequency content of a signal. In theory, this would enable a far smaller data sampling rate compared to a traditional Nyquist sampling rate [2]. Thus, compressed sensing exploits the low rate of significant events in sparse signals and achieves low-power consumptions for wearable sensors.

To counteract compressed sensing’s natural weakness to noise, which Fig. 1 shows, a spike-sorting algorithm is introduced to filter out noise. Through the spike-sorting algorithm, thresholds of input signals are calculated within the chip and filtered and aligned to a common reference [3]. For data reconstruction, this combination of two algorithms is particularly attractive because it achieves a robust data reconstruction regardless of noise.

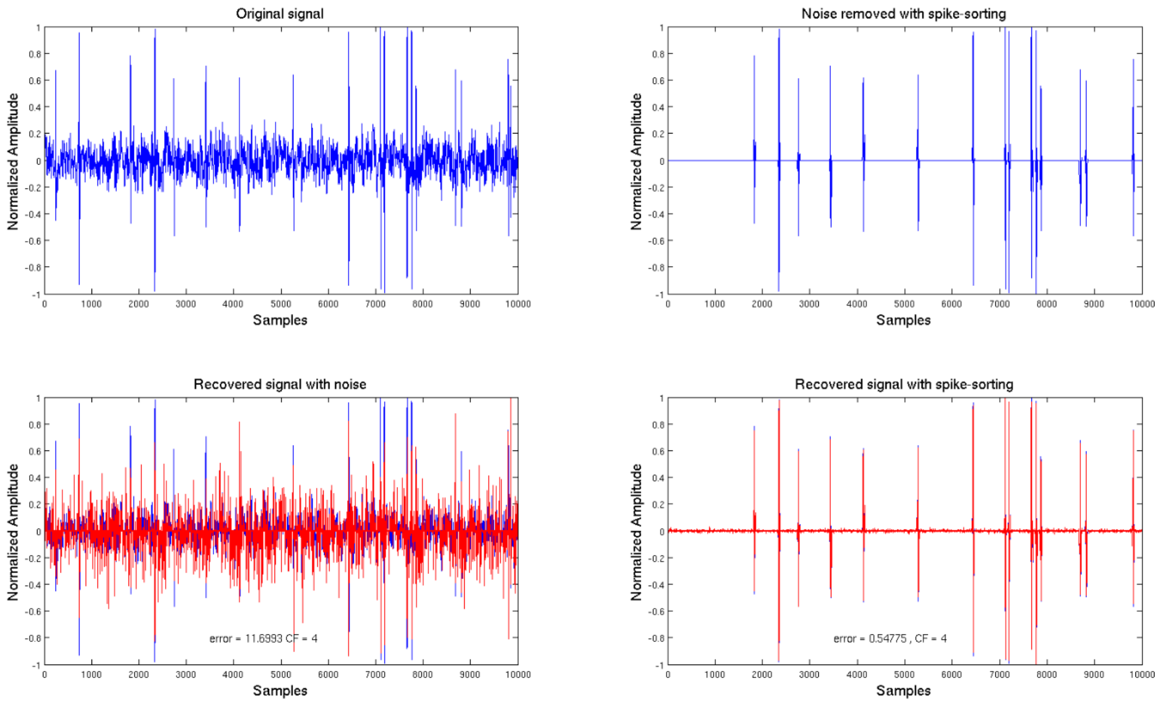


Fig 1. Compressed sensing on bio-signal without (top) and with (bottom) spike-sorting; original signal (blue) and reconstructed signals (red).

Section II of this paper describes the background of the spike-sorting and compressed sensing algorithm. Section III presents the design details of each block and discusses the energy-efficient choices made during the architecture design. Power optimization strategies are described in Section IV. Section V presents measured results. Section VI concludes the paper.

II. BACKGROUND

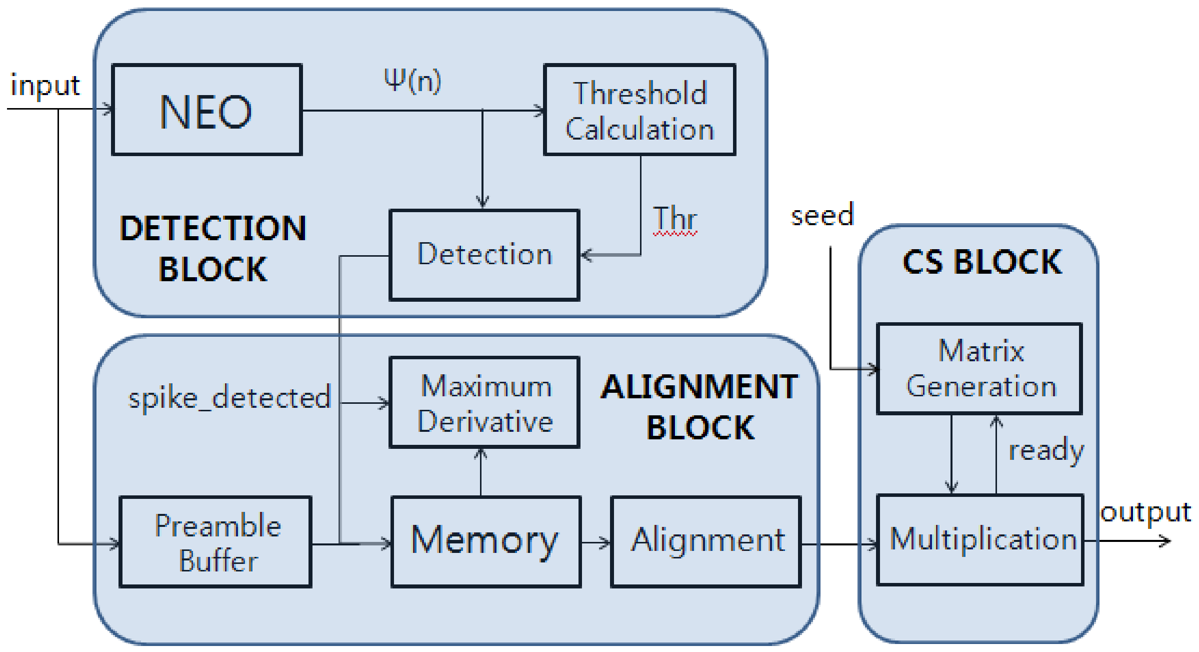


Fig. 2. Block diagram of the whole system.

i. Spike Sorting Algorithm

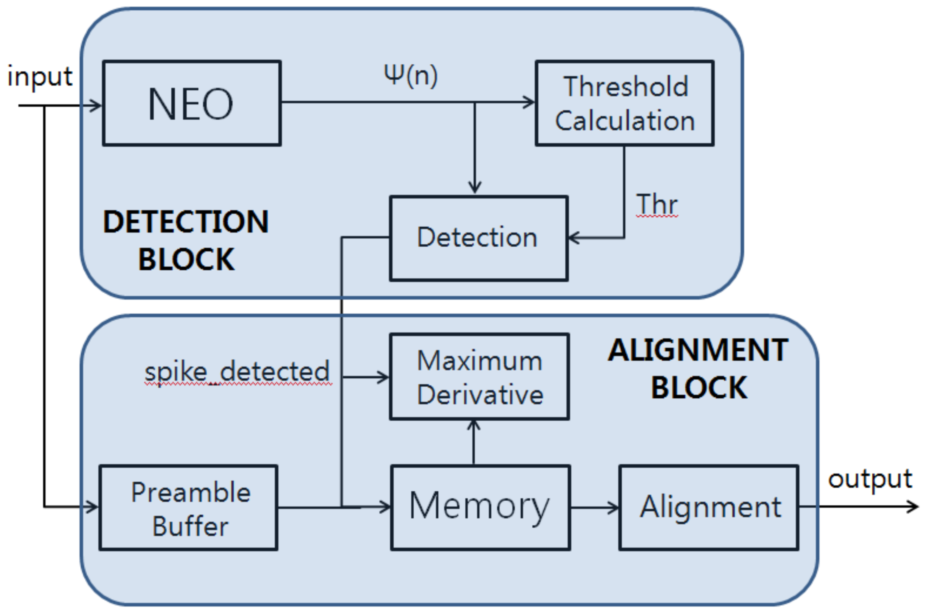


Fig. 3. Block diagram of the single-channel spike-sorting DSP block.

Fig. 3 shows the block diagram of a single-channel spike-sorting DSP module. In the NEO block, $\Psi(n)$ is calculated for each input sample $x(n)$:

$$\Psi(n) = x^2(n) - x(n+1) * x(n-1).$$

The threshold calculation block calculates the threshold as

$$Thr = C\frac{1}{N}\sum_{n=1}^{N}\Psi(n).$$

We chose the number of samples in period N to be 1024, a power of 2, so that the division can be calculated by shift operation. C is constant and 8 was chosen for our implementation [4]. When $\Psi(n)$ exceeds the calculated threshold, the detection block outputs spike_detected signal. When a spike is detected, 50 crossing samples need to be saved in the memory. The maximum derivative block calculates the point of maximum derivative (k) as [5]

$$k = \mathrm{argmax}(s(n) - s(n-1)).$$

The point of maximum derivative serves as the address offset for reading the aligned spikes from the memory. The aligned spike is read out so that k corresponds to the 12th sample of the 50-sample-long aligned spike.

ii. Compressed Sensing Algorithm

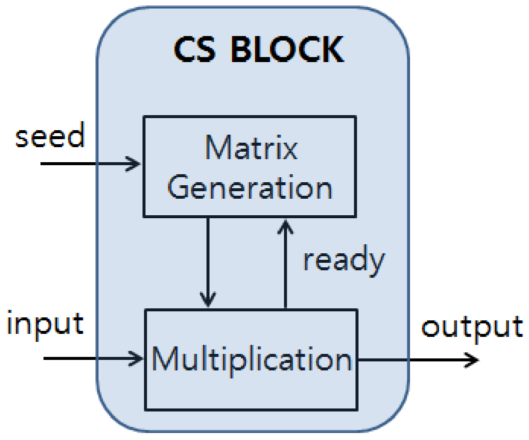


Fig. 4. Block diagram of the single-channel compressed sensing DSP block.

Data compression with compressed sensing is defined by the matrix multiplication [1],

$$[Y] = [\Phi][X],$$

where $[X]$ = N-dimensional input signal, $[\Phi]$ = $M\times N$ measurement matrix,
$[Y]$ = M-dimensional set of measurement ($M < N$).

The compressed sensing algorithm performs a linear projection from the N-dimensional input $[X]$, to an M-dimensional set of measurements $[Y]$ with the $M\times N$ measurement matrix $[\Phi]$, which reduces the data to be transmitted and saves power dissipation by a similar factor. Sparsity is determined as $(1 - K/N)$ where K is the number of significant values among the N input samples [1]. Compressed sensing relies primarily on the signal of interest, so sparsity implies compressibility. For example, a sine wave signal requires many coefficients to represent in time domain, while it requires only one non-zero coefficient to be represented in the Fourier domain [6]. Fortunately, many bio-signals are sparse in representation, which makes them suitable for compressed sensing.

III. IMPLEMENTATION

i. Architecture

At the release of reset, the detection block goes into a “training-mode” where most other blocks are powered off, and the threshold is calculated for the first 1024 samples. Once the threshold is calculated, the input is conditioned using the threshold to detect “spikes.”

Using an external 12.8 MHz clock, a 20 kHz clock is generated in a clock divider block. By implementing a clock divider instead of having a 12.8 MHz clock, the dynamic power consumption is decreased dramatically. All of the blocks are clocked at 20 kHz as the input/output data rates are 20 kHz. Since the design runs at a low frequency, leakage power dominates the total power consumption.

Though the Synopsis tool library includes standard memory cells which are big in size, it would be inefficient to our design. Instead, we used registers with pointers to implement a circular structure. The preamble buffer has a size of at least 22, and it is sufficient to keep 40 post-samples. If the maximum derivative occurs such that more post-samples are needed, the preamble buffer will output zeros when past the post-samples size.

ii. NEO

In the basic architecture version, the NEO calculates $\Psi(n)$ from four registers, x(n0), x(n), x(n+1), and x(n-1), which store the incoming input stream sequentially, shifting them from the leftmost register to the rightmost register, and finally out to the buffer. Four registers are needed for Verilog implementation so that the earliest value in the x(n0) register can be sent to the buffer while the NEO computes the next value from the other three registers.

Calculation halts for 50 clock cycles when a spike is detected.

iii. Threshold Calculation

Once the NEO block first begins to produce values of $\Psi(n)$, the threshold calculation block starts to accumulate them according to the equation: $\mathrm{Thr} = C\frac{1}{N}\sum_{n=1}^{N}\Psi(n)$, with N = 1024 and C = 8. The minimum word length of 23 was used for the accumulation to reduce the number of memory elements. Once the threshold is computed, the threshold calculation is deactivated.

iv. Preamble Buffer

The preamble buffer is a circular buffer implemented with 22 registers and two pointers, P1 and P2. P1 is an internal pointer to the location which stores the data that comes from the four registers of the NEO block, and it increments by one each time. P2 is a register used for sending out data from the buffer to construct the final output data stream of 50 samples. When a spike is detected, P1+1 is sent to P2, and the buffer stops accepting input data for 40 counts. However, detections may occur close enough in time that the buffer must have enough pre-detection samples for the next spike, so an input signal to the buffer informs it when the output is not retrieving samples from the buffer (i.e. output samples are retrieved from the memory instead). Therefore, the buffer resumes storage of input samples to account for close, adjacent spikes.

v. Memory

The memory is composed of 40 registers with a single pointer that increments linearly and resets to zero for each spike detection. When a spike is detected, input samples are stored, and the maximum derivative calculation is processed immediately on the input stream to produce the index of maximum derivative. The derivative calculation updates a register with an index number between 1 and 40 when it completes the calculation, so that output samples can be retrieved from the buffer and memory, with the sample at the index as the 22nd sample of the output data stream.

vi. Alignment

When the index number is calculated from the memory, the 50 output data can be prepared. Depending on the value of the index, the data can come from both the preamble buffer and the memory, or from the memory alone if the index is greater than 22. If the data comes from the memory alone, which is of size 40, it will output zeros when past the memory size. If a spike occurs when the memory has not been read yet (if the two spikes occurs too close to each other), then the second spike is ignored. Otherwise, the cost of the overhead to handle these two spikes would be too large.

vii. Matrix Generation

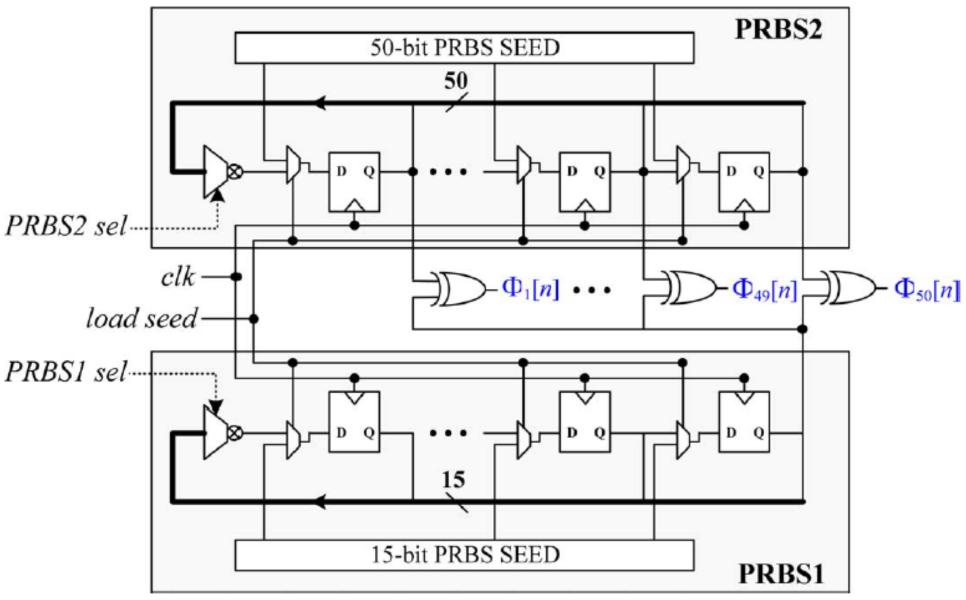


Fig. 4. Block diagram of the measurement matrix generation block [6].

The matrix generation block requires two PRBS generators. As shown in Fig. 4, a 15-bit PRBS1 output is XOR'd with the every output of 50-bit PRBS2 in our implementation. A wide variety of pseudo-random matrices can be produced, because the seed sequence and the length of the matrix are programmable. The resulting implementation requires only 65 flip-flops for an M of 50, compared to what would have been 750 flip-flops to produce PRBS sequences with the same length matrix [7]. With these improvements, the matrix generation power is reduced to less than 10% of the total power consumption.

viii. Multiplication

Once a random matrix of size M is generated from a matrix generation block, the matrix is stored in the registers of a multiplication block, and multiplication with each element of random matrix at every clock edge proceeds. In order to reduce power consumption, the matrix generation block is powered off while matrix calculation is running. Multi-channel multiplication block implementation is able to achieve parallel computation because we have a long slack and a large voltage overhead to reduce. However, the current Synopsys tool we are using does not allow voltage to be reduced below 0.78 V, so we need to characterize low-voltage libraries ourselves. Thus, multi-channel block implementation can be attempted in future work.

IV. EXPLORATION

i. VDD Reduction and Multi-Vt Transistor

At the 20 kHz clock, many modules remain mostly in standby mode. In this situation, leakage power could dominate total power consumption. We confirmed this fact by simulating and analyzing power consumption in a pt-pwr report. We synthesized our design mostly with HVT cells because HVT cells have less leakage current and saved more than 90% in the total power consumption at each voltage level.

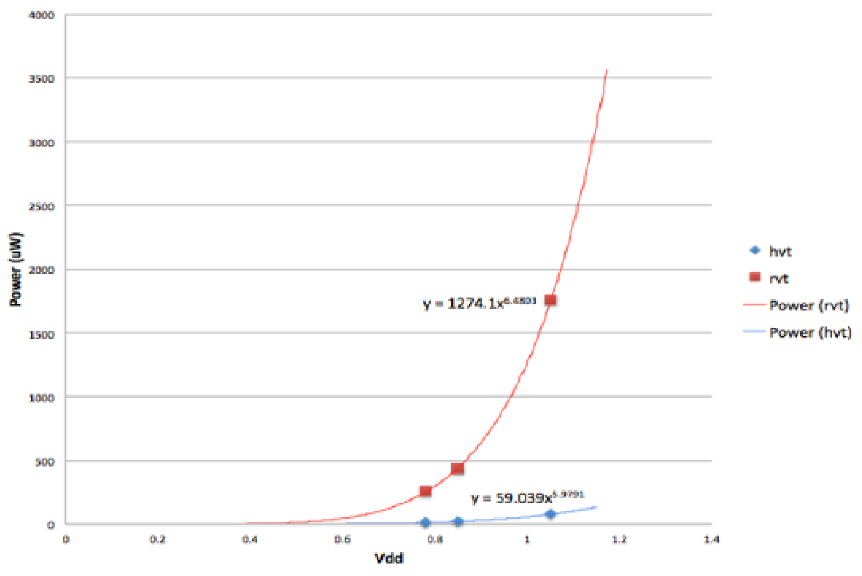


Fig. 5. Power vs VDD for RVT and Multi-Vt standard cells.

ii. Clock Gating

In Design Compiler, the compile_ultra command is used to synthesize RTL code into a gate-level netlist with many levels of optimizations. During this process, the -gate_clock option can be used to gate clocks to registers whenever possible. Using this option, 98% of the registers were clock gated by the Design Compiler during synthesizing. Clock gating resulted in a significant reduction in dynamic and internal power, while the leakage power was reduced by about 8%. The total power was effectively reduced by about 8%.

TABLE I. Power Reduction with Clock Gating.

	Switch Power	**Int. Power**	**Leak. Power**	**Total Power**
Reduction in Power with Clock gating	25%	50%	8%	8%

iii. Memory and Buffer

Without any low-power techniques, almost 100% of the power was attributed to leakage power, and about 47% came from the memory and about 18% came from the buffer. There are about 40 registers, each with 10 bits; hence, 400 registers are used by the memory, which is about 50% of the total registers used in the design. Similarly, the buffer uses 220 registers. The memory and buffer are register arrays. Instead of shifting the data in the register array, the circular memory and buffer use a pointer to indicate the register that is used to read and write data. By doing this, the circular memory and buffer are able to save a great amount of switching power. But in terms of the total power consumption, this saving is insignificant. To reduce the power consumption of the memory and buffer, we need to reduce the leakage power by using HVT cells. In order to further improve our performance, four registers (10 bits each) used in the NEO computing block were removed. Data saved in the preamble buffer could then be used for the NEO computation instead. With this optimization, we saved static power by reducing the number of registers. The result is summarized in the following table.

The total power savings should be about 1.5 uW $\times$40 registers = 60 uW, which agrees with the results shown in the table below.

TABLE II. Power consumption for architectures with and without four dedicated NEO registers

	Total Power Consumption (mW)	**Saving (%)**
With Four Register in NEO	1.74	NA
Without Four Register in NEO	1.68	3.4

iv. Word-length

Reducing the number of registers can decrease the power consumption. First, by using the given equations for threshold and NEO calculations and the fact that the input stream has 10 bits, we found that the minimum number of bits required in the worst case. Then, we used MATLAB simulations on the actual data to find out the maximum value of $\Psi(n)$, the threshold value, the terms in the threshold summation, and the maximum derivative, so that the number of bits could be reduced to the minimum.

The following tables summarize the number of bits used and power consumption after optimizing for word-length. Compared to the basic design, the leakage and total power decreased by 10uW, while dynamic power decreased by roughly 20%. This is expected because the basic design uses 11 more registers than the minimum required, and each register consumes 1uW of leakage power on average.

TABLE III. Summary of word-length requirement.

Size of Registers	**Based on Real Data**	**Worst Case**	**Actual Bits Used**
Input / Output	10	10	10
$\Psi(n)$	20	20	20
Threshold	16	22	16

Accumulate	23	30	23
Derivative	11	11	11

v. Measurement Matrix

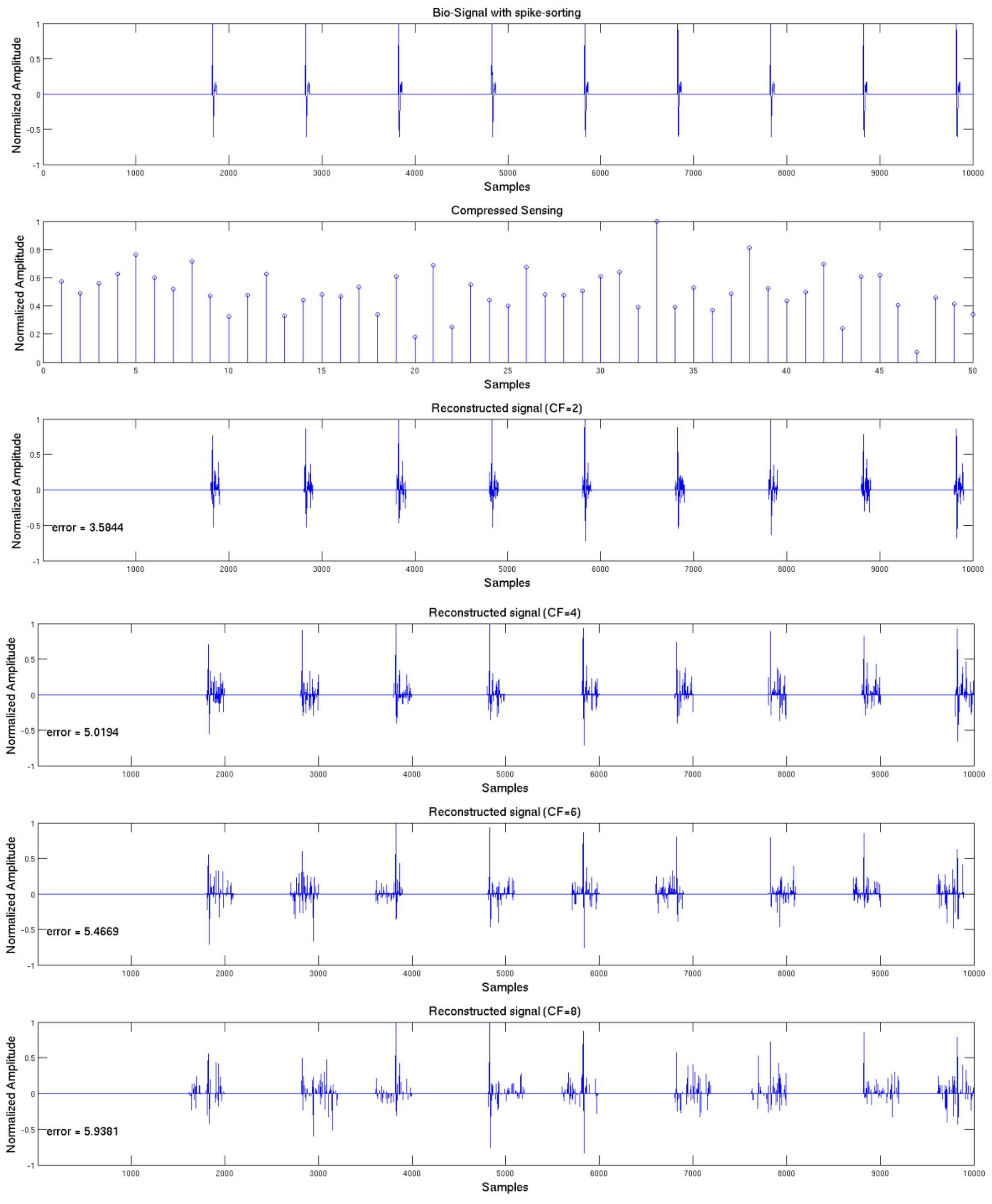


Fig. 6. From the top: bio-signal with spike-sorting; compress sensing on bio-signal with spike-sorting; reconstructed signals with CF=2, 4, 6, and 8, respectively.

Fig. 6 shows compressed sensing on bio-signals with spike-sorting and reconstructed signals with compression factors (CF) = 2, 4, 6, and 8, respectively. Compression factor (CF) is defined by N/M, with an N-dimensional input and M-dimensional output. In each plot, M is fixed to 50 and N is swept to vary compression factor. Because the purpose of compressed sensing is data compression, a higher compression factor is desirable and is proportional to power savings. As expected, the details of reconstructed signals decreases as the data compression factor increases. Based on our simulation results, details of signal are lost at CF = 8 and beyond. Thus, details of signals can be accurately reconstructed with CF = 6 and below.

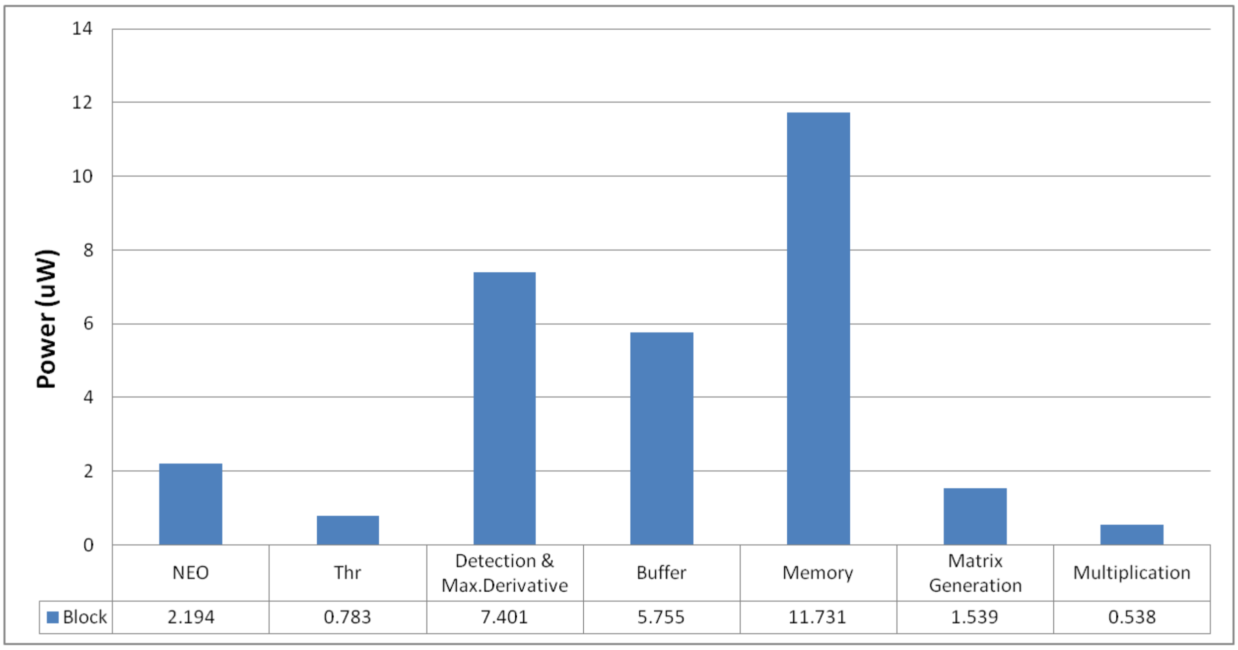
V. MEASURED RESULTS

Each block of the whole system is implemented in Verilog and synthesized at 0.78V with multi-threshold transistors (mostly with high-threshold transistors) by following the 28/32nm Synopsys design tool flow. At every stage of synthesizing (RTL, dc-syn, and icc-par), the output of the design is verified with respect to the output of MATLAB's simulation result.

i. Power

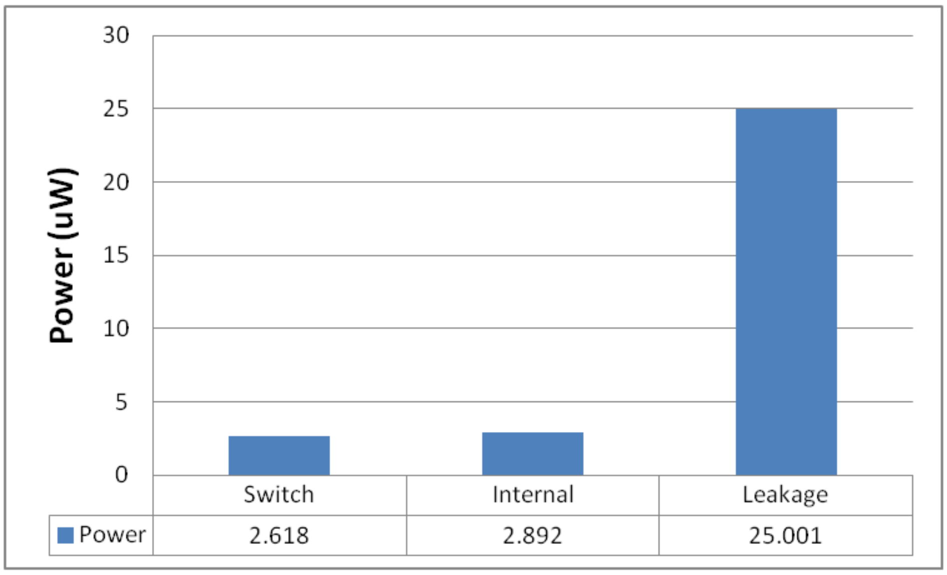
Power is measured by Synopsys PrimeTime. Fig. 7 shows the power breakdown of each block in the system, and Fig. 8 shows the power breakdown of the total power consumption. As expected, the memory block consumes the most power because leakage power dominates in our system. Our system operates at 20kHz, which sets every block in standby mode for a long time. We can also notice that internal power consumes even more than switch power; this is caused by the short-circuit current between the power and ground. Both leakage and internal power can be reduced by implementing a power-gating technique effectively. For the next phase in the EE241B project, we are planning to implement a power-gating technique in Synopsys tools. Also, we cannot reduce the voltage supply below 0.78V in Synopsys tools now, so we are manually characterizing the low-voltage library. Considering that we have enough time slack, we can further reduce power.

Another way of reducing power in this situation is to implement SRAM for memory. SRAM costs area and is difficult to be implemented in Verilog; however, Chisel, a new hardware description language (HDL) developed by UC Berkeley, allows us to implement a SRAM in HDL level. Thus, we will try it in the next phase as well. The current total power consumption is 30.462 uW.



	NEO	Thr	Detection & Max.Derivative	Buffer	Memory	Matrix Generation	Multiplication
■ Block	2.194	0.783	7.401	5.755	11.731	1.539	0.538

Fig. 7. Power breakdown of each block.

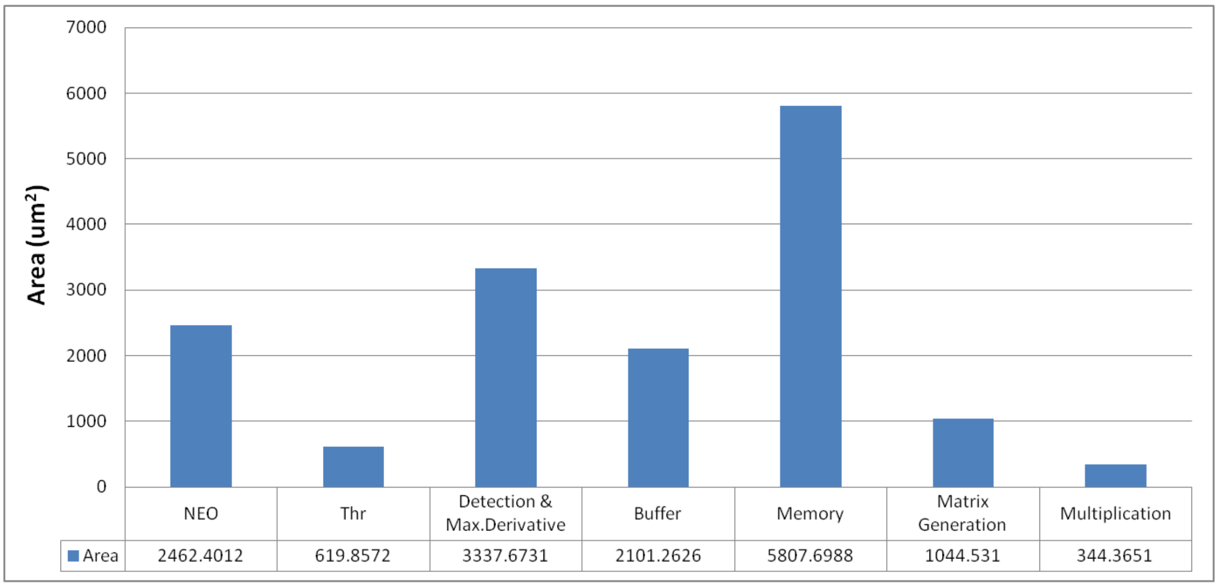


	Switch	Internal	Leakage
■ Power	2.618	2.892	25.001

Fig. 8. Power breakdown of top module.

ii. Area

Area is measured by a Synopsys IC compiler. As expected, the memory takes up the most part in the total area. The trend in power and area is quite proportional, so we reached the conclusion that we can reduce power by optimizing area as much as we can. Thus, we synthesized our design to target a density of no more than 70 or 80 percent to allow space for clock tree synthesis, buffer insertion, and routing. The width and height of our layout is 165 um x 165 um, and the total area of the block is 17953.53 um^2.



	NEO	Thr	Detection & Max.Derivative	Buffer	Memory	Matrix Generation	Multiplication
🟦 Area	2462.4012	619.8572	3337.6731	2101.2626	5807.6988	1044.531	344.3651

Fig. 9. Area breakdown of each block.

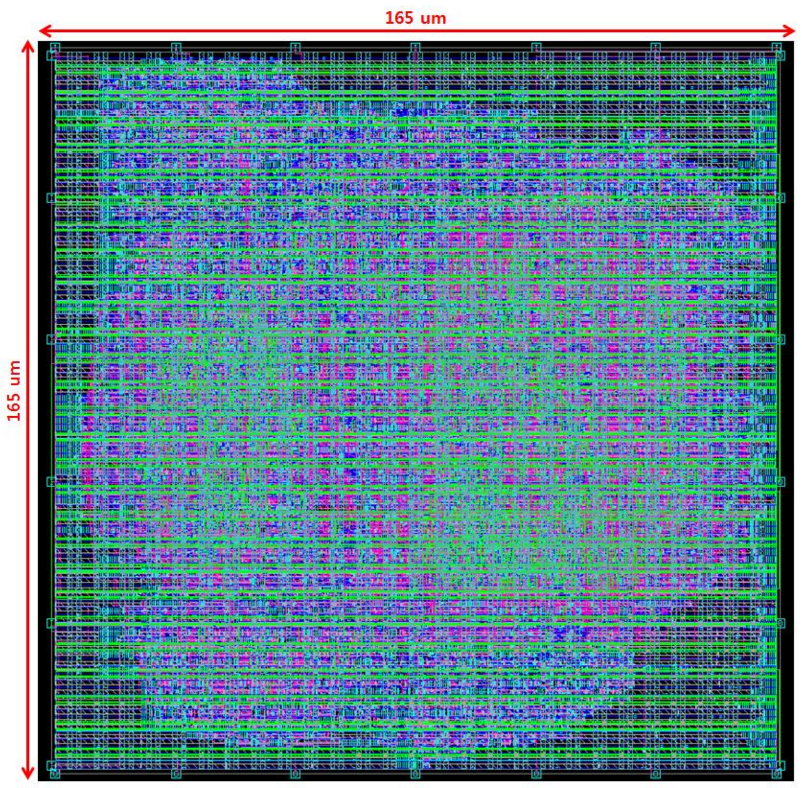


Fig. 10. Layout by IC-Compiler in 28/32nm Synopsys tool.

VI. CONCLUSIONS

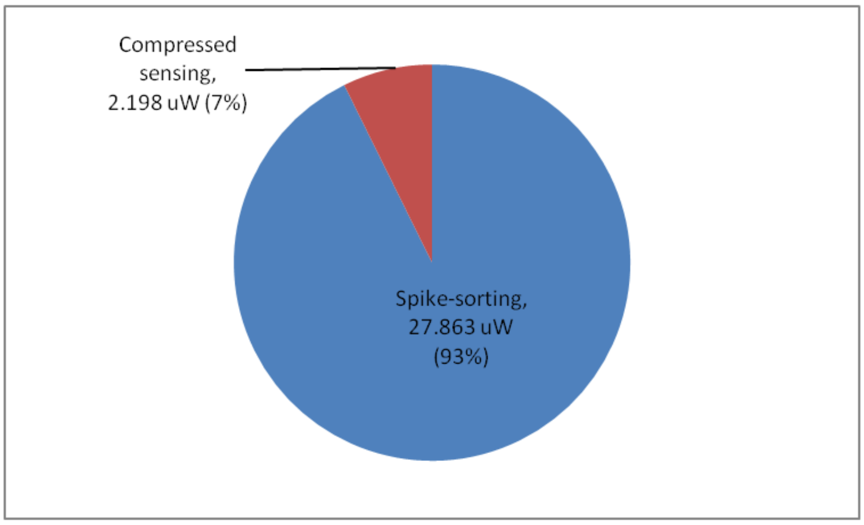


Fig. 11. Power consumption comparison for spike-sorting and compressed sensing.

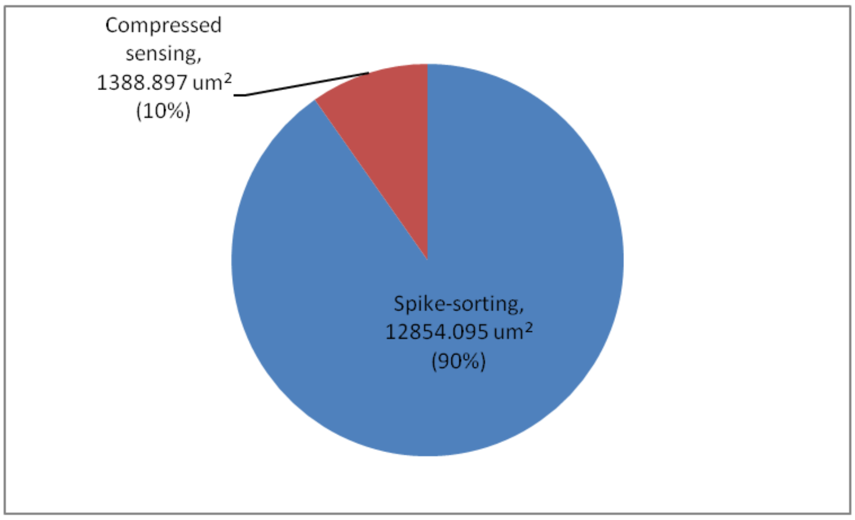


Fig. 12. Area comparison for spike-sorting and compressed sensing.

As shown in Fig. 12 and Fig. 13, a spike-sorting algorithm is more expensive than compressed sensing in both power and area. However, considering the natural weakness of compressed sensing on noise, this combination of both techniques can be implemented to achieve a robust operation. For example, compressed sensing is often exploited in MRI applications where noise frequently appears, and our approach can be considered for this situation.

TABLE IV. Summary

Technology	**28 / 32 nm Synopsys Tool**	
VDD	**0.78 V**	
No. of Channels	**1**	
Clock	**20 kHz**	
Compression Factor	**6**	
Total Power	**30.462 uW/channel**	
	Power (spike-sorting)	**27.863 uW/channel**
	Power (compressed sensing)	**2.198 uW/channel**
Area	**17953.53 um^2**	

This paper explains the design of combining spike-sorting and compressed sensing algorithms in a DSP chip. The chip has a core area of 17953.53 um^2. Clocked at 20 kHz, it consumes 30.462 uW/channel at 0.78V for a single channel with a 10bit/sample input stream.

REFERENCES

[1] Gangopadhyay, D.; Allstot, E.G.; Dixon, A.M.R.; Natarajan, K.; Gupta, S.; Allstot, D.J., "Compressed Sensing Analog Front-End for Bio-Sensor Applications," *Solid-State Circuits, IEEE Journal of* , vol.49, no.2, pp.426,438, Feb. 2014.

[2] D. L. Donoho, "Compressed Sensing," *IEEE Trans. Inf. Theory*, vol. 52, no.4, pp. 1289-1306, 2006.

[3] V. Karkare, S. Gibson, and D. Marković, "A 130-uW, 64-Channel Neural Spike-Sorting DSP Chip," *IEEE J. Solid-State Circuits*, vol. 46, no. 5, pp. 1214-1222, 2011.

[4] S. Gibson, J. W. Judy, and D. Marković, "Comparison of spike-sorting algorithms for future hardware implementation," in *Proc. IEEE Engineering in Medicine and Biology Conf.*, 2008, pp. 5015–5020.

[5] M. Rizk *et al.*, "A single-chip signal processing and telemetry engine for an implantable 96-channel neural data acquisition system," *IOP J. Neural Eng.*, pp. 309–321, Sep. 2007.

[6] F. Chen, A. P. Chandrakasan, and V. M. Stojanović, "Design and analysis of a hardware-efficient compressed sensing architecture for data compression in wireless sensors," *IEEE J. Solid-State Circuits*, vol. 47, pp. 744–756, Mar. 2012.

[7] X. Chen, Z. Yu, S. Hoyos, B. M. Sadler, and J. Silva-Martinez, "A sub-Nyquist rate sampling receiver exploiting compressive sensing," *IEEE Trans. Circuits Syst. I*, vol. 58, no. 3, pp. 507.520, 2010.