# A Hardware Design for In-Brain Neural Spike Sorting

Yinan Liu Jiayi Sheng Martin C. Herbordt
Department of Electrical and Computer Engineering
Boston University, Boston, MA

***Abstract*—Neural spike sorting is used to classify neural spike signals based on neuron type and so is an essential step in decoding brain signals. Because of its computational complexity, spike sorting is generally carried out offline or, at least, using a transmitted signal. In contrast, in-brain spike sorting would reduce the data that needs to be transmitted by orders of magnitude with a corresponding reduction in transmission power. This would enable real-time wireless neural recordings. In this paper, we design and characterize a hardware prototype for in-brain spike sorting. Our design is able to reduce the wireless transmission power by a factor of over 200 over direct transmission. Also, compared with the current state-of-the-art, our design increases the sorting accuracy from 75% to 93% while remaining within hard constraints for power, power density, and real-time processing.**

***Keywords*-Spike Sorting; Wireless body sensor networks; ASIC;**

## I. Introduction

In the human brain billions of neurons communicate with each other via electro-chemical signals. The electrical signals associated with this communication are called "spikes." To acquire such signals in fine detail and localization, current technologies require the implanting of electrode arrays. Every electrode detects signals from multiple neurons surrounding it; these are called multi-unit activities. In many applications, e.g., brain-machine interfaces (BMI), processing signals from single neurons (single-unit activities) is essential, but difficult to perform directly [1], [2]. By decoding multi-unit activity signals into single-unit signals, scientists are able to determine the meaning of instructions from brain to body. One of the goals is actuation, i.e., the ability to reverse the process and generate signals to control external devices that can help disabled individuals. For example, BMI has enabled control of a prosthetic arm [3] and a smart wheelchair [4]. The critical step in this process is to classify the multi-unit signals into single-unit signals according to the neuron source types [5].

Figure 1 shows the four steps in spike sorting. The first is to use an analog front-end (AFE) filtering module to amplify, filter, and digitize the raw signal. After that, the processed signal is sent to the detection-and-alignment module. There, spikes are detected and aligned to a common feature, e.g., their peaks. The aligned spikes are then sent to the feature extraction module, where the signals are converted to fewer data points by reducing their dimensionality. In the final step, the spike features are sent through a clustering algorithm where the spikes are categorized into classes based on source neuron type. Figure 2 shows intermediate results corresponding to the four steps. Figure 2 (a) shows the filtered signals and the detected spikes. Figure 2 (b) shows the spikes after alignment. In this example, the spikes are all aligned to the peak that has the highest absolute value. Figures 2 (c) and (d) show the sorted results. Figures 2 (c) shows the average waveform of each cluster. Each dot in Figure 2 (d) represents an entire spike; this works because by now each spike has been reduced to a 2D vector through feature extraction.

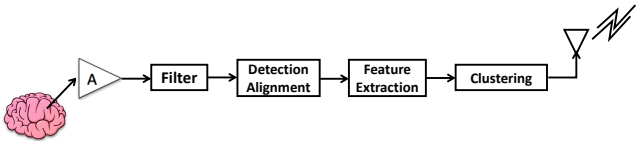


Figure 1. Flow chart of a general spike sorting scheme

Three possible neural recording systems are shown in Figure 3. The first (a) transmits the raw neural signals from the brain through physical wires. All the processing is performed independently of the sensor. This setup does not require real-time in-body processing. It does, however, restrict freedom of movement of the subject and increase of the chance of infection. Also, the wires can increase the noise. In the second configuration (b), the raw signals from the neurons are transmitted through a wireless transmitter. While (b) obviates the need for wires, transmitting raw signals implies a high transmission rate and therefore high transmission power. High power is especially undesirable in BMI because the heat generated is dangerous to brain tissue. The first and second alternatives both require software to do the spike sorting. An 8-hour experiment generates about 100GB of data, which requires about 30 hours to process using a conventional spike sorting software package [5].

Our goal here is to facilitate a configuration as shown in Figure 3(c): spike sorting is performed in the brain and the data transmission reduced to only vectors in a feature space and corresponding time-stamps. To do this in real-

This work was supported in part by the Charles Stark Draper Laboratory through their URAD Program Award #SC001-0000000834. Email: (ynliu|jysheng|herbordt)@bu.edu

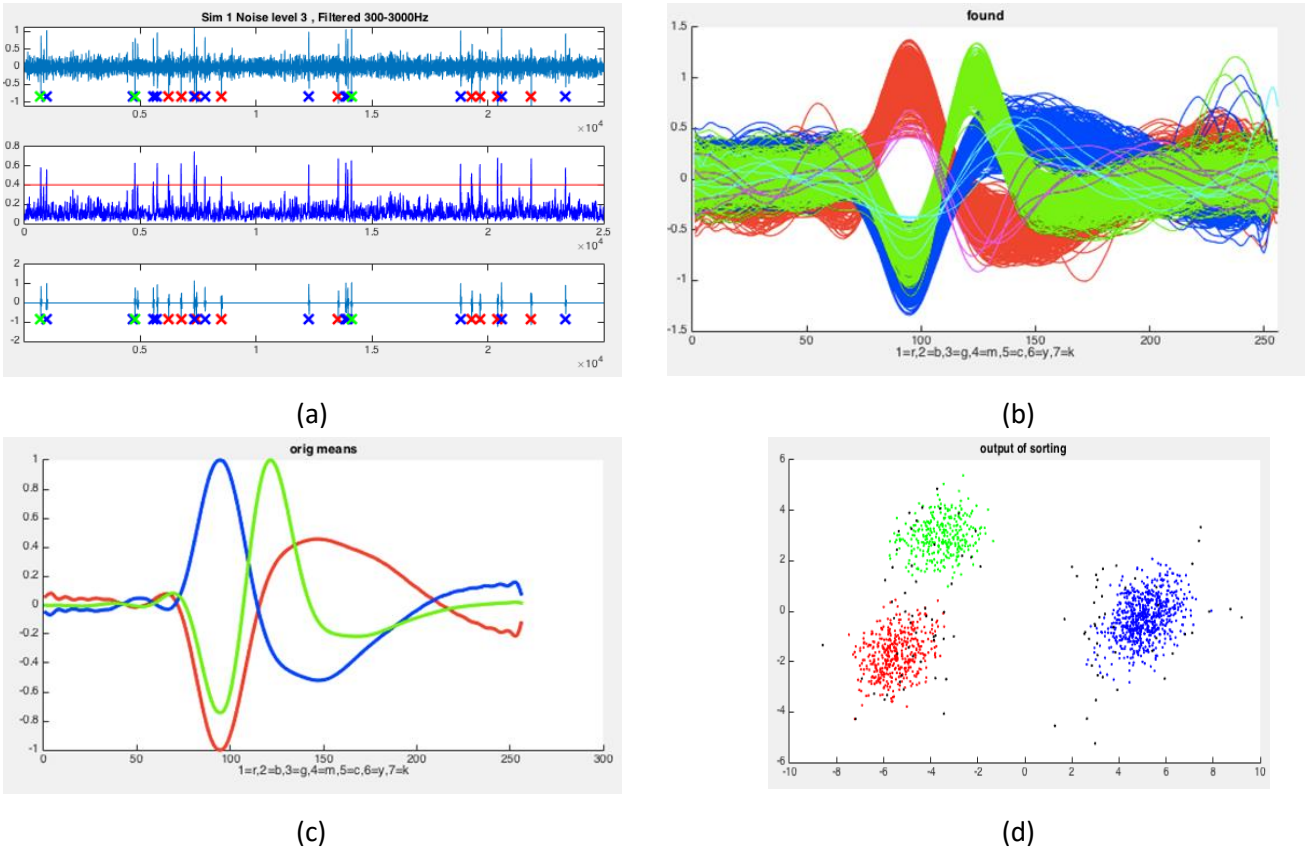


Figure 2. Examples of data generated during four steps of spike sorting: (a) filtered waveform and detected spikes, (b) aligned spikes, (c) and (d) clusters in two representations, function average by type and distribution in feature space.

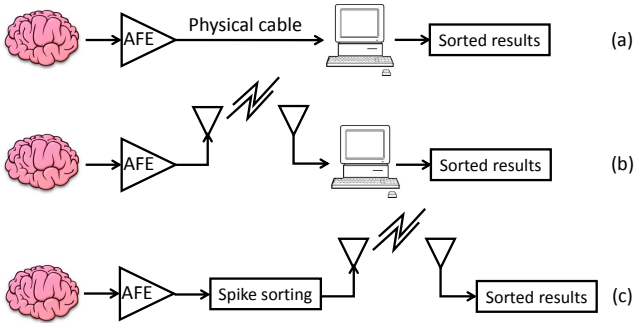


Figure 3. Three possible neural recording systems

time, with low power, and with high accuracy requires a dedicated ASIC. In this work, we propose a novel spike-sorting hardware prototype that reduces transmission power by a factor of 278 over the non-preprocessed solution. Moreover, our design meets the two critical requirements: first, it processes and transmits in real-time and, second, its power output is within the budget to allow embedding in brain. Perhaps the most significant contribution of this work is that it meets those constraints while improving accuracy from 75% to 93% over the current state-of-the-art.

Some prior implementations of spike-sorting DSP hardware are as follows. The work by Olsson and Wise [6] is one of the earliest implementations of wireless in-brain spike-sorting, but its function is limited to spike detection. Karkare et al. [7] and Chae et al. [8] add feature extraction but do not implement clustering. A later study by Karkare et al. [9] implements Osort for in-brain unsupervised clustering and is the starting point for our work here; we refer to this as the *reference design.* We add several features. Rather than using simple value detection, we apply a sliding window that can buffer the data in front of the threshold point and so enable the capture of a more complete waveform. We also add alignment and more complex detection.

The rest of the paper is organized as follows. In the next section, we introduce the algorithms for the various spike sorting steps and explain the rationale behind certain design decisions. In the third section, we introduce the details of our design including architecture and state machine. In the fourth section, we evaluate the design and present our conclusions.

## II. Spike Sorting Algorithms

In this section we give an overview of the algorithms used in our design.

### A. Filtering

The first step is filtering. The raw data is bandpass-filtered to remove the local field potential and high-frequency noise [5]. This step is usually done in the analog front end, which is beyond the scope of this work.

### B. Detection and Alignment

The detection step separates spikes from background noise. The two main steps are spike determination and application of a threshold. In our design, the threshold is determined by the standard deviation of the signals. The application of the threshold works as follows. A comparator continuously checks the voltage levels of incoming data points. A circular buffer stores the 48 points preceding the current data point and the 79 points after it. Whenever the absolute value of current data point goes above the threshold, another module determines the positive and negative peaks among the middle 64 points (24 points preceding and 39 succeeding).

There are three possibilities. (i) If neither of the absolute values of the two peaks is above twice the threshold, then the 39 points succeeding points are abandoned. The next point to be checked is then the 40th point after the current point. (ii) If both peaks are above twice the threshold, then the algorithm picks the first one as the base point. (iii) If only one point is above twice the threshold, then that one is selected to be the base point. Once the base point has been determined, the 24 points preceding and the 39 points succeeding (and the base point itself) are assumed to be a complete spike. Alignment is now completed with all spikes aligned to their peaks. The detected and aligned spikes then are sent to the feature extraction and clustering stage.

### C. Feature Extraction and Clustering

Spike Sorting is based on the assumption that each neuron generates a different and distinct wave shape that remains constant in the recording session [5]. We therefore need a clustering algorithm that can distinguish neurons from the mixture of their spikes. In our scenario, our candidate clustering algorithm needs to have the following characteristics: unsupervised, real-time, online, and easy to implement in hardware. *Unsupervised* means that the algorithm can figure out how many clusters are needed with no further information. This is as opposed to supervised clustering algorithms such as K-means, which need the number of clusters *a priori*. *Real-time* means that the processing rate is able to keep up with the input data rate.

The current consensus appears to be that the OSort algorithm is one of the best candidates, not only because it meets all of these requirements, but also because it does not need an explicit feature extraction step [9], [10]. OSort performs real-time clustering using an iterative process by comparing the Euclidean distances among spike waveforms in the time domain. Its flow chart is shown in Figure 4.

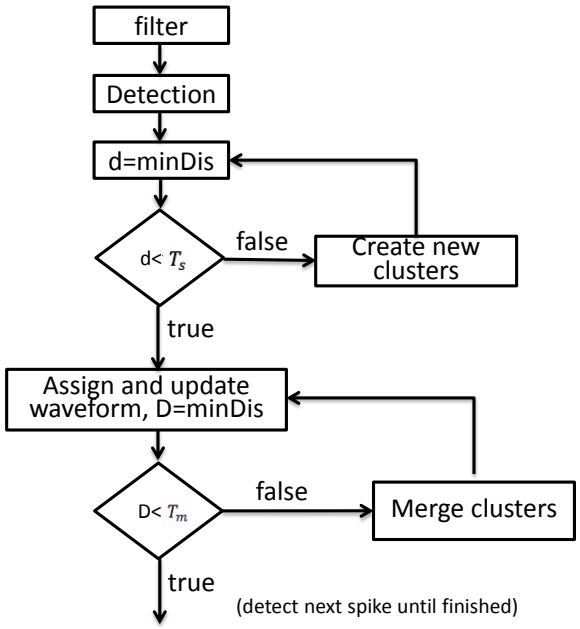


Figure 4. Flow chart of OSort algorithm

In the first step, the first cluster is initialized with the first spike. After that, whenever a new spike comes in, the algorithm computes the Euclidean distances between it and all of the existing clusters. If the minimum distance is smaller than a predefined threshold, the new spike is assigned to the corresponding cluster based on Equation 1. The threshold is the crucial element of OSort. It is equal to the squared average standard deviation of the signal, calculated with a sliding window [10].

$$c'(n) = \frac{(N-1)c(n) + s(n)}{N} \tag{1}$$

In this equation, $N$ is the number of spikes that have been assigned to the current cluster, $c(n)$ is the mean spike waveform of the current cluster, and $s(n)$ is the incoming spike waveform. This equation shows that the new spike $s(n)$ becomes less important when $N$ is large. In our implementation, we stop updating the cluster when $N$ is over 50. It has been proven that the mean spike waveform converges when $N$ is larger than 50 and sorting accuracy is not affected if we assume that the average spike waveform is constant. If the minimum distance is larger than the predefined threshold, then a new cluster is created based on this new spike. At this point, we have either a new cluster or a updated cluster. In either case, the algorithm checks distances between this new (or modified) cluster and all the other clusters. If there is a pair of clusters whose distance

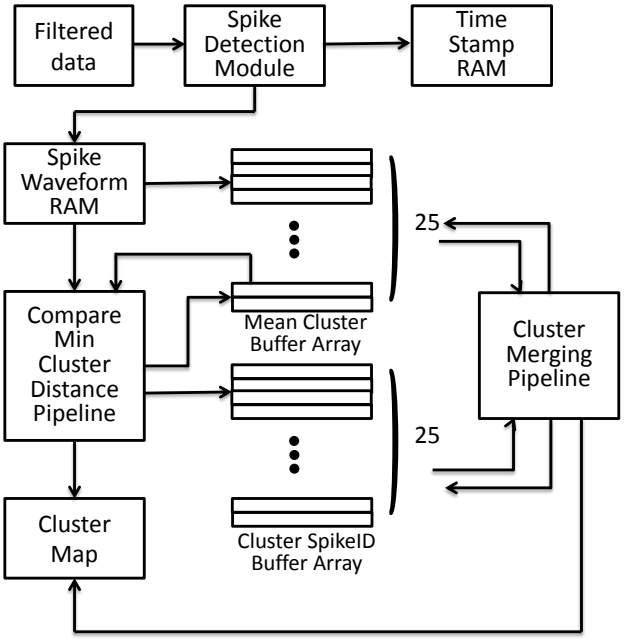


Figure 5. Top-level architecture

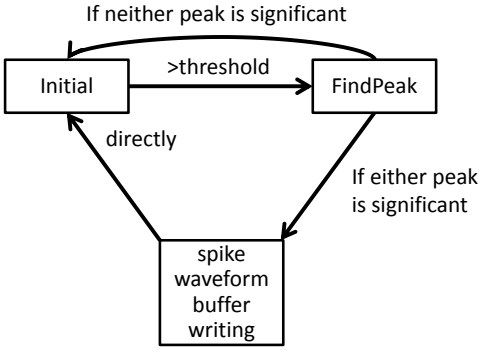


Figure 6. FSM of spike detection module

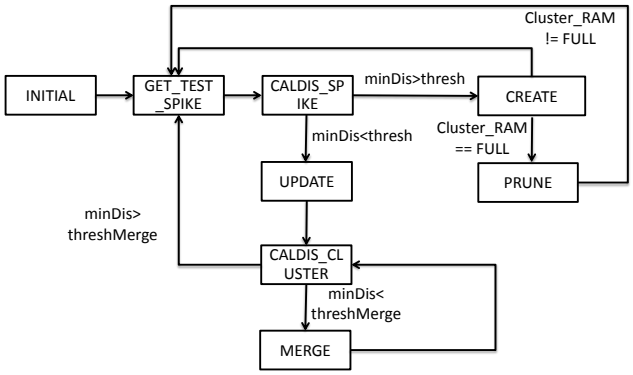


Figure 7. FSM of OSort implementation

is smaller than a threshold, then they are merged into one cluster. The mean waveform of the new cluster equals the weighted mean of previous two clusters.

In our simulations we have observed that OSort generates many more temporary clusters than converged. In particular, we have found that capping the number of temporary clusters at 25 results in no loss in spike sorting accuracy; i.e., the clustering algorithm still converges to the number of types of neurons (usually fewer than 6) around the electrode. If the 25 temporary clusters are all occupied and a new cluster needs to be created, all of the temporary clusters that have only one spike are cleared. We refer to this step as pruning. The entire flow repeats until all the spikes are processed.

After OSort processes all the spikes, we have two types of results. The first is the mean waveform of each cluster. The second is the cluster ID and the timestamp of each spike. Usually we just need to transmit the second type via the wireless transmitter. Compared with transmitting the raw waveform, the data rate is reduced substantially.

## III. Implementation

In this section we give the details of our design. The top-level architecture is illustrated in Figure 5.

### A. Spike Detection Module

After the raw spike data are filtered, they are sent to the spike detection finite state machine (FSM); the state transition diagram is shown in Figure 6. In the first state (initial), a comparator monitors whether there is a data point greater than the threshold. If one is found, then the state machine transits into the FindPeak state. There the 128 buffered data points are sent to a FindPeak module. This module spends 128 cycles to determine the positive and negative peaks. If either one of them is significant, which means a spike has been detected, then the state machine goes to the spike-waveform buffer-writing state. This requires 64 clock cycles. If neither peak is significant, then the state machine goes back to the initial state. After the spike waveform is written into the buffer, the FSM returns to the initial state. Whenever a spike is detected, the spike ID and its timestamp are sent to a timestamp RAM while the spike waveform is sent to the spike waveform RAM.

### B. OSort Module

As illustrated in Figure 4, there are several other important components. The spike waveform RAM stores the detected spike and triggers the OSort FSM to begin work. The min cluster distance pipeline computes the Euclidean distance between two spike waveforms. Two spike waveforms could be the either the mean waveform of a cluster or the waveform of a new spike. There are two buffer arrays to hold the temporary clusters because each cluster has two kinds of information to be stored: the mean waveform and the spike IDs. As justified above, we allocate 25 slots to the two buffer arrays. Whenever a cluster needs to be modified or deleted, its corresponding slots in both buffer arrays are

simultaneously modified and invalidated.

The cluster map module tracks the mapping between cluster IDs and buffer slot IDs. It plays an important role in two scenarios. The first is when a new cluster needs to be created. This is because the module provides the IDs of buffer slots that are available to map to the new cluster. The second is to tell the min distance pipeline which buffer slot to work on (as it is traversing all of the clusters). Most of time the occupied slots are scattered among 25 hard-wired slots; i.e., they are rarely contiguous. The final component is the cluster merging pipeline. It starts working when two clusters are too close to each other and need to be merged.

Following the OSort algorithm, we have designed a corresponding FSM (illustrated in Figure 7). Whenever the spike waveform RAM gets a new complete spike, the FSM enters the CALDIS_SPIKE state. It pushes the spike waveform into the min cluster distance pipeline, where the closest cluster is determined. The state machine then enters the CREATE or UPDATE state depending on whether or not the closest distance is over a threshold. If it enters the CREATE state, the cluster map module determines which cluster buffer is available. If there none is available, the state machine goes to the PRUNE state. There, all the temporary clusters having only one cluster are invalidated. If the state machine enters the UPDATE state, the mean waveform of the closest cluster is updated (according to Equation 1).

After the UPDATE state, the state machine goes into the CALDIS_CLUSTER state. The pairs formed by the modified cluster, and all the other clusters, are sent to the min cluster distance pipeline to compute the distances between clusters. If the minimum distance is smaller than a predefined threshold, then the two corresponding clusters are merged. Their IDs are sent to the cluster merging pipeline, where one cluster is moved from its own buffer to the other buffer, and its original buffer space becomes available again. This new information changes the value of the cluster map so that incoming spikes can be assigned to the newly available buffer space.

## IV. Evaluation

In this section, we evaluate our design with respect to the following three measures: (i) data processing rate, to make sure it meets the real-time requirement; (ii) power and area consumption, to make sure it is within the power budget; and (iii) sorting accuracy, for comparison with the state-of-the-art. We built an RTL model using Verilog and simulated and validated our design using ModelSim. For synthesis and place & route we used the Cadence Encounter RTL Compiler [11] using a 45nm technology.

### A. Timing evaluation of our spike sorting design

The timing results are shown in Table I. From our simulation results, we observe that our design processes 0.444 samples every cycle on average. The typical input data rate of neural spikes is about 25K samples per second [5], which means that our clock frequency could be as low as 56kHz and still keep up with the input data stream. Since this operating frequency is easy to achieve in likely ASIC processes, our design therefore satisfies the real-time requirement.

Table I
TIMING EVALUATION

| Input data rate | processing rate | lowest possible clock frequency |
|---|---|---|
| 25kSa/s | 0.444 Sa/cycle | 56kHz |

### B. Power and Area consumption

Table II
AREA CONSUMPTION

| Gates | Cells | Area | state-of-the-art area [9] |
|---|---|---|---|
| 97076 | 28772 Sa/cycle | 0.077 $mm^2$ | 0.07 $mm^2$ |

The area consumption results are presented in Table II. The area consumption of our design is 0.077 $mm^2$. We compare this to the area consumption of the state-of-the-art reference design [9], which is 0.07 $mm^2$. As they used a 65nm technology, our design likely consumes roughly twice the area, after normalizing for process technology.

The reason for the difference is the relative complexity of our spike detection module; in the reference study, spike detection is just an absolute value comparator. Our spike detection module is an improvement for two reasons. First, we have an FSM that buffers the data points before the point that exceeds the threshold so the waveform of the spike captured is more complete. Second, our FSM ensures that all the spikes are aligned to their peaks, while the reference design does not do any alignment.

Table III
POWER CONSUMPTION

| chip power | transmission power | total power | power density |
|---|---|---|---|
| 10.3 $\mu W$ | 4.32 $\mu W$ | 14.6 $\mu W$ | 189.6 $\mu W/mm^2$ |

The power consumption is presented in Table III. A typical sampling rate of raw spike signals is 25K samples per second. We use 16 bits to represent a data, which means that the input data rate is 400 Kbps. A typical spike firing rate of a neuron is 40 spikes per second [5]. We use 32 bits to represent the timestamp of each spike and 4 bits to represent the cluster ID. This means that, by applying spike sorting, the data rate could be reduced to 1.44 Kbps, for a 278 fold reduction compared with the original raw data. The energy spent on wireless transmission is about 3 nJ per bit [12]. The transmission power in our design is therefore about 4.32 $\mu W$. The total chip power is 10.3 $\mu W$.

Since we now have values for both chip area and power consumption, we compute the power density: 189.6 $\mu W/mm^2$. According to Kim et al. [13], in order to make sure that an embedded chip is safe to brain tissue, the power density must be less than 277 $\mu W/mm^2$. Our design clearly meets this requirement.

### C. Accuracy Evaluation

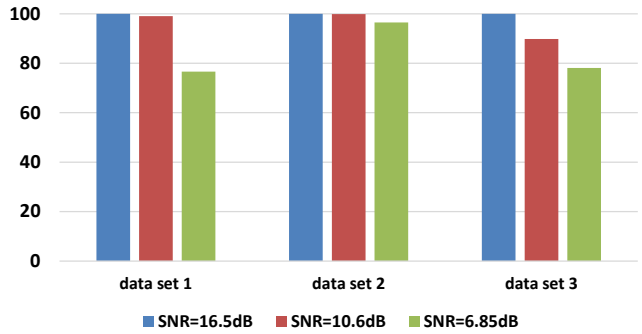


Figure 8. Accuracy

In order to evaluate sorting accuracy, we compare with the reference design [9] using the same data sets [10] and a similar procedure. We selected three data sets obtained from measurements in three different parts of the brain. Each data set was sampled under three different noise levels (in the archive). These are run separately for a total of nine combinations. We run each combination 5 times; and the average accuracy of each combination is displayed in Figure 8. The accuracy degrades with SNR from 100% at 16.5dB to about 96% at 10.6dB to about 83% at 6.85dB. The overall geometric mean is 92.9%. In contrast the accuracy of the reference design over a similar noise range is 75%. This is mainly because of the more sophisticated spike detection algorithm (described above), which ensures that the detected spikes are more complete and accurate.

## V. Conclusion

In this work, we propose a novel spike sorting hardware prototype designed to perform this operation in conjunction with the sensor embedded in the brain. The projected benefit is a massive reduction in transmission power at the cost of increased *in situ* processing. We find that our design reduces transmission power by a factor of 278 while simultaneously meeting the real-time and power-budget constraints. A major difference between our design and previous efforts is implementation of a more complex spike sorting algorithm. This has the effect of improving accuracy from 75% to 93%. This requires some increase in resources, but the overall design remains well within the hard constraints. In future work, we may be able to combine this work with semantically independent methods of reducing transmission as we explored previously [14].

## References


[1] M. Velliste, S. Perel, M. C. Spalding, A. S. Whitford, and A. B. Schwartz, "Cortical control of a prosthetic arm for self-feeding," *Nature*, vol. 453, pp. 1098–1101, 2008.

[2] G. Santhanam, S. Ryu, B. Yu, A. Afshar, and K. Shenoy, "A high performance brain-computer interface," *Nature*, vol. 442, pp. 195–198, 2006.

[3] S. Morishita, K. Sato, H. Watanabe, Y. Nishimura, T. Isa, R. Kato, T. Nakamura, and H. Yokoi, "Brain-Machine Interface to Control a Prosthetic Arm with Monkey ECoGs During Periodic Movements," *Frontiers Neuroscience*, vol. 8, p. 417, 2014.

[4] S. Rajangam, P. Tseng, A. Yin, G. Lehew, D. Schwarz, M. Lebedev, and M. Nicolelis, "Wireless cortical brain-machine interface for whole-body navigation in primates," *Scientific Reports*, vol. 6, 2016.

[5] S. Gibson, "Neural Spike Sorting in Hardware: From Theory to Practice," Ph.D. dissertation, University of Califonia, Los Angeles, 2012.

[6] R. Olsson and K. Wise, "A three dimensional neural recording microsystem with implantable data compression circuitry," *IEEE Journal of Solid-state Circuits*, vol. 40, pp. 2796–2804, 2005.

[7] V. Karkare, S. Gibson, and D. Markovic, "A 130-uW, 64-Channel Neural Spike-Sorting DSP Chip," *IEEE Journal of Solid-state Circuits*, vol. 48, pp. 1214–1222, 2013.

[8] M. Chae, Z. Yang, M. Yuce, L. Hoang, and W. Liu, "A 128-Channel 6 mW Wireless Neural Recording IC With Spike Feature Extraction and UWB Transmitter," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 17, pp. 312–321, 2009.

[9] V. Karkare, S. Gibson, and D. Markovic, "A 75-W, 16-Channel Neural Spike-Sorting Processor With Unsupervised Clustering," *IEEE Journal of Solid-state Circuits*, vol. 48, pp. 2230–2238, 2013.

[10] U. Rutishauser, E. Schuman, and A. Mamelak, "Online detection and sorting of extracellularly recorded action potentials in human medial temporal lobe recordings, in vivo," *Journal of Neuroscience Methods*, vol. 154, pp. 204–224, 2006.

[11] Cadence. (2013) Encounter Power System Datasheet. Cadence.

[12] F. Chen, A. Chandrakasan, V. Stojanovic, and B. Rao, "Design and Analysis of a Hardware-Efficient Compressed Sensing Architecture for Data Compression in Wireless Sensors," *Solid-State Circuits, IEEE Journal of*, vol. 47, no. 3, pp. 744–756, 2012.

[13] S. Kim, P. Tathireddy, R. A. Normann, and F. Solzbacher, "Thermal impact of an active 3-D microelectrode array implanted in the brain," *Neural Syst. Rehabil. Eng., IEEE Transaction on*, vol. 15, no. 4, pp. 493–501, 2007.

[14] J. Sheng, C. Yang, and M. Herbordt, "Hardware-Efficient Compressed Sensing Encoder Designs for WBSNs," in *Proc. IEEE Conf. High Performance Extreme Computing*, 2015.