Title
Neural Spike Sorting in Hardware: From Theory to Practice

Author
Gibson, Sarah Paige

Publication Date
2012

University of California

Los Angeles

Neural Spike Sorting in Hardware: From Theory to Practice

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical Engineering

by

Sarah Paige Gibson

2012

Abstract of the Dissertation

Neural Spike Sorting in Hardware: From Theory to Practice

by

Sarah Paige Gibson

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2012

Professor Dejan Marković, Chair

Brain–machine interfaces require real-time, wireless signal acquisition systems. However, wireless transmission of raw data is impossible for high-channel-count systems given the power constraints. Data rates could be reduced, thereby enabling wireless data transmission, by performing spike sorting—mapping each recorded action potential to the neuron that generated it—on a DSP at the recording site and transmitting only the sorting results. Our first objective was to design such a DSP. We first developed a standardized dataset and methodology in order to perform an extensive, unbiased comparison of published spike-sorting algorithms to determine which would be most appropriate for hardware implementation. We then considered various implementation issues, such as whether analog or digital spike detection is more efficient and how best to quantize neural signals. This work led to two low-power digital spike-sorting chips.

Our second objective was to provide an offline solution for the research setting that would accelerate the processing of data that has already been recorded using conventional data-acquisition systems. Here, we present an FPGA-based spike-sorting platform that can increase the speed of offline spike sorting by at least 25 times, effectively reducing the time required to sort data from long experiments from several hours to just a few minutes. We attempted to preserve the flexibility of software by implementing several different algorithms in the design, and by providing user control over parameters such as spike detection thresholds.

The dissertation of Sarah Paige Gibson is approved.

Richard Joseph Staba

Abeer A. H. Alwan

Jack W. Judy

Dejan Marković, Committee Chair

University of California, Los Angeles

2012

Table of Contents

1 Introduction . . . 1

1.1 Electrophysiology and Extracellular Recording . . . 1

1.2 Current State of the Art . . . 3

1.3 Objectives . . . 4

1.3.1 Need for Wireless Recording Systems . . . 5

1.3.2 Need for Hardware-Accelerated Data Processing . . . 7

2 Spike Sorting . . . 8

2.1 Signal Composition . . . 8

2.1.1 Unit Activity . . . 8

2.1.2 Local Field Potential (LFP) . . . 12

2.2 The Spike-Sorting Process . . . 13

2.3 Classifying Spike-Sorting Algorithms . . . 14

2.4 Overview of Spike-Sorting Algorithms . . . 16

2.4.1 Detection . . . 16

2.4.2 Alignment . . . 19

2.4.3 Feature Extraction . . . 21

2.4.4 Dimensionality Reduction . . . 24

2.4.5 Clustering . . . 27

2.4.6 Alternative Methods for Spike Sorting . . . 34

2.5 Challenges in Spike Sorting . . . 37

2.5.1 No Ground Truth . . . 37

2.5.2 Non-Gaussian Noise . . . 40

2.5.3 Non-Stationarities 40

2.5.4 Overlapping Spikes 41

2.6 Single vs. Multichannel Signal Processing 41

2.7 Controversy 42

2.8 Conclusion 45

3 Algorithm Evaluations 47

3.1 Introduction 47

3.2 Test Data 48

3.3 Testing Methodology 49

3.3.1 Accuracy Calculations 49

3.3.2 Receiver Operating Characteristic (ROC) 53

3.3.3 Complexity Calculations 53

3.3.4 Assumptions 55

3.4 Algorithms 56

3.5 Results 57

3.5.1 Spike Detection 57

3.5.2 Feature Extraction 58

3.5.3 Dimensionality Reduction 60

3.6 Discussion 63

3.7 Conclusion 64

4 Analog vs. Digital Spike Detection 66

4.1 Introduction 66

4.2 Methods 67

4.2.1 Algorithms 67

4.2.2 Analog-to-Digital Converter (ADC) 68

4.2.3 Analog Spike Detection 69

4.2.4 Effect of SNR and Firing Rate on Analog Detection Power 74

4.2.5 Digital Spike Detection 75

4.3 Results 78

4.4 Conclusion 78

5 The Effects of Quantization on Spike Sorting 81

5.1 Introduction 81

5.2 Spike-Sorting Algorithms 82

5.3 Quantization 84

5.3.1 Detection 87

5.3.2 Clustering 87

5.4 Simulation Results 88

5.5 Conclusion 92

6 Spike-Sorting ASICs 94

6.1 Spike-Sorting Chip 94

6.2 Osort Chip 95

6.2.1 Clustering Algorithm Evaluations 97

6.2.2 Chip Results 98

7 FPGA-Based Spike-Sorting Platform 103

7.1 Introduction 103

7.1.1 Motivation 103

7.1.2 Proposed Solution 104

7.1.3 Design Features 105

7.2 Functionality of Current Prototype 106

7.2.1 Inputs 106

7.2.2 Processing 106

7.2.3 Outputs 106

7.2.4 Modes of Operation 107

7.2.5 GUI 107

7.3 Implementation 107

7.3.1 Overview 107

7.3.2 Targeted Hardware 109

7.3.3 FPGA Implementation 110

7.3.4 Python Wrapper 116

7.3.5 MATLAB Wrapper 118

7.4 Performance 118

7.4.1 Accuracy 118

7.4.2 Processing Speed 118

7.4.3 Timing Analysis 120

7.5 Tool Availability 121

7.6 Conclusion 121

7.7 Acknowledgments 121

8 Conclusion 123

8.1 Contributions 123

8.2 Looking to the Future 124

References 126

List of Figures

1.1 Example of an electrical signal recorded from a microelectrode 3

1.2 Block diagram of the conventional neural recording system. 3

1.3 Screenshot of a commercial spike-sorting program 4

1.4 Example of an experimental setup for neural recordings from rat 6

1.5 Block diagram of the proposed wireless neural recording system 6

2.1 *Left*: Examples of intracellular action potential and corresponding extracellular spike. *Right*: Diagram of a neuron 10

2.2 Electrical circuit models of extracellular recording 11

2.3 The signal-processing chain used to obtain single-unit activity 14

2.4 Examples of pre-emphasized signals and threshold values for three different detection methods 17

2.5 Examples of two different alignment methods 20

2.6 Sample results of feature extraction using PCA 22

2.7 Example showing the DWT of raw data 23

2.8 Illustration of feature extraction using the integral transform 23

2.9 Illustration of how feature distribution information can be used in dimensionality reduction 25

2.10 Example showing the calculation of the “dip” statistic for Hartigan’s Dip Test 27

2.11 Example execution of the Maximum-Difference Test 28

2.12 Example of results of manual cluster cutting on PCA features 29

2.13 Example results from clustering 30 seconds of real human data using three different clustering methods 35

2.14 Example of an MUA signal, compared to the raw signal, unit activity, and LFP 44

3.1 Signal-processing chain used to evaluate algorithm accuracies 50

3.2 Average spike waveforms used as templates in the test datasets 51

3.3 Median ROC curve for each detection method 58

3.4 Histograms of choice probabilities for each detection method 59

3.5 *Left*: Probability of detection (P_{D}) vs. SNR and *Right*: probability of false alarm (P_{FA}) vs. SNR for each detection method 59

3.6 Accuracy of all datasets and noise levels vs. normalized computational cost for each spike-detection, feature-extraction, and dimensionality-reduction algorithm 60

3.7 Mean classification accuracy of each dimensionality-reduction algorithm . . . 62

3.8 Mean classification accuracy of each dimensionality-reduction algorithm when using (a) DWT feature extraction and (b) DD feature extraction 62

3.9 Pie charts indicating total system complexity: (a) OPS, (b) area 63

3.10 Example of data-rate reduction provided by each step of the spike-sorting process 64

4.1 Block diagram for (a) digital spike detection and (b) analog spike detection . 67

4.2 Outputs of spike detector for each mode of operation 68

4.3 Performance of recent ADCs, and FoM contours 70

4.4 Schematic for a low-power dynamic comparator 71

4.5 Implementation of the Nonlinear Energy Operator algorithm in the discrete-time analog domain 72

4.6 Implementation of analog memory for buffering of the signal before a spike has been detected 73

4.7 Variation in power of analog NEO spike detection due to changes in SNR and neuronal firing rates 75

4.8 Power estimates obtained from Synopsys for NEO 77

4.9 Area estimates for NEO as a function of the number of channels interleaved 77

4.10 Power per channel and area per channel for each algorithm and output mode 79

5.1 Block diagrams for two potential recording systems 82

5.2 Plot of received signals in 2-dimensional space 84

5.3 Examples of uniform and optimal quantization levels for narrow and wide Gaussian pdfs 86

5.4 *Left*: SQNR vs. resolution for a uniform quantizer compared to the optimal quantizer. *Right*: Input pdfs for spikes and raw data 88

5.5 Example of a synthetic dataset composed of spikes from two neurons 89

5.6 Comparison of detection and classification results when a uniform vs. an optimal quantizer is used 91

5.7 Example of the benefits to clustering of using an optimal quantizer 92

6.1 Micrograph of spike-sorting chip 95

6.2 Sample of spike-sorting chip output 96

6.3 Box plots showing accuracy for each clustering algorithm tested 98

6.4 Sample output from one of the channels processing human neural data . . . 100

6.5 Die photo the 16-channel Osort chip 101

7.1 Spike Sorter GUI 108

7.2 Photograph of the ROACH FPGA processing board 109

7.3 Data-streaming interface 112

7.4 Block diagram of the processing elements on the FPGA 113

7.5 Implementation of a variable-length delay used in spike buffering for arbitrary sampling rates 115

7.6 State diagram of Osort 117

7.7 Speed improvements of this hardware tool over the equivalent software running on three different computers with different processing powers 119

7.8 Breakdown of how various system components contribute to the total time to process one packet. 120

List of Tables

2.1 Summary of Clustering Algorithm Characteristics 35

3.1 Complexity of Spike-Detection Algorithms 58

3.2 Complexity of Feature-Extraction Algorithms 60

3.3 Complexity of Dimensionality-Reduction Algorithms 63

3.4 Overall System Complexity 64

6.1 Spike-Sorting Chip Summary 95

6.2 Comparison with Prior Work 96

6.3 Osort Chip Summary 99

6.4 Comparison with Prior Work 100

7.1 Latency of Clustering Operations 116

7.2 Computer Specifications 119

Acknowledgments

Portions of Chapters 1–2 have been reprinted, with permission, from: S. Gibson, J. W. Judy, and D. Marković, "Spike Sorting: The First Step in Decoding the Brain," *IEEE Signal Process. Mag.*, vol. 29, no. 1, pp. 124–143, Jan. 2012. Copyright © 2012, IEEE. J. W. Judy and D. Marković were the PIs.

Portions of Chapter 3 have been reprinted, with permission, from: S. Gibson, J. W. Judy, and D. Marković, "Technology-Aware Algorithm Design for Neural Spike Detection, Feature Extraction, and Dimensionality Reduction," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 18, no. 4, pp. 469–478, Oct. 2010. Copyright © 2010, IEEE. J. W. Judy and D. Marković were the PIs.

Portions of Chapter 4 have been reprinted, with permission, from: S. Gibson, R. Chandler, V. Karkare, D. Marković, and J. W. Judy, "An Efficiency Comparison of Analog and Digital Spike Detection," in *Proc. 4th Int. IEEE EMBS Conf. Neural Eng.*, Antalya, Turkey, 2009, pp. 423–428. Copyright © 2009, IEEE. R. Chandler provided the analog circuit analysis, and V. Karkare provided the digital circuit analysis. D. Marković and J. W. Judy were the PIs.

Portions of Chapter 5 have been reprinted, with permission, from: S. Gibson, V. Wang, and D. Marković, "Effects of Quantization on Neural Spike Sorting," in *Proc. 2011 IEEE Int. Symp. Circuits Syst.*, Rio de Janeiro, Brazil, 2011, pp. 2099–2102. Copyright © 2011, IEEE. V. Wang proposed the idea of using nonlinear quantization for spike sorting and provided the code for the quantizers. D. Marković was the PI.

Portions of Chapter 6 have been reprinted, with permission, from: V. Karkare, S. Gibson, and D. Marković, "A 130-μW, 64-Channel Neural Spike-Sorting DSP Chip," *IEEE J. Solid-State Circuits*, vol. 46, no. 5, pp. 1214–1222, May 2011. Copyright © 2011, IEEE. V. Karkare performed all chip design. D. Marković was the PI.

Other portions of Chapter 6 have been reprinted, with permission, from: V. Karkare, S. Gibson, C.-H. Yang, H. Chen, D. Markovic, "A 75μW, 16-Channel Neural Spike-Sorting

Processor with Unsupervised Clustering," in *Proc. Int. Symp. on VLSI Circuits*, Kyoto, Japan, 2011, pp. 252–253. Copyright © 2011, IEEE. V. Karkare performed all chip design. C.-H. Yang contributed to the memory-bank design. H. Chen helped with chip testing. D. Marković was the PI.

Vita

1982 Born, Houston, TX.

1997 – 2001 Alief Hastings High School, Houston, TX.

2005 B.S., Electrical and Computer Engineering,
Baylor University, Waco, TX.

2006 Mars Flight Project Intern, Jet Propulsion Laboratory, Pasadena, CA.
HiRISE instrument team, Mars Reconnaisance Orbiter.

2006 – 2007 Graduate Fellowship, Department of Electrical Engineering,
University of California, Los Angeles.

2007 Intern, Northrop Grumman Space Technology, Redondo Beach, CA.
Systems Engineering, James Webb Space Telescope.

2007 – 2012 Graduate Student Researcher, Department of Electrical Engineering,
University of California, Los Angeles.

2008 Teaching Assistant, Signals and Systems,
University of California, Los Angeles.

2008 M.S., Electrical Engineering,
University of California, Los Angeles.

2009 Intern, Sigenics Inc., Sierra Madre, CA.
Systems Engineering, wireless telemetry for neural recording devices.

2010 – 2012 Writing Consultant, Graduate Writing Center,
University of California, Los Angeles.

Publications

S. Gibson, J. W. Judy, and D. Markovic, "Spike Sorting: The First Step in Decoding the Brain," *IEEE Signal Process. Mag.*, vol. 29, no. 1, pp. 124–143, Jan. 2012.

V. Karkare, S. Gibson, C.-H. Yang, H. Chen, D. Markovic, "A 75μW, 16-Channel Neural Spike-Sorting Processor with Unsupervised Clustering," in *Proc. Int. Symp. on VLSI Circuits* (VLSI'11), Kyoto, Japan, 2011, pp. 252–253.

Z. Charbiwala, V. Karkare, S. Gibson, D. Markovic, and M. Srivastava, "Compressive Sensing of Neural Action Potentials Using a Learned Union of Supports," in *Proc. Int. Conf. on Body Sensors Networks* (BSN'11), Dallas, Texas, 2011, pp. 53–58.

S. Gibson, V. Wang, and D. Markovic, "Effects of Quantization on Neural Spike Sorting," in *Proc. 2011 IEEE Int. Symp. Circuits Syst.* (ISCAS'11), Rio de Janeiro, Brazil, 2011, pp. 2099–2102.

V. Karkare, S. Gibson, and D. Markovic, "A 130-μW, 64-Channel Neural Spike-Sorting DSP Chip," *IEEE J. Solid-State Circuits*, vol. 46, no. 5, pp. 1214–1222, May 2011.

S. Gibson, J. W. Judy, and D. Markovic, "Technology-Aware Algorithm Design for Neural Spike Detection, Feature Extraction, and Dimensionality Reduction," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 18, no. 4, pp. 469–478, Oct. 2010.

V. Karkare, S. Gibson, and D. Markovic, "A 130-μW, 64-Channel Spike-Sorting DSP Chip," in *IEEE Asian Solid-State Circuits Conf.* (ASSCC'09), Taipei, Taiwan, 2009, pp. 289–292.

R. Chandler, S. Gibson, V. Karkare, S. Farshchi, D. Markovic, and J. W. Judy, "A System-Level View of Optimizing High-Channel-Count Wireless Biosignal Telemetry," in *Proc. Int. IEEE Engineering in Medicine and Biology Conf.* (EMBC'09), Minneapolis, Minnesota, 2009, pp. 5525–2230.

S. Gibson, R. Chandler, V. Karkare, D. Markovic, and J. W. Judy, "An Efficiency Comparison of Analog and Digital Spike Detection," in *Proc. 4th Int. IEEE EMBS Conf. Neural Eng.* (NER'09), Antalya, Turkey, 2009, pp. 423–428.

S. Gibson, J. W. Judy, and D. Markovic, "Comparison of Spike-Sorting Algorithms for Future Hardware Implementation," in *Proc. Int. IEEE Engineering in Medicine and Biology Conf.* (EMBC'08), Vancouver, Canada, 2008, pp. 5015–5020.

CHAPTER 1

Introduction

1.1 Electrophysiology and Extracellular Recording

For centuries, scientists have been using electrophysiology to study the electrical properties of biological cells and tissues. In 1791 Luigi Galvani discovered that he could induce contraction in a frog leg muscle by applying an electric current [1]. In 1952 Hodgkin and Huxley, using an experimental technique they developed called the "voltage clamp," made a number of groundbreaking discoveries on the movement of ions across the membranes of nerve cells during action potential generation for which they eventually received a Nobel Prize [2–5]. In 1977 Hubel and Wiesel (also Nobel Prize recipients) used electrophysiological recordings to provide the first information about how the activity of individual neurons contributes to higher visual processing [6].

Electrophysiological recordings can be made from within cells (*intracellular*) or from outside cells (*extracellular*). In studies of the central nervous system, small-diameter electrodes can be positioned in the extracellular space to record electrical signals from surrounding neurons (Fig. 1.1). These electrodes are able to detect action potentials from individual neurons. The ability of extracellular recording to provide researchers with neuron-level activity combined with its relatively low level of difficulty to perform (as compared to intracellular recording, for example) has led extracellular recording to become one of the dominant experimental techniques in neuroscience research. For example, there has been a movement towards studying not only individual neurons but networks of neurons in order to understand how the activity of interconnected neurons results in higher-order functions such as perception, understanding, movement, and memory. Such studies require extracellular recording from ensembles of neurons using multichannel electrode arrays. In *Methods for Neural Ensemble Recordings*, contributing authors Sameshima and Baccalá go so far as to claim that "extracellular recordings are the only practical choice in experiments that intend to establish correlations between neural ensemble responses and behaviors involving awake animals" [7].

Electrophysiology is also used in clinical settings. For example, in presurgical patients with severe medically intractable epilepsy, electrophysiological recordings from depth electrodes placed inside the brain are used to localize brain areas where seizures begin. These larger electrodes mainly record electroencephalogram (EEG) signals, but often microelectrodes are implanted as well for use in research (e.g., [8–10]), since single-unit activity provides greater detail on changes in signal transmission that could distinguish normal from abnormal activity. And over the past decade, the technique of extracellular recording has received additional attention as researchers have begun to tap into its potential use in medical technologies for the treatment of disorders such as paralysis [11, 12], epilepsy [11], and even cognitive and memory loss [13]. Many of these technologies are based upon the idea of brain–machine interfaces (BMIs), in which implanted electronics record and decode brain signals that can be used to control "machines" such as computers or prosthetic limbs.

Whether the application is basic science research, clinical diagnostics, or medical technology, the signals from individual neurons ("single-unit activity") are often of particular interest. In basic science, for example, the researcher may require knowledge of single-unit activity in order to study how a type of neuron responds to a specific stimulus or how the activities of different neurons are correlated. Similarly, most neural prosthetic technologies employ some sort of "decoding" algorithm—which may decode movement [11, 12, 14], intentions [15], or memories [13]—that typically operates on signals from individual neurons. But because of the relatively large sizes of recording electrodes[1], the recorded signal is the sum of the signals from several (usually 2–10) neurons surrounding the electrode ("multiunit activity," illustrated in Fig. 1.1). In such cases, *spike sorting*, the process of separating multiunit activity into groups of single-unit activity, is necessary.

[1] Microwire electrode tips used in extracellular recording typically have diameters of 13 to 80 μm [16]. The Cyberkinetics implementation of the popular Utah silicon microelectrode array has conical electrode tips with lengths of about 40 μm and surface areas of about 1600 μm^2[17].

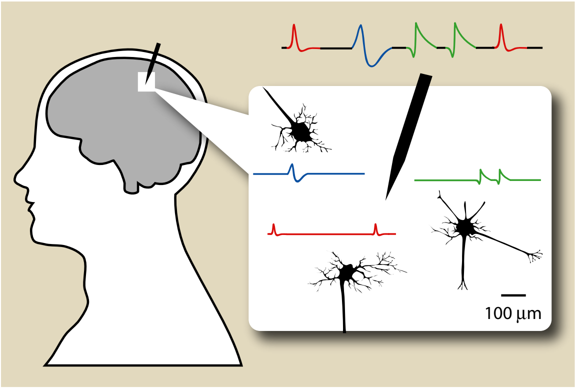


Figure 1.1: The electrical signal recorded from a microelectrode is the sum of the postsynaptic and action-potential activity of many neurons in the surrounding area.

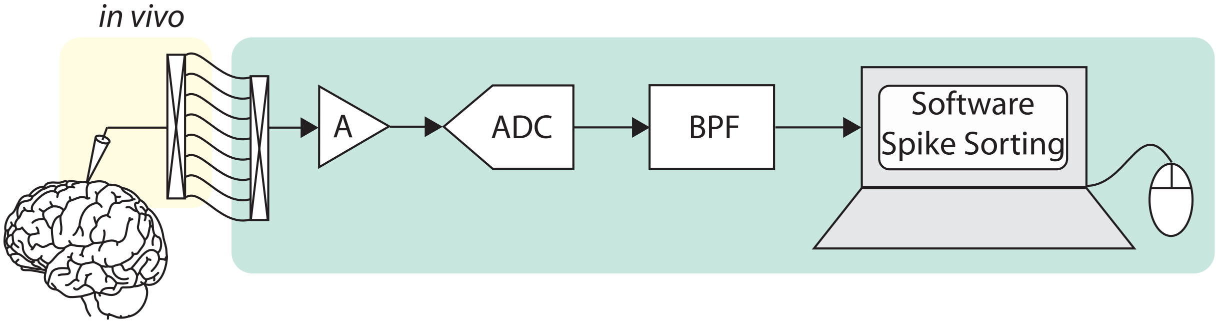
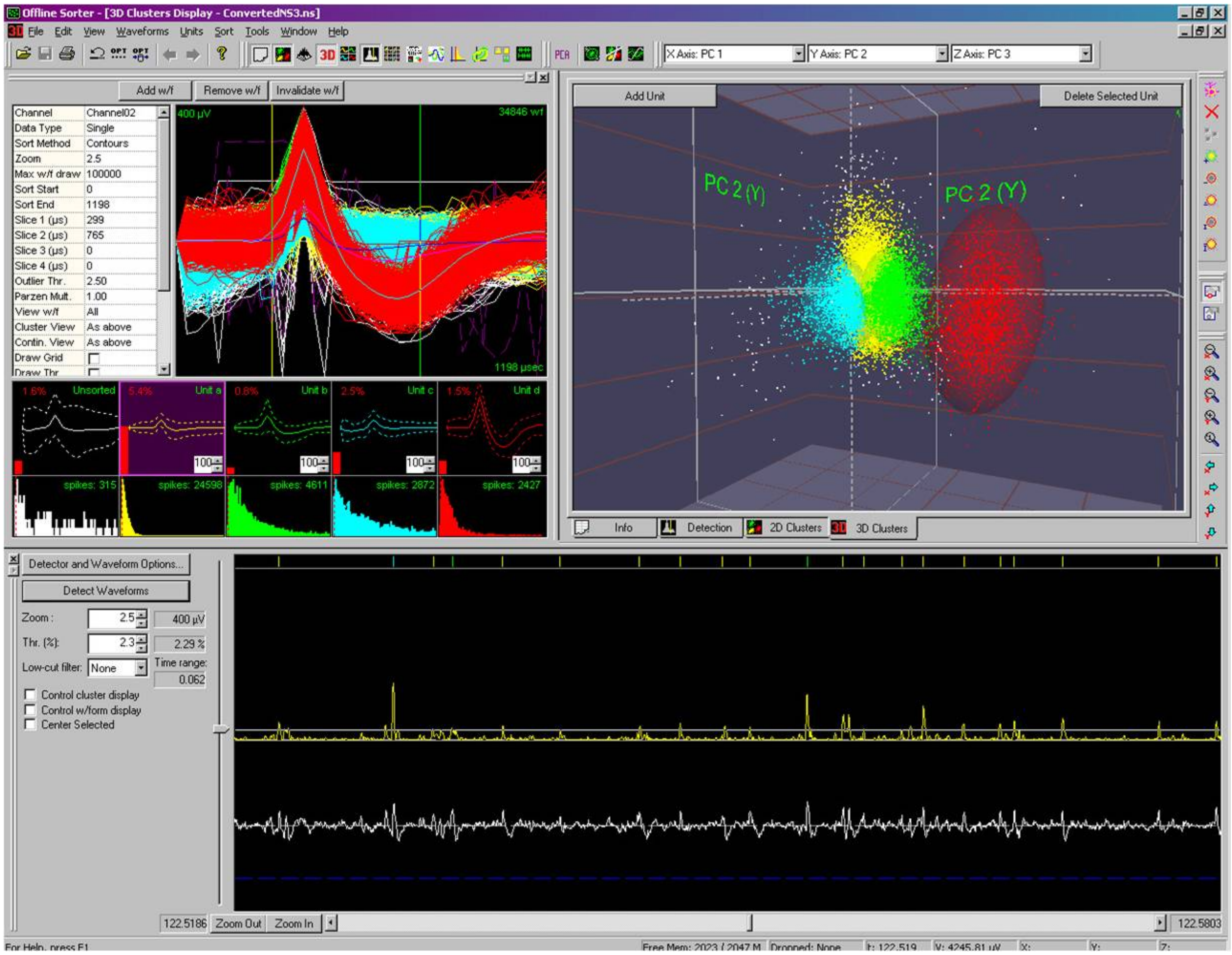


Figure 1.2: Block diagram of the conventional neural recording system.

1.2 Current State of the Art

In a traditional neural recording system, shown in Fig. 1.2, electrodes provide the direct interface to the brain, and the unamplified raw data is sent outside the body through wires to the rest of the data-acquisition hardware. The raw data is recorded onto computer hard disks. All data processing, including spike sorting, is performed offline, in software.

An example of a commercial spike-sorting software package is shown in Fig. 1.3. The yellow waveform in the bottom plot shows the energy of the recorded signal. The user must set a detection threshold by manually adjusting the position of the white horizontal line. Once spikes are detected, the user can choose to plot certain features, such as the spike heights and widths or principal components (shown in the upper right plot), in two or three dimensions. The operator can then use built-in automatic or semi-automatic clustering routines. The results of clustering are then displayed on the left-most plots. Statistics about



Channel	Channel02
Data Type	Single
Sort Method	Contours
Zoom	2.5
Max w/f draw	100000
Sort Start	0
Sort End	1198
Slice 1 (µs)	299
Slice 2 (µs)	765
Slice 3 (µs)	0
Slice 4 (µs)	0
Outlier Thr.	2.50
Parzen Mult.	1.00
View w/f	All
Cluster View	As above
Contin. View	As above
Draw Grid	□
Draw Thr	□

Zoom : 2.5 400 µV
Thr. (%): 2.3 2.29 %
Low-cut filter: None
Time range: 0.062
□ Control cluster display
□ Control w/form display
□ Center Selected

Figure 1.3: Screenshot of Offline Sorter, a commercial spike-sorting program by Plexon [18].

each cluster, such as the mean waveform, variance, and interspike interval (ISI) histograms, are given so that the operator can evaluate the clustering results and make manual changes, such as combining two clusters into one, if needed.

1.3 Objectives

There are several problems with the state of the art in neural recording, which led to the two main objectives of this thesis:

1.3.1 Need for Wireless Recording Systems

In the current setup, data is transferred from the subject to a computer via thick cables. In the research setting, cables restrict the physical movement of subjects (Fig. 1.4), thereby limiting the quality and diversity of experiments that can be performed. Furthermore, these cables can increase the severity of noise and motion artifacts seen in the recording. Cables are obviously unwanted in BMI applications as well, where the goal is for patients to be able to lead normal lives—meaning they cannot be tethered to electronic equipment. Thus, there is a clear need for wireless recording systems.

Such a wireless recording system would presumably include an on-site radio to wirelessly transmit data off-site. However, the high-channel-count systems of today incur high data rates. Consider the example of a 100-channel system using a sampling rate of 30 kSa/s and a resolution of 10 bits, which would produce data at 30 Mbps [19]. Wireless transmission at this data rate would require about 600 mW of power[2]. This is problematic for two reasons. First, such high power consumption would result in a battery life of only about 40 minutes[3], which is acceptable neither for experiments that require days of continuous recordings nor for BMIs, which cannot afford such frequent battery changes. Second, such high power dissipation could be dangerous for the surrounding tissue [21].

The alternative recording setup shown in Fig. 1.5 would solve this problem: a digital signal processor (DSP) would perform on-site, real-time spike sorting and only the sorting results would be transmitted, thereby achieving enough data reduction to enable the wireless transmission of data. To continue the above example, assume that this new system transmits only 11 bits per spike (4 bits for the cluster ID and 7 bits for the channel number). A spike rate of 100 spikes/s would result in an output data rate of 110 kbps, a transmitter power of 2.25 mW, and a battery life of over 7 days (a 272x improvement). **The first objective of this thesis was, therefore, to design the DSP for this wireless recording system.**

Towards this objective, we first surveyed the spike-sorting algorithms already published

[2] assuming the transmitter in [20], which was designed for a 100-channel wireless neural recording system; $20.45\ \text{nJ/b} \cdot 30\text{Mb/s} = 613.5\ \text{mW}$ at 13 cm.

[3] assuming an Energizer CR1632, with a capacity of 130 mAh at 3 V = 390 mWh; battery life is $390\ \text{mWh} \div 613.5\ \text{mW} = 38$ min.

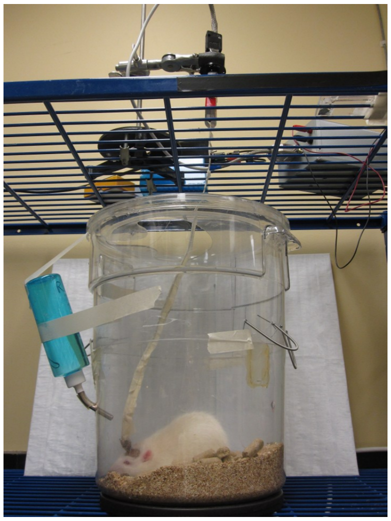


Figure 1.4: Example of an experimental setup for neural recordings from rat. The rat has been implanted with electrodes, and a thick bundle of wires delivers the recorded signals to the data-acquisition equipment. A commutator (seen above the cage) must be used to prevent the rat's movement from applying a torque to the headstage, which could break the delicate wires.

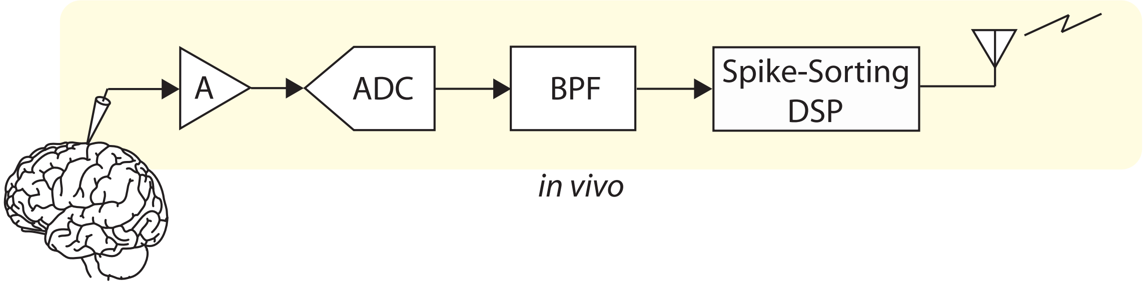


Figure 1.5: Block diagram of the proposed wireless neural recording system.

in the field (Chapter 2) and performed an extensive, unbiased algorithm comparison to determine which are most appropriate for hardware implementation (Chapter 3). We then considered various implementation issues, such as whether analog or digital spike detection is more efficient (Chapter 4) and what the best method is for quantizating neural signals (Chapter 5). Examples of digital chips that have been designed based on this work will be presented in Chapter 6.

1.3.2 Need for Hardware-Accelerated Data Processing

For the time being, many researchers are still working with wired systems and the data that has already been recorded using such systems. This data is usually acquired with sampling rates of 20–30 kHz and resolutions of 12–24 bits per sample [22–27] for 64–128 channels simultaneously. Consider a human epilepsy study in which 64 channels of data are sampled at 27.777 kHz and quantized to 16 bits. An 8-hour experiment would accumulate about 100 GB of data, which would require about 30 hours to sort using conventional software tools[4] (processing rate of 0.94 MBps). Now consider dedicated hardware running at 100 MHz (200 MBps); the processing time would be reduced from 30 hours to 8.5 minutes (212x).

Thus, the second objective of this thesis was to develop a hardware spike-sorting tool for accelerating the offline processing of existing neural data. In order to provide a good compromise between speed and flexibility, we proposed an FPGA-based spike-sorting platform. A library of algorithms was developed for each step in the spike-sorting process. That way, rather than being tied down to one particular spike-sorting algorithm, users are able to customize data processing to fit their needs. This library may also be used to generate RTL for future ASIC implementations. This work will be described in Chapter 7.

[4] Osort software package [28] running in Windows on an Intel Core2 Duo Processor.

CHAPTER 2

Spike Sorting

2.1 Signal Composition

Let us begin by looking at the composition of the signals that are recorded by extracellular electrodes. These signals are usually pre-lowpass-filtered in the analog domain and then sampled at a rate of 20–30 kHz. Different cellular mechanisms are responsible for different frequency components of the recorded signals. The high-frequency content (about 300 to 6000 Hz) is referred to as *unit activity*, while the low-frequency signal content (below about 600 Hz) is referred to as *local field potential*.

2.1.1 Unit Activity

If unit activity is the signal of interest, as it is in this thesis, then the sampled signal is bandpass-filtered with a low cutoff frequency of 100 to 300 Hz and a high cutoff frequency of 3000 to 10 000 Hz. As indicated by its name, the source of this "unit activity" is action potentials from individual neurons.

On some level, the action potential can be thought of as the discrete, binary event by which neurons communicate. Much of what we know about the mechanism was discovered in the seminal works of Hodgkin and Huxley [2–5]. Neuronal membranes are impermeable to charged ions except at sites of ligand- and voltage-gated channels, which allow the passage of charged ions between the intra- and extra-cellular space. When the ion channels are closed or inactive, the concentrations of potassium (K^+) and chloride (Cl^-) ions inside the cell are high relative to oustide the cell, while the concentration of sodium (Na^+) ions is high outside the cell relative to inside the cell. At rest, the cell membrane potential, defined with respect to the inside of the cell, is about -70 mV. When a cell's membrane is depolarized, for example by excitatory synaptic input, this depolarization "activates" (opens) the Na^+ channels, causing Na^+ ions to rush *into* the cell along the concentration gradient. This influx of Na^+ causes the membrane to become even more depolarized, consequently causing more Na^+ channels to become activated. Eventually the membrane potential reaches threshold, at which point external input is no longer needed to depolarize the cell, and the positive feedback caused by the Na^+ current continues the depolarization at an even faster rate. This sharp influx of Na^+ into the cell results in the rising phase of the action potential shown in Fig. 2.1 (left–top). Once the cell reaches a peak depolarization of about 40 mV, two things happen: the Na^+ channels become "inactivated" such that no more Na^+ ions can pass through, and the voltage-gated K^+ channels open. Now, K^+ ions flow *out* of the cell along the concentration gradient, and the cell membrane begins to "hyperpolarize"; this efflux of K^+ results in the falling phase of the action potential (Fig. 2.1, left–top). This hyperpolarization continues until the cell has returned to its resting potential. In some cases, the hyperpolarization is followed by a slow after-hyperpolarization, where the resting potential is overshot, before the membrane potential returns to rest. The action potential usually begins at the axon hillock, near the cell body (soma), and propagates down the axon (Fig. 2.1, right). Depolarization of the axon terminal then triggers the release of neurotransmitters into the synaptic cleft (the gap between the pre-synaptic cell's axon terminal and the post-synaptic cell's dendrite), in turn depolarizing the post-synaptic cell. It is in this way that neurons communicate with each other.

Extracellularly recorded action potentials are called *spikes*. (The terms "action potential" and "spike" are sometimes used interchangeably; to be precise, however, we will use "action potential" to refer to the intracellular event and "spike" to refer to the captured extracellular waveform.) As shown in Fig. 2.1 (left–bottom), a spike looks slightly different from an intracellular action potential. First, because the recording electrode is placed outside of the cell rather than inside the cell, the polarity is reversed. Second, the filtering properties of the extracellular medium result in an extracellular signal that is about two to three orders of magnitude smaller than the corresponding intracellular signal ($\sim$10 to 100 μV compared

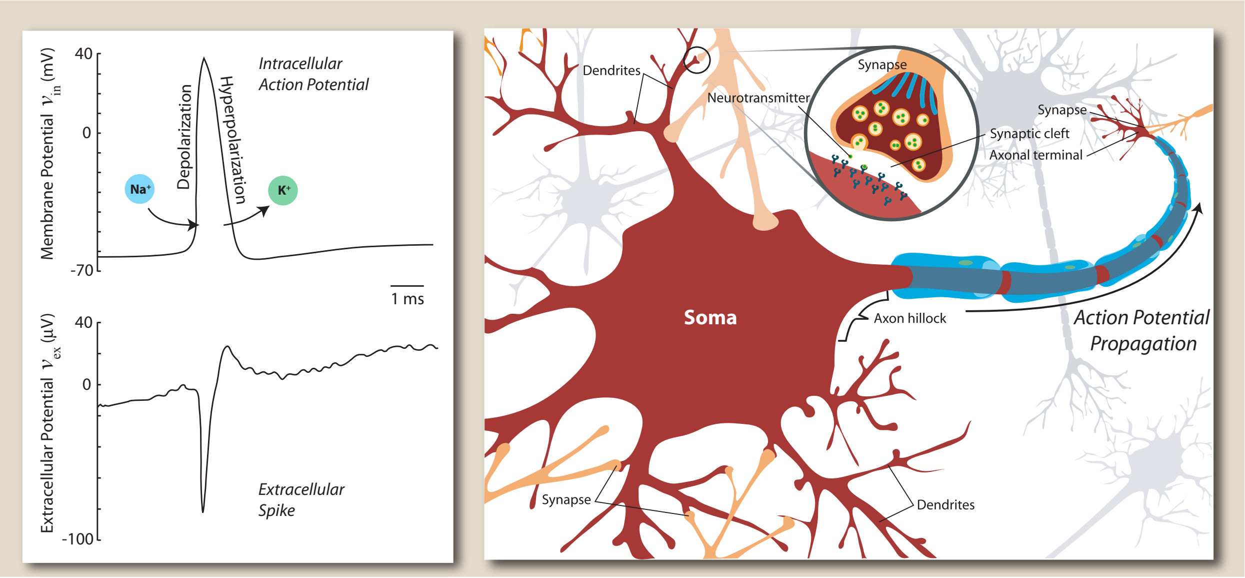


Figure 2.1: *Left*: Adapted from [29]. *Left, Top*: Illustration of the change in membrane potential during a typical action potential. *Left, Bottom*: Illustration of the corresponding change in potential as seen by an *extracellular* electrode. Note the difference in vertical-axis scales. *Right*: Diagram of a neuron. Action potentials begin at the axon hillock and propagate down the axon. Depolarization of the axon terminal then triggers the release of neurotransmitters into the synaptic cleft, in turn depolarizing the post-synaptic cell.

to ∼10 mV). Third, because the membrane acts like a resistor and capacitor in parallel, that is, a highpass filter (Fig. 2.2a), the recorded extracellular potential is approximately equal to the derivative of the intracellular potential [30].

The shape of the intracellular action potential depends on a number of cell properties, including the cell type, the cell geometry, and the ion-channel distribution. This shape is generally considered to be constant for a given neuron, except in special cases such as "burst" (high-frequency) firing. Since the extracellular waveform is directly related to the intracellular waveform, the extracellular spike shape also depends on these properties, as well as on the position of the recording electrode relative to the cell, on the distance of the electrode from the cell, and on interference from other nearby neurons ("background noise"). This "biological noise" is the largest source of noise in a neural recording, having amplitudes approaching that of the unit activity. But the recording hardware itself, including the electrode, the amplifier, and the ADC, also adds a significant amount of noise, the scale of which is largely dependent on the given circuit implementation. It is usually assumed that

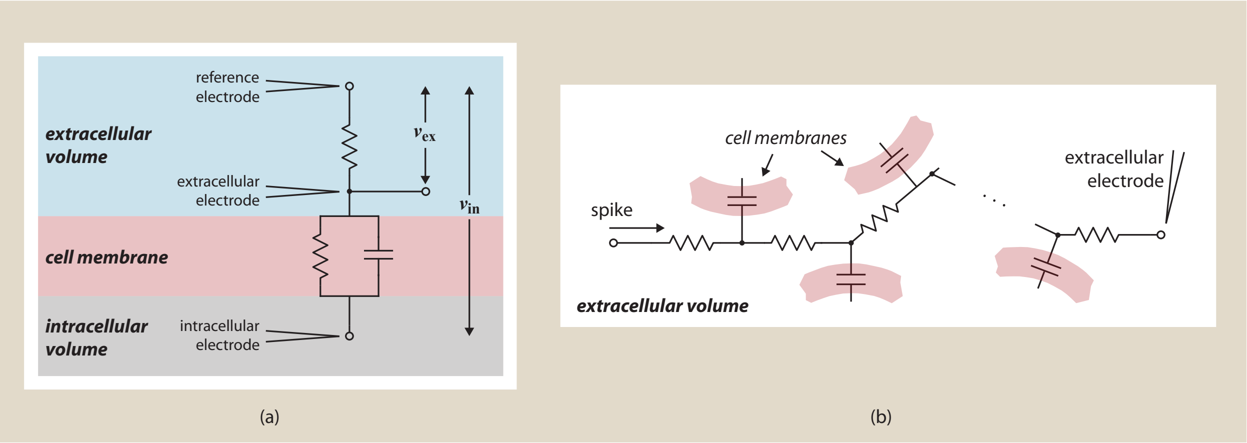


Figure 2.2: (a): A simple electrical-circuit model of extracellular recording [31]. Assume that an intracellular electrode is placed inside the cell, an extracellular electrode is placed outside but near the cell, and the reference electrode is placed very far away from the cell. The extracellular material can be modeled as pure resistance, while the cell membrane can be modeled as a resistor and a capacitor in parallel. The cell membrane highpass-filters the action potential signal, such that the extracellulary recorded signal (v_{ex}) is approximately equal to the derivative of the intracellularly recorded signal (v_{in}) [30]. (b): As the broadband signal passes through the extracellular medium, the capacitive membranes of nearby cells attenuate its high-frequency components.

the signal and the noise are statistically independent and that they sum linearly. Thus, in a given recording session where the electrode placement is assumed to be constant relative to the tissue, we assume that the extracellular spike shape for each neuron can be modeled as a deterministic waveform plus random noise. Note that while the recording noise usually is Gaussian, the background noise typically is not [32].

Spike trains can be treated as point processes with arrival times following a Poisson distribution. A neuron's *firing rate*, the frequency at which it generates action potentials, depends on the cell type and brain area. Neurons in the visual cortex, for example, which are either silent or firing at a base frequency of around 5 spikes per second (or simply Hz), respond to their preferred stimulus with firing rates of about 15–75 Hz [33]. A bursting neuron, on the other hand, can fire as many as 300 to 800 spikes per second [34].

There is disagreement within the neuroscience community as to exactly how information is encoded in the brain. The dominant theory contends that information is encoded by the *frequency* of action potentials ("rate coding") [35, 36]. Alternative theories propose that information is contained in the absolute *timing* of action potentials ("temporal coding") [36, 37] or in the *phase* of action potentials relative to specific oscillations ("phase coding") [38, 39]. No matter which of these theories is true, it is clear that the action potential is the main player in encoding, and that precise timing of action potentials must be obtained in order to "decode" neural activity.

2.1.2 Local Field Potential (LFP)

While LFPs are not the focus of this thesis, a brief discussion of their properties is warranted here. We will also refer to these signals again later when we discuss alternative methods of decoding neural signals for BMIs.

A comprehensive description of the physiological basis for LFP was provided by Buzsáki and Traub in [40]. To summarize, LFPs come from several sources, the most significant of which is synaptic activity. Because the capacitive lipid membranes of cells in the brain act as a lowpass filter, the high-frequency components of neuronal signals are greatly attenuated as they travel through the extracellular medium; equivalently, slow signals are able to propagate much farther than are high-frequency signals (Fig. 2.2b). As a result, the low-frequency component of the signal recorded at any given point within the brain is a linear sum of the activity from large populations of cells. Thus, LFP can be interpreted as an indication of the "cooperative actions" of neurons.

Note that this signal is referred to as "LFP" when recorded by a microelectrode inserted into the brain (hence the "local" in LFP) but as "EEG" if recorded using scalp electrodes and as "electrocorticography (ECoG)" if recorded using epidural or subdural grid electrodes. Some scientists favor using ECoG and EEG because these methods are less invasive and easier to perform. However, because LFPs must propagate through a capacitive medium on their way to these recording sites, EEG and ECoG are actually "spatially smoothed" versions of the LFP. As such, EEG and ECoG contain very little information about the activity of the neurons that actually generate the signals. Furthermore, scalp and dural grid electrodes are only sensitive to signals originating in the superficial layers of the cortex; contributions to the signal from neurons in deeper layers of the cortex and from subcortical areas are virtually negligible. Thus, extracellular recording, which provides the experimenter with LFP measurements as well as unit activity, is the experimental technique most capable of providing information about the cooperative actions of neurons at high temporal and spatial resolutions.

2.2 The Spike-Sorting Process

In order to obtain (multi)unit activity, the extracellular data is first bandpass-filtered to remove the LFP and high-frequency noise (as described in Section 2.1.1). To then obtain single-unit activity, we must perform spike sorting by sending this “raw” data through the signal-processing chain shown in Fig. 2.3. The first steps are *spike detection*, the process of separating spikes from background noise, and *alignment*, the process of aligning all detected spikes to a common temporal point relative to the spike waveform. Once the spikes have been identified, spike sorting can take place.

Most spike sorting methods—relying heavily on the previously mentioned assumption that each neuron produces a different, distinct shape (as seen by the electrode) that remains constant throughout a recording session—are based on spike waveform information. Thus, the first step in such methods is *feature extraction*, in which spikes are transformed into a certain set of features, such as principal components, that emphasizes the differences between spikes from different neurons as well as the differences between spikes and noise. After feature extraction, some form of *dimensionality reduction* typically takes place, in which feature coefficients that best separate spikes are identified and stored for subsequent processing while the rest are discarded. Finally, spikes are classified into different groups, corresponding to different neurons, based on the extracted feature coefficients; this process is referred to as *clustering*. The result, the signal of interest to the experimenter and to BMIs, is the train of spike times for each neuron. This information can be represented graphically by a raster plot, where ticks are drawn to indicate spike occurrences versus time, as shown in the right-most plot of Fig. 2.3 for three neurons.

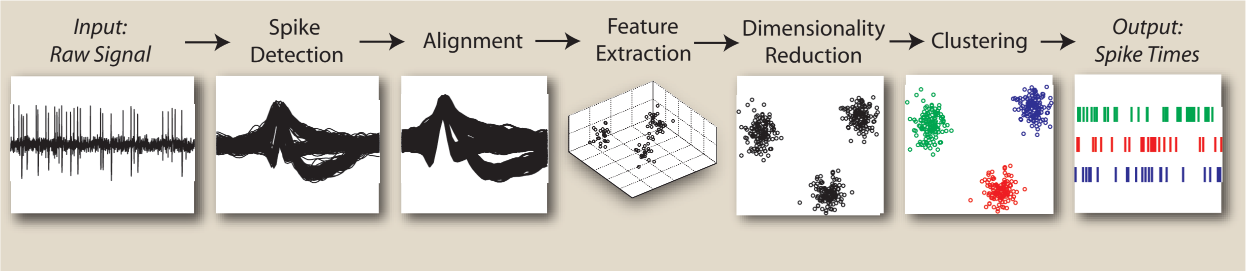


Figure 2.3: The signal-processing chain used to obtain single-unit activity.

2.3 Classifying Spike-Sorting Algorithms

Spike-sorting methods can be categorized according to a number of different characteristics. The first is the level of *autonomy*: methods can be "automatic/unsupervised" (fully autonomous) or "manual" (not at all autonomous). Automatic or unsupervised methods require no user input, while manual methods require constant supervision by an operator. Methods can also fall anywhere between these two extremes; a "semi-automatic" method is a method with both a manual stage and an automatic stage. Examples of semi-automatic methods include detection methods that require the manual setting of a threshold, or window-discriminator methods that require the manual definition of windows, but that then work automatically [41], as well as classification methods that require the user to manually re-assign clusters after automatic cluster determination [42]. For neural prosthetic applications, spikes must be sorted in real time, thus precluding manual spike sorting. And because of the growing amount of data resulting from an increase in the number of simultaneously recorded channels, manual spike sorting is no longer a viable option in research settings either. Therefore, automatic methods are now usually required.

A second way to classify spike-sorting algorithms is by whether or not they are *real-time* (also called *online*). The standard practice for many years has been to first record and store all the data and then to perform spike sorting offline after the experiment. As a result, many of the spike-sorting methods that have been developed are noncausal, in that they rely on access to all of the data at once. When using principal component analysis, for example, the principal components are often calculated using all of the detected spikes, and then each spike is projected onto these basis vectors before the actual classification takes place. It is increasingly common, however, for applications that require spike sorting to require that the spike sorting occur in real time. This requirement renders a number of hitherto commonly used methods inadequate. A compromise would be to modify offline algorithms to include an offline training period followed by a real-time classification period.

The third attribute by which spike-sorting methods can be categorized is *adaptivity*. As we will describe later, extracellular signals are not always stationary. In such cases it would be beneficial to use an algorithm that can adapt to a changing environment, as opposed to a static algorithm. There can be intermediate cases on this scale as well. For example, a static algorithm that requires a training period can be made adaptive by retraining it periodically.

Clustering algorithms can be further classified as either *parametric* or *nonparametric*. Koontz et al. define a nonparametric clustering algorithm as "an algorithm for clustering multivariate data which is not based on a parametric model of an underlying probability density function. In particular, a nonparametric algorithm should identify clusters of arbitrary shape and size" [43]. In other words, any algorithm that assumes a certain structure to the data or that is biased towards a particular cluster shape, such as spherical or ellipsoidal, will be considered parametric. The underlying probability density function for neural data is not known a priori (see Section 2.5.2), so nonparametric clustering algorithms are highly desirable.

Early spike-sorting algorithms were very simple, but not very accurate. In general, the more complex the method, the better the performance. This inherent tradeoff between algorithm accuracy and complexity leads to another characteristic by which to classify algorithms: the *accuracy–complexity* measure. As we described in the introduction, many applications require spike sorting in implantable hardware. Any implantable hardware is subject both to strict power-density constraints and to high reliability requirements. Thus, it is crucial to choose the spike-sorting methods with the optimal balance between accuracy and complexity to implement in hardware.

In the next section, we will give some examples of algorithms that have been used for each step of spike sorting. We will also mention some alternative methods, mostly statistical/probabalistic in nature, that do not conform to the block diagram shown in Fig. 2.3.

2.4 Overview of Spike-Sorting Algorithms

Note that Lewicki provided a nice review of spike-sorting methods in 1998 [44]. Here, we provide a relatively high-level description of the evolution of spike-sorting techniques as well as an update of more recent algorithms, and present them in the context of hardware spike sorting.

2.4.1 Detection

Nearly all detection methods involve two main steps: the pre-emphasis of the signal and the application of a threshold. Spike-detection methods vary in how the signal is pre-emphasized and in how the threshold is determined. All of these methods run automatically given the detection threshold, so whether or not the algorithm is fully automatic depends on whether or not the threshold can be determined automatically. All of these methods are also real-time (save a small latency for buffering spikes) given the detection threshold, but automatic calculation of the threshold usually involves a training period.

The early days of spike sorting came in a time before personal computers. Processing was done purely in analog hardware. As a result, spike-sorting methods were relatively primitive. Spike detection was typically performed using a simple **voltage trigger** or **Schmitt trigger**, where the voltage threshold was set manually by the user. Any time the voltage signal crossed that threshold, a pulse would be generated to indicate the presence of a spike [44]. Or, if the user needed the spike waveforms for subsequent spike sorting, a threshold crossing would trigger the capture of the spike waveform. This method is appealing because of its simplicity, and, as a result, is still used today by many experimenters. Some researchers have modified this method to include an **absolute-value** operation before the compare (or, equivalently, a compare to ± threshold, as shown in Fig. 2.4); the absolute-value threshold was confirmed to be better than a simple threshold in [45].

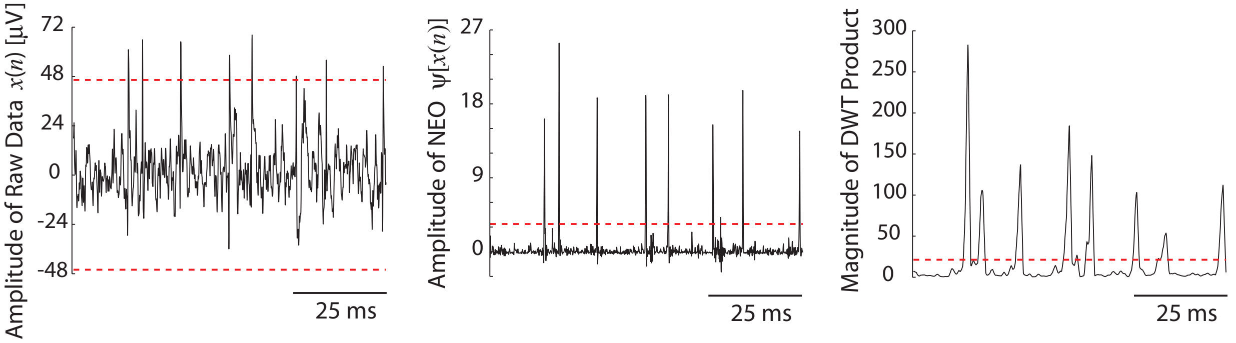


Figure 2.4: Examples of pre-emphasized signals and threshold values (dashed red lines) for three different detection methods. *Left*: Absolute-Value, *Middle*: NEO, *Right*: DWT Product [48].

For an autonomous spike-sorting system, the threshold must be calculated automatically. An intuitive value for this threshold would be a multiple of the standard deviation of the noise. This would minimize the probability of noise exceeding the threshold. One method for estimating the noise standard deviation would be to calculate the standard deviation of the entire signal (including spikes), under the assumption that spikes are sparse. However, [46] showed that as the firing rate of the data increases, this estimate becomes too high. So they suggested the following estimator:

$$\hat{\sigma}_{\mathrm{N}} = \mathrm{median}\left(\frac{|x(n)|}{0.6745}\right), \tag{2.1}$$

where $\hat{\sigma}_{\mathrm{N}}$ is the estimate of the noise SD and $x(n)$ is a sample of the original signal x at time n[1], and proposed the following detection threshold Thr:

$$Thr = 4\hat{\sigma}_{\mathrm{N}}. \tag{2.2}$$

Another class of spike-detection algorithms are based on detecting changes in the energy of the signal. One such algorithm is called the **nonlinear energy operator (NEO)** or the Teager energy operator (TEO). Originally described in [49], the NEO has been proposed for

[1] This formula may come from the fact that the median absolute deviation (MAD) of a standard normal distribution is 0.6745 [47]. It follows that for a zero-mean normal distribution with non-unit variance, $\sigma = \frac{\mathrm{MAD}(x)}{0.6745}$.

use in spike detection by [45, 50, 51]. In discrete time, the NEO ψ is defined as

$$\psi[x(n)] = x^2(n) - x(n+1) \cdot x(n-1), \tag{2.3}$$

where $x(n)$ is a sample of the waveform at time n. The NEO is large only when the signal is both high in power (i.e., $x^2(n)$ is large) and high in frequency (i.e., $x(n)$ is large while $x(n+1)$ and $x(n-1)$ are small). Since a spike by definition is characterized by localized high frequencies and an increase in instantaneous energy [50], this method has an obvious advantage over methods that look only at an increase in signal energy or amplitude without regarding the frequency. This can be seen in Fig. 2.4, which shows that the NEO operation increases the SNR of the signal, making detection less sensitive to the detection threshold. Another advantage of this method is that it is relatively simple to implement, whether in the digital or analog domain. The threshold Thr for this method can be automatically set to a scaled version of the mean of the NEO:

$$Thr = C\frac{1}{N}\sum_{n=1}^{N}\psi[x(n)], \tag{2.4}$$

where N is the number of samples in the signal [50]. The scale can be chosen initially by experiment (e.g., as described in Section 3.3.2) and then used as a constant.

Other spike-detection algorithms are based on **template matching**. If the spike waveforms of interest are known a priori to the user, then matched filters can be used to correlate the incoming signal with the spike templates; if the correlation crosses a certain threshold then a spike has been detected. With known cluster templates, this method can also be used for the actual spike classification. A related method is detection using the **discrete wavelet transform (DWT)**. The DWT, which is ideally suited for the detection of signals in noise (e.g., edge detection, speech detection), has recently also been applied to neural spike detection ([48, 52, 53]). This method has an intuitive appeal in that it is similar to template matching, where we correlate the signal with a known waveform, only it is scale-invariant. The DWT is also appealing because it can be implemented using a series of filter banks, keeping the complexity relatively low.

An example of one possible implementation is the DWT Product [48]. First, the stationary wavelet transform (SWT) is calculated at 5 consecutive dyadic scales ($W(2^j, n), j = 1, \ldots, 5$). Then the scale $2^{j_{\max}}$ with the largest sum of absolute values is found:

$$j_{\max} = \underset{j \in \{3,4,5\}}{\operatorname{argmax}} \left(\sum_{n=1}^{N} |W(2^j, n)| \right). \tag{2.5}$$

From here, we calculate the point-wise product $P(n)$ (or "SWTP") between the SWT at this scale and the SWTs at the two previous scales:

$$P(n) = \prod_{j=j_{\max}-2}^{j_{\max}} |W(2^j, n)|. \tag{2.6}$$

This product is then smoothed by convolving it with a Bartlett window $w(n)$ in order to eliminate spurious peaks, and a threshold is applied. The threshold Thr can be set automatically to a scaled version of the mean of this result:

$$Thr = C \frac{1}{N} \sum_{n=1}^{N} w(n) * P(n), \tag{2.7}$$

where N is the number of samples in the signal and C is a constant. Once again, Fig. 2.4 shows that the pre-emphasized DWT signal has an increased SNR compared to the original signal, making detection less sensitive to the detection threshold.

2.4.2 Alignment

When spike detection is performed in the digital domain, whenever the voltage signal crosses a threshold, a window is applied and a spike waveform is captured. At this point, each spike is essentially aligned to the point of the threshold crossing. However, sampling jitter combined with noise effects may leave features of interest, such as maximum and minimum values, misaligned. This temporal misalignment has the effect of increasing the spread of points in feature space, making clustering more difficult. Thus, alignment should be performed prior to classification.

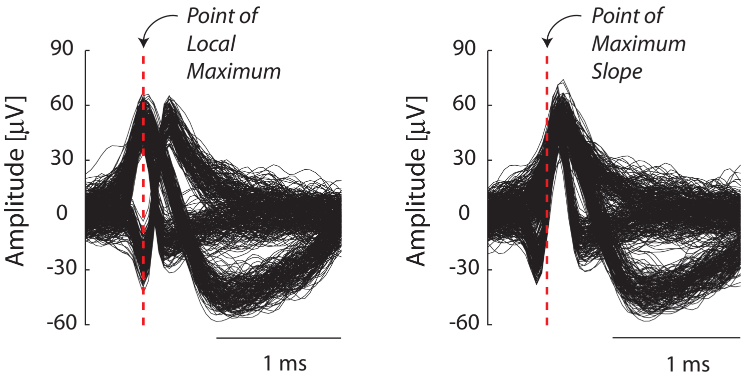


Figure 2.5: Examples of two different alignment methods. *Left*: alignment to maximum amplitude, *Right*: alignment to maximum slope.

The alignment process usually begins by upsampling the signal (using an interpolation method such as cubic spline) to help reduce the effects of sampling jitter. Then, the signal is aligned to some event in time. The aligned spikes may be downsampled to the original sampling rate after alignment.

The most common method of temporal alignment is to align each spike to the point of its **maximum amplitude** (Fig. 2.5) [44]. Alignment to the point of **maximum slope** (Fig. 2.5) has also been proposed [54], which is intuitive since the rising slope of the action potential has biological significance (Fig. 2.1). This method would be especially convenient if discrete derivatives (described in Sec. 2.4.3) were already being used for feature extraction. Others have proposed alignment to the maximum of an energy measure such as the **NEO** [55], which would be convenient if NEO were already being used for spike detection. Similarly, alignment to the **maximum integral** [56] would be convenient if the integral transform (described in Sec. 2.4.3) were being used for feature extraction. Indeed, it would be convenient to perform alignment with respect to any measure that is already being calculated in the sorting process.

Although the aforementioned alignment methods will usually improve classification accuracy, alignment to a metric that is derived from the whole spike rather than from a single point may be less susceptible to the effects of background noise. One example of such a metric is the spike's **center of mass** [57]. Note that all of the algorithms that have been described in this section are completely automatic and real-time.

2.4.3 Feature Extraction

Feature-extraction methods were also primitive in the early days of spike sorting. Often only very simple features such as the **maximum spike amplitude**, **peak-to-peak amplitude**, and **spike width** were used [44]. This approach, although simple, is quite susceptible to noise, as well as to intrinsic variations in spike shapes.

In the 1970s, as digital computers gained popularity and processing capacity, researchers began using more sophisticated algorithms for feature extraction, such as **principal component analysis (PCA)** [58]. In PCA, the orthogonal basis (i.e., the "principal components" or PCs) that captures the directions in the data with the largest variation is calculated by performing eigenvalue decomposition of the covariance matrix of the data. Each spike is then expressed as a series of PC coefficients c_i:

$$c_i = \sum_{n=1}^{N} PC_i(n) \cdot s(n), \tag{2.8}$$

where s is a spike, N is the number of samples in a spike/PC, and PC_i is the i^{th} PC (Fig. 2.6). These coefficients are then clustered to obtain the spike classifications. This method raised the bar on the classification performance that could be achieved, especially for noisier data. An added bonus is that, because most of the variance is captured in the first few components, the dimensionality can be reduced by keeping only the first two or three PCs, thereby reducing the computation time of PC coefficient calculation and of subsequent clustering. Even today, PCA is the most trusted and most commonly used method of spike sorting. The downside to PCA is that it is not a real-time algorithm. It is usually performed offline after the acquisition of the entire dataset, but it can be modified to include a training period during which the PCs are calculated followed by a real-time PC-coefficient-calculation period. Additionally, PCA is most effective on Gaussian data, while spike data may be non-Gaussian (Section 2.5.2).

Besides for spike detection, the **DWT** has also been proposed for feature extraction by

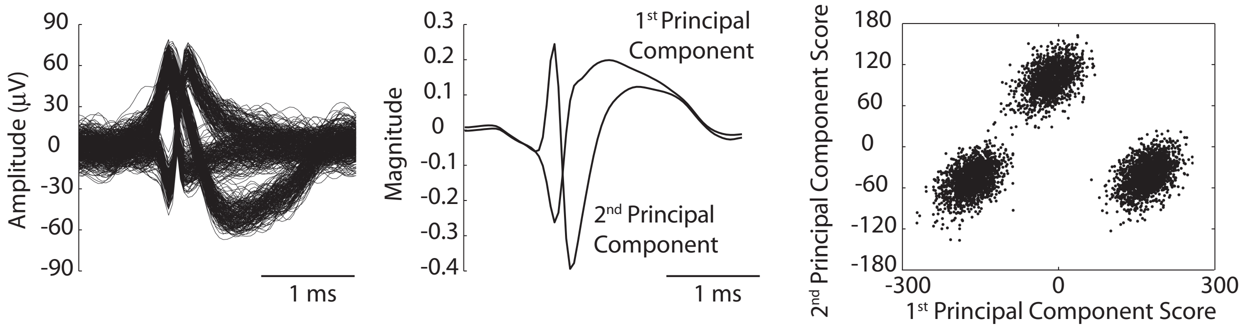


Figure 2.6: Sample results of feature extraction using PCA. For the detected spikes (*left*), principal components are calculated (*middle*), and each spike is expressed by its first two PC coefficients (*right*).

[46]. The DWT is given by

$$W(u,2^j) = \sum_{n=-\infty}^{\infty} s(n) \cdot \frac{1}{2^{j/2}} \cdot \Psi\left(\frac{n-u}{2^j}\right),$$

where u is a translation parameter (analagous to time), j is an integer, 2^j is a scale parameter (analagous to frequency), and Ψ is the wavelet function. The DWT should work well for feature extraction since it is a multi-resolution technique that provides good localization in both time and frequency. As in PCA, performing the DWT on spike waveforms results in a set of "expansion coefficients," which can then be clustered to achieve spike classification. An example showing the DWT of raw data is shown in Fig. 2.7.

Methods have also been developed with the accuracy–complexity tradeoff in mind. One such method is called **discrete derivatives (DD)** and is like a simplified version of DWT [59]. Discrete derivatives are calculated by computing the slope at each sample point, over a number of different time scales:

$$dd_\delta(n) = s(n) - s(n-\delta), \tag{2.9}$$

where s is a spike and δ is an integer related to the time scale.

Another such method is called the **integral transform (IT)** [60], in which spikes are classified based on the areas under the positive and negative phases of the spike, I_A and I_B,

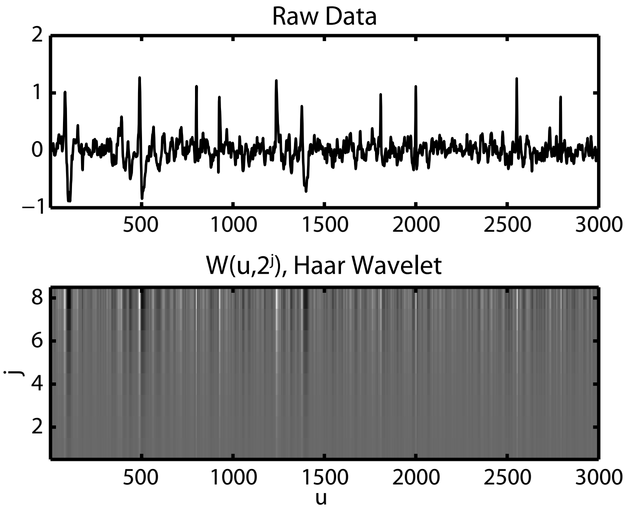


Figure 2.7: Example showing the DWT (lower plot) of raw data (upper plot), where u is a translation parameter (analagous to time), j is an integer, and 2^j is a scale parameter (analagous to frequency). Vertical axis of upper plot represents the signal amplitude (arbitrary units).

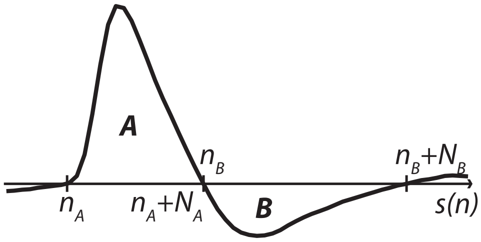


Figure 2.8: Illustration of feature extraction using the integral transform (IT).

respectively:

$$I_A = \frac{1}{N_A} \sum_{n=n_A}^{n_A+N_A-1} s(n)\,, \quad I_B = \frac{1}{N_B} \sum_{n=n_B}^{n_B+N_B-1} s(n), \tag{2.10}$$

where s is the spike, n_A is the first sample of the positive phase, N_A is the total number of samples in the positive phase, n_B is the first sample of the negative phase, and N_B is the total number of samples in the negative phase of the spike (Fig. 2.8). Parameters N_A, N_B, n_A, and n_B are all determined by offline training. This method is appealing because of the simple hardware implementation presented. Since only two features are extracted from each spike (I_A and I_B), the resulting dimensionality of this method is 2, and no dimensionality reduction is required before clustering.

2.4.4 Dimensionality Reduction

Dimensionality reduction is a critical step in spike sorting for a number of reasons. The most obvious reason is that it will significantly reduce the required memory and computational complexity of clustering, resulting in significant reductions in the area and power of the spike-sorting hardware. Another obvious benefit is that it reduces the output data rate of spike-sorting hardware configured to output features only. A third reason that makes dimensionality reduction critical is that it improves the accuracy of clustering. Adding dimensions in clustering improves the performance only up to a certain point, after which adding more dimensions can cause the performance of the clusterer to degrade. One reason for this may be that dimensions in which the data is not separated introduce noise or confusion into the clusterer.

The most primitive way that the dimensionality of features can be reduced is with **uniform sampling**, in which to reduce the dimensionality from N to D we simply choose D evenly spaced samples, for example, by choosing every N/D^{th} sample beginning with sample number $D/2$. This is essentially the same as choosing D random samples.

A smarter way to choose the features that can best separate clusters, as shown in Fig. 2.9, is by finding those features that have mulitmodal distributions across spikes, as multimodal distributions are an indication that more than one population (collection of spikes from the same neuron) is present in the dataset. Next, we present three dimensionality-reduction algorithms that use this approach.

The first of these algorithms is called the **Lilliefors Test** [61], a modification of the Kolmogorov-Smirnov Test. The null hypothesis is that the data under question comes from any normally distributed population (whereas the Kolmogorov-Smirnov Test tests the null hypothesis that the data comes from a standard normal distribution). The basic steps of the test are:

1. Calculate the population mean and population variance of the data.

2. Calculate the empirical distribution function (EDF) of the data.

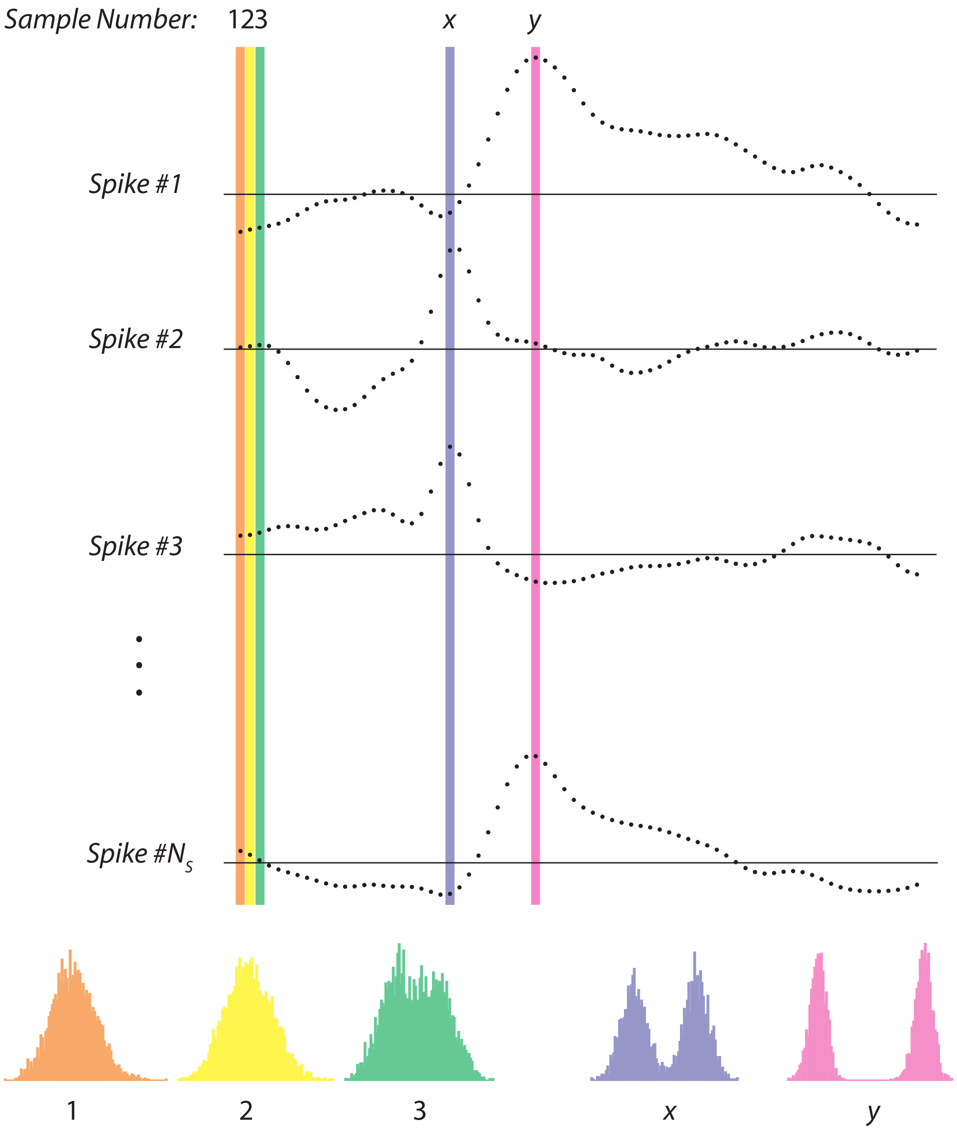


Figure 2.9: An illustration of how feature distribution information can be used in dimensionality reduction. For visualization purposes, we use the time samples of the spikes as features. In this example, the distributions of the amplitudes for samples 1, 2, and 3 are unimodal, so they would not be good choices of features to be used in clustering. Samples x and y, on the other hand, have bimodal distributions, indicating that clustering of these features would reveal the two underlying populations within the data.

3. Test statistic: The maximum discrepancy between the EDF and the cumulative distribution function (CDF) of a normal distribution having the mean and variance calculated in 1).

The Lilliefors Test has been used by [46] for dimensionality reduction in spike sorting, where the test statistic was used to find the coefficients whose distributions differed most from the normal distribution. The assumption is that the null hypothesis will be rejected for coefficients with multimodal distributions but not for coefficients with unimodal distributions.

Hartigan's Dip Test [62], [63] is a statistical test that looks specifically for multimodality. The CDF of a unimodal distribution has only one mode and is convex before the mode and concave after the mode. On the other hand, CDFs of multimodal distributions have more than one mode, and therefore have regions alternating between concave and convex. The basic steps in the Dip Test (illustrated in Fig. 2.10) are:

1. Calculate the EDF of the data.
2. Calculate the greatest convex minorant (GCM) and the least concave majorant (LCM).
3. Test statistic: The maximum distance ("dip") between the EDF and the unimodal distribution function that minimizes the maximum difference.

The coefficients whose distributions are "more multimodal" will have larger test statistics. Thus, we choose the coefficients that have the largest test statistics for use in clustering.

We proposed an alternative to the above algorithms with comparable accuracy yet far less complexity in [64]. In the **Maximum-Difference Test** (illustrated in Fig. 2.11), as in the Lilliefors Test, we seek the coefficients with the most variability, only now under the limited-memory conditions typical of implantable hardware. For an initial feature dimensionality N, four N-sample arrays of memory are initialized to zero: `maximum_difference`, `local_difference`, `spike_new`, `spike_old`. Throughout the training period, the i^{th} iteration of the algorithm is as follows:

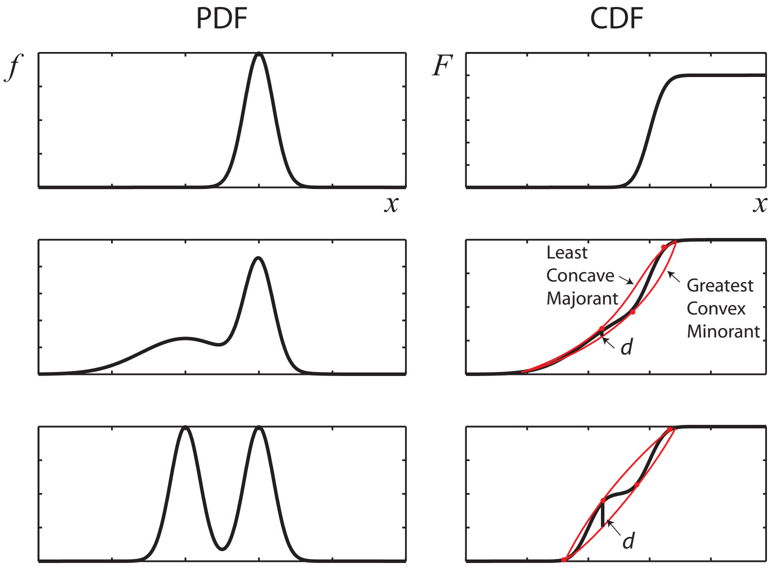


Figure 2.10: Example showing the calculation of the "dip" statistic (d) for Hartigan's Dip Test for various PDFs ($f(x)$ vs. x) and CDFs ($F(x)$ vs. x).

1. Write the current feature samples to the array `spike_new`.

2. Subtract the values in `spike_new`, coefficient by coefficient, from the values in `spike_old`, and write the absolute value of the result to `local_difference`.

3. Find the indices corresponding to the 3 largest values in `local_difference`.

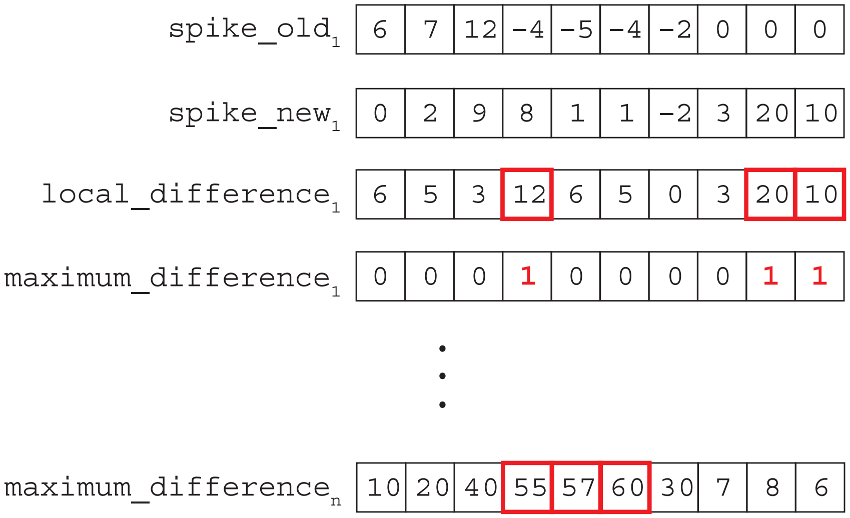
4. Increment the values in `maximum_difference` indexed by these 3 indices.

5. Overwrite the values in `spike_old` with the values in `spike_new`.

These steps are repeated until the end of the training period. At this point, assuming that the goal is to reduce the dimensionality from N to D, `maximum_difference` is scanned for the locations corresponding to the D largest values, and the coefficients corresponding to these indices are identified as the coefficients that will be used in clustering.

2.4.5 Clustering

Clustering, especially unsupervised clustering, is often the most difficult and most complex part of the sorting process.



$\texttt{spike\_old}_1$	6	7	12	-4	-5	-4	-2	0	0	0
$\texttt{spike\_new}_1$	0	2	9	8	1	1	-2	3	20	10
$\texttt{local\_difference}_1$	6	5	3	12	6	5	0	3	20	10
$\texttt{maximum\_difference}_1$	0	0	0	1	0	0	0	0	1	1
⋮										
$\texttt{maximum\_difference}_n$	10	20	40	55	57	60	30	7	8	6

Figure 2.11: Example execution of the Maximum-Difference Test. At the beginning of the algorithm, the first spike would be stored in `spike_old` and the second spike in `spike_new`. The difference between the two arrays is calculated and its absolute value stored in `local_difference`. The indices corresponding to the three largest values in `local_difference` are 4, 9, and 10; the values in `maximum_difference` indexed by these indices are each incremented by 1. These steps are repeated until the end of the training period, when we have the final value of `maximum_difference`. Assuming that we want to reduce the dimensionality from 10 to 3, we choose the features indexed by the indices corresponding to the 3 largest values of `maximum_difference`, 4, 5, and 6, to be used in clustering.

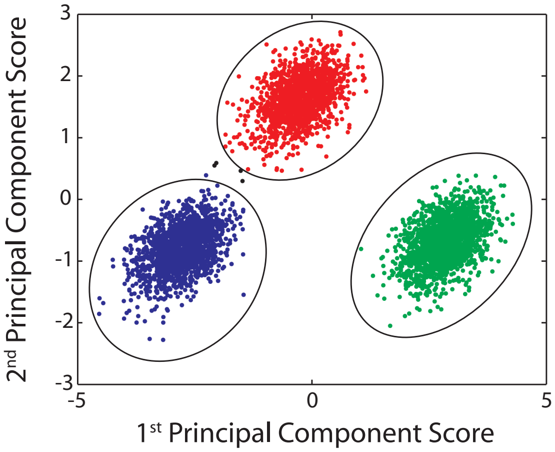


Figure 2.12: Example of results of manual cluster cutting on PCA features.

In the early days of spike sorting, the most common method of clustering was **manual cluster cutting**; extracted features were plotted on a scatter plot and cluster boundaries were defined by hand [44]. Even today, most commercial software packages for spike sorting provide the user with the capability to define cluster boundaries by drawing polygons in the chosen feature space using a mouse pointer (example in Fig. 2.12). But because this method is prone to human errors [41, 42], not to mention the time that is required of the operator, automatic and semi-automatic methods are desirable. Another primitive but at least semi-automatic technique is that of **window discriminators**, in which spike waveforms that intersect one or several user-defined windows are assigned to the same neuron.

A more sophisticated method of clustering, which is now the benchmark clustering method in this field, is called ***k*-means** [65]. The k-means algorithm is based on a distance metric. The main steps of the algorithm are described in Box 1. The main benefit of using k-means is that it is a very simple and fast algorithm. However, a major drawback to this algorithm is that it is not unsupervised, as it requires the user to input k. For applications such as BMIs, the spike sorting must be completely automatic, so there will be no user to input this information. Moreover, even if there were a user, determining the number of neurons is a nontrivial, often difficult task. Efforts are being made, however, to find ways of automatically and reliably determining the number of neurons in a recording [66]; perhaps these techniques could be used to initialize k-means, making it a fully unsupervised

1. Define k (number of clusters/neurons).
2. Randomly define the k cluster centroids.
3. Assign each data point to the cluster with the closest (usually by Euclidean distance measure) centroid.
4. Recompute each cluster centroid as mean of that cluster.

Steps 3–4 are repeated until a convergence criterion (either that the assignments stop changing or that the maximum number of iterations has been reached) is met.

Box 1: k-Means Clustering

algorithm. Another drawback of k-means is that it is not real-time, making it unsuitable for real-time applications. A compromise would be to adapt the algorithm to have a training period, where the cluster centroids are defined, followed by a real-time classification period, but this would only be appropriate for stationary data. Yet another drawback of this algorithm is that it is parametric; since each point is assigned to a cluster based solely on its Euclidean distances from the cluster centroids, the determined clusters will necessarily be spherical. There are many instances when the distribution of neural data will not be spherical. For example, during electrode drift, data tends to form ellipsoidal clusters. K-means would force spherical clusters, possibly dividing ellipsoidal clusters into two.

One unsupervised clustering algorithm is called **valley seeking** [67]. The idea in valley seeking is to first calculate the normalized density derivative (NDD) and then to find the peaks of this function. The cluster boundaries are then identified as the regions between the peaks (i.e., the valleys). An overview of the algorithm is provided in Box 2. The benefits of the valley-seeking algorithm are that it is unsupervised (not even the number of clusters is required to be provided by the user) and nonparametric, giving it the ability to cluster datasets that have nontrivial shapes, such as donuts and spirals. The algorithm is not real-time, however, making it unsuitable for real-time applications. Additionally, from a hardware point of view, the algorithm has the serious drawback of high complexity. It requires the computation and storage in memory of at least 6 N_S-by-N_S matrices, where N_S is the number of spikes being clustered. For large values of N_S, valley seeking may not be a

Definitions. Let x and x' be two data points. Denote the neighbor number (NN) of x' with respect to x as $NN(x, x') = k$ and $NN(x', x) = l$. Denote the i^{th} neighbor of x as $x_{(i)}$, i.e., $NN(x, x_{(i)}) = i$, and $NN(x_{(i)}, x) = a_{(i)}, i = 1, 2, \ldots, k-1$. Similarly, denote the j^{th} neighbor of x' as $x'_{(j)}$, $NN(x', x'_{(j)}) = j$, $NN(x'_{(j)}, x') = b_{(j)}, j = 1, 2, \ldots, l-1$. Algorithm steps:

1. Input threshold parameters t_1, t_2, and t_3.

2. Calculate the Euclidean distance matrix $D = (d_{ij})$.

3. Determine the neighbor number (NN) matrix $L = (l_{ij})$, where $l_{ij} = NN(x_i, x_j)$.

4. Calculate the matrix $S = (s_{ij})$, where
$$s_{ij} = \frac{(l_{ij} + l_{ji})}{2}.$$

5. Estimate the NDD matrix $J = (J_{ij})$, where
$$J_{ij} = \frac{|l_{ij} - l_{ji}|}{s_{ij}^{1+1/d}}, s_{ij} \leq t_1.$$
and d is the dimensionality of the feature space.

6. Estimate the convexity $D2 = (d2_{ij})$, where
$$d2_{ij} = \frac{l_{ij}\sum_{i=1}^{k-1} a_{(i)} + l_{ji}\sum_{j=1}^{l-1} b_{(j)}}{l_{ij}\sum_{i=1}^{k-1} i + l_{ji}\sum_{j=1}^{l-1} j}, s_{ij} \leq t_1.$$

7. Determine the discretized connectivity matrix $C = (c_{ij})$, where
$$c_{ij} = I(s_{ij} \leq t_1, J_{ij} \leq t_2, d2_{ij} \leq t_3)$$
is the indicator of whether x_i and x_j belong to the same cluster

8. Assign cluster labels to observations according to the discretized connectivity matrix.

Box 2: Valley-Seeking Clustering

viable choice for hardware implementation.

Superparamagnetic clustering (SPC) [68] is another unsupervised clustering algorithm that has found application to spike sorting [46]. In SPC, the data is modeled as a granular magnet, where each point is assigned a spin. The model is heated from low temperatures to high temperatures. At very low temperatures, all the spins will be aligned; this is referred to as the "ferromagnetic region." At high temperatures, the system is disordered and all the spins are random; this is called the "paramagnetic region." At temperatures that lie between these regions (called the "superparamagnetic region"), spins within the same high-density region are aligned while the spins of different high-density regions are not aligned; here the clusters are revealed. A summary of the algorithm steps is provided in Box 3.

SPC has benefits similar to those of valley seeking: it is unsupervised (again, no a priori knowledge of the number of clusters is required) and nonparametric. It also has the same major drawback: complexity. It requires the computation of at least 9 N_{S}-by-N_{S} matrices, where N_{S} is the number of spikes being clustered. Again, a large N_{S} requires a prohibitive number of operations and amount of memory. SPC also requires a Monte Carlo simulation, which increases the computation time. The algorithm can be simplified by using a mean-field approximation in place of the Monte Carlo simulations. But although this simplification reduces the runtime, it actually increases the complexity. Furthermore, like the valley-seeking method, SPC is an offline algorithm. Thus, SPC is also not a practical choice for hardware implementation.

The only clustering algorithm known to the author at this time that is both automatic and online and that has a good accuracy–complexity tradeoff is called **Osort** [69]. This method was developed by researchers who needed to isolate single neurons during their experiments, which requires processing large amounts of data in real time. Out of necessity, they proposed a much simpler way of clustering, where each data point is assigned to a cluster "on-the-fly". The algorithm is described in Box 4. This method of clustering appears to be extremely efficient. Very little memory is required. Therefore, of the three unsupervised clustering algorithms presented here, this method is the only one suitable for

1. Calculate the Euclidean distance matrix $D = (d_{ij})$.

2. Identify the K nearest neighbors of each point v_i. v_i and v_j are considered neighbors iff v_i is one of the K nearest neighbors of v_j and v_j is one of the K nearest neighbors of v_i.

3. Assign a Potts spin $s_i = 1, 2, \ldots, q$ to each point v_i randomly (or set all $s_i = 1$), where q is a constant representing the number of possible spins. Note: The value chosen for q does not imply anything about the number of clusters. The authors of [68] used $q = 20$.

4. Calculate the interaction strength J_{ij} between neighboring points v_i and v_j, where

$$J_{ij} = \begin{cases} \frac{1}{K}\exp(-\frac{d_{ij}^2}{2a^2}) & \text{if } v_i \text{ and } v_j \text{ are neighbors,} \\ 0 & \text{otherwise,} \end{cases}$$

   and a is the average of all d_{ij}'s between neighboring points.

5. For each temperature (e.g. $T = 0 : 0.02 : 0.2$), perform the following Monte Carlo simulation of iterations $m = 1 : M$:

   (a) Assign a frozen bond between nearest-neighbor points v_i and v_j with probability:

$$p_{i,j}^f = 1 - \exp(-\frac{J_{ij}}{T} \cdot \delta_{s_i,s_j}), \text{ where } \delta_{s_i,s_j} = \begin{cases} 1 & \text{if } s_i = s_j, \\ 0 & \text{otherwise.} \end{cases}$$

   (b) Generate a random number x from a uniform distribution on $[0, 1]$. If $x < p_{i,j}^f$, there is a bond between v_i and v_j.

   (c) Define clusters as all points that are connected by a bond.

   (d) Define c^m, where

$$c_{ij}^m = \begin{cases} 1 & \text{if } v_i \text{ and } v_j \text{ are in the same cluster,} \\ 0 & \text{otherwise.} \end{cases}$$

6. Calculate the two-point connectedness C_{ij}, where

$$C_{ij} = \frac{1}{M} \sum_{m=1}^{M} c_{ij}^m.$$

7. Calculate the spin-spin correlation function:

$$G_{ij} = \frac{(q-1)c_{ij} + 1}{q}.$$

8. If $G_{ij} > \theta$, where θ is a pre-defined threshold, v_i and v_j belong to the same cluster.

9. Assign cluster labels to observations according to G.

Box 3: Superparamagnetic Clustering

1. Initialization: Assign the first data point to its own cluster.
2. Calculate the Euclidean distance between the next data point and each cluster centroid.
3. If the smallest distance is less than the merging threshold T_M, assign the point to the nearest cluster and recompute that cluster's mean using the N most recent points. Otherwise, start a new cluster.
4. Check the distances between each cluster and every other cluster. If any distance is below the sorting threshold T_S, merge those two clusters and recompute its mean.

Steps 2–4 are then repeated indefinitely. In the simplified version of this algorithm, $T_M = T_S$, which is equal to the variance of the data computed continuously on a long ($\sim$ 1 minute) sliding window. Note that when computing cluster centroids, only the N most recent points are used. This helps to account for electrode drift, since the clusters are allowed to drift as well.

Box 4: Osort Clustering

implementation in hardware. The main drawback to this method is that, like in k-means, it bases its decisions on a distance metric, essentially assuming a spherical distribution of data. So while it can track spherical clusters moving in time to form ellipsoidal clusters, it cannot resolve a stationary ellipsoidal cluster (which would result, for example, from multivariate noise).

An example showing the results of clustering using valley-seeking clustering, SPC, and Osort is shown in Fig. 2.13. Valley seeking and SPC give similar results, whereas Osort appears to over-cluster—that is, to find too many clusters, or to divide a single-unit cluster into sub-clusters. A summary of various characteristics of each of the algorithms described in this section is given in Table 2.1. Because Osort is the only algorithm that is real-time and unsupervised while at the same time low in complexity, Osort would be the only practical choice for hardware implementation.

2.4.6 Alternative Methods for Spike Sorting

While most spike-sorting methods to date involve feature extraction followed by clustering of these features using nonparametric, nearest-neighbor methods as described above, other,

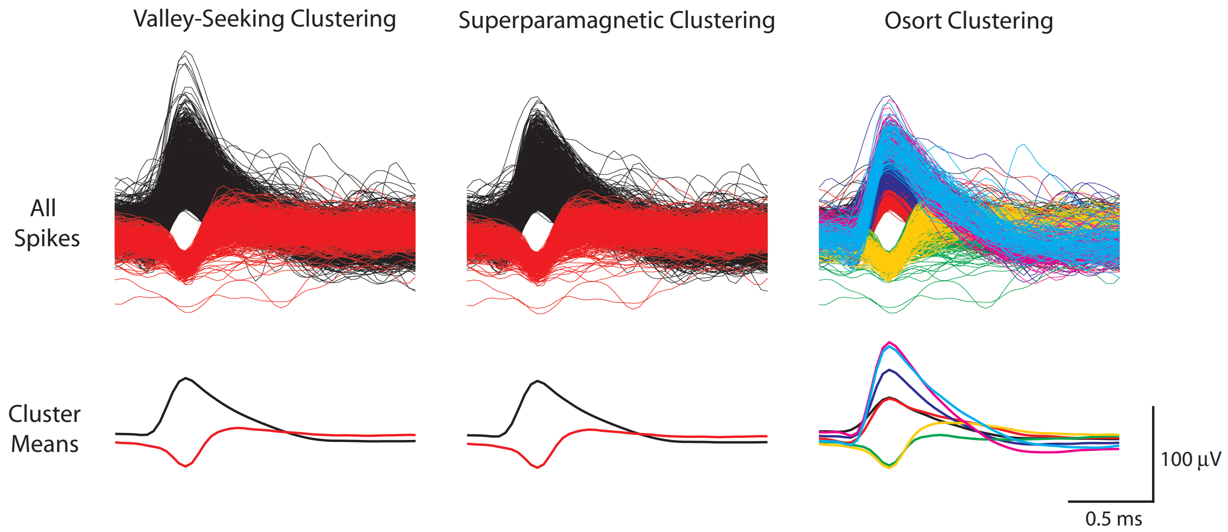


Figure 2.13: Example results from clustering 30 seconds of real data (human entorhinal cortex) using three different clustering methods. Note that for the valley-seeking and superparamagnetic methods, PCA was performed for feature extraction prior to clustering, whereas Osort uses only time-domain samples for clustering by default.

Table 2.1: Summary of Clustering Algorithm Characteristics

	Manual	**Window Discriminators**	***k*-Means**	**Valley Seeking**	**SPC**	**Osort**
nonparametric	NO	YES	NO	YES	YES	NO
unsupervised	NO	NO	NO	YES	YES	YES
real-time	NO	YES	NO	NO	NO	YES
adaptive	NO	NO	NO	NO	NO	YES
accuracy	low^{+}	?	0.90*	0.74*	0.85*	0.74*
complexity	–	–	LOW	HIGH	HIGH	LOW

$^{+}$[42], *[70]

more sophisticated methods have been developed based on Gaussian mixture models [57, 71–73] or t-distribution mixture models [74] in attempt to provide optimal solutions to the clustering problem based on statistics and probability theory. Many of these methods involve calculating a noise model for a particular dataset, performing noise whitening, and using Bayesian or maximum-likelihood estimation.

An example of one such method was presented in [73]. In summary, the method begins with calculating an empirical model for the recording noise and using this model to perform noise whitening on the data. Next, a "data generation model" (which includes the number of clusters and their positions in the event space) that maximizes the a posteriori probability to observe the samples that are actually observed (i.e., to maximize the likelihood function) is calculated as follows:

1. Specify a model M by specifying the number of neurons K, their discharge frequencies $\pi_j, j = 1, \ldots, K$, and their template waveforms $\boldsymbol{u}_j, j = 1, \ldots, K$.

2. Compute the probability for unit (neuron) j to have generated the event (spike) $\boldsymbol{e}_i$, $p(\boldsymbol{e}_i|\boldsymbol{u}_j)$, as follows:

   (a) Define the residual vector $\boldsymbol{\Delta}_{ij} = \boldsymbol{e}_i - \boldsymbol{u}_j$.

   (b) Then $p(\boldsymbol{e}_i|\boldsymbol{u}_j) = \frac{1}{(2\pi)^{D/2}} \cdot \exp\left(-\frac{1}{2} \cdot \boldsymbol{\Delta}_{ij}^T \boldsymbol{\Delta}_{ij}\right)$, where D is the dimensionality of the event space.

3. Calculate the probability P_i for the model to have generated event $\boldsymbol{e}_i$: $P_i = \sum_{j=1}^{K} \pi_j \cdot p(\boldsymbol{e}_i|\boldsymbol{u}_j)$, where π_j is the probability for unit j to occur.

4. Maximize the a posteriori probability, $\mathcal{P}(S; M) = \prod_{i=1}^{N} P_i$, or the likelihood function, $\mathcal{L}(S; M) = \sum_{i=1}^{N} \ln P_i$.

5. Use an iterative algorithm such as Expectation-Maximization to maximize $\mathcal{L}$.

Once the model is established, each event $\boldsymbol{e}_i$ is attributed to one of the K units by finding the j that minimizes $|\boldsymbol{\Delta}_{ij}|^2$, which is equivalent to choosing the unit with the highest probability to have generated the event.

The assumptions built in to this method are that the spike waveforms generated by a given neuron are constant, that the signal and the noise are statistically independent, that the signal and the noise sum linearly, and that the noise is well described by its covariance matrix. Non-stationary noise would violate this last assumption, but the authors in [73] claimed that in such cases several noise covariance matrices could be used successively to describe the noise. The same assumption would be violated if the noise covariance matrix were to have third or higher order moments. However, the authors showed that, at least for their data, the background noise is well described by its covariance matrix.

2.5 Challenges in Spike Sorting

2.5.1 No Ground Truth

There are many unique characteristics of neural recording that make classification of neural data more difficult than for other types of data. One such characteristic is that there is almost always a lack of any sort of "ground truth." Many popular classification techniques, such as support vector machines, rely on a training period that uses known data in order to define cluster boundaries before the automatic classification period begins. In extracellular recording, however, experimenters typically must play a more passive role; we can only observe the neural activity, we cannot influence it[2]. Thus, we have no ground truth to be utilized in training the algorithms.

The lack of a ground truth also makes it nearly impossible to quantify the performance of the classifier. Let's revisit the problem, illustrated in Fig. 1.1. The recording electrode is inserted into the neural tissue. Although each neuron in the tissue is generating its own, often independent, train of spikes, the recording electrode receives only the sum of the activity from all neurons in its vicinity. We want to use spike sorting to separate the composite signal into the individual spike trains. Now consider an analogous problem in classical

[2] Neural activity can be influenced by electrical stimulation, but intracellular stimulation (i.e. voltage clamping) is difficult to perform, while extracellular stimulation is not precise enough to influence individual cells.

communications theory. Let's say we want to quantify the performance of an error-correcting code (spike-sorting algorithm). To do so, we would generate a known test vector (signals from individual neurons), encode the signal, corrupt it with noise (mix the signals from individual neurons together), decode the signal (perform spike sorting), and finally calculate the bit error rate (classification performance) as the percentage of correctly received bits (percentage of correctly classified spikes). The problem is that in extracellular recording, we have no control over, or even knowledge of, the input signals. If we have no access to known test vectors, how can we quantify the performance of the classifier?

The best known solution to this problem has been to perform simultaneous intracellular/extracellular recordings [75]. Although we still have no control over the input signals, the intracellular recordings at least provide us with some knowledge of them, so we can to some degree evaluate the clustering performance. The problem with this method is that intracellular recordings are very difficult to make, and there are very few such datasets already in existence, making thorough algorithm evaluations difficult. Furthermore, for each of the paired intracellular/extracellular recordings in [75], although the extracellular electrodes (tetrodes) may have recorded signals from multiple neurons, only one neuron's spikes were intracellularly confirmed. This limits the degree to which the accuracy of an algorithm can be assessed. Another solution to the problem has been to use a real dataset that has been annotated by an expert according to spike occurrences and classes. However, studies have shown that the performance of human operators is actually much lower than that of semi-automatic clustering tools [41,42]. Therefore, it does not make sense to take the performance of a human operator as ground truth, particularly when calculating the accuracy of automatic methods that are likely to outperform the human operators. A third option has been to create biologically accurate synthetic datasets, which would provide both a ground truth and the flexibility to manipulate the signal variance and feature complexity in a way that is not possible using real data [46,64,69,76–78].

An entirely different approach to evaluating the performance of spike sorting is to use post-processing techniques based on our knowledge of the statistical properties of firing neurons. For example, one can look at the distribution of interspike intervals (ISI) for each neuron after spike sorting [44, 69, 79, 80]. Under most circumstances, after firing an action potential a neuron cannot fire again until after a refractory period, typically 1–3 ms. An interspike interval histogram showing a significant number of samples less than 3 ms would indicate bad clustering (e.g., that spikes from two neurons were combined into one multiunit cluster, or that the cluster is a noise cluster).

Tankus et al. developed a method specifically to identify a cluster as a single cell or multiple units [80]. This task is normally performed by human visual inspection of the distributions of spike waveforms around the spike mean (the variation around the mean for single units should be small). As such, their approach was to mimic the performance of the human classifier. The method is composed primarily of two parts. First, the ISI distribution of each cluster is examined as described above, and a cluster is declared multiunit if more than 1% of ISIs are less than 3 ms. For each remaining cluster, the variance of the spike waveforms around the main rise in voltage of the mean waveform is quantified. Then clusters whose variances exceed a certain threshold are also declared multiunit.

Pouzat et al. also developed several additional clever post-processing techniques, including the S.D. (standard deviation) test, the χ^2 test, and the projection test [73]. The idea behind the S.D. test is that, assuming that the spike waveform generated by a given neuron is constant and that the signal and noise sum linearly, the sample-by-sample S.D. over all the spikes from one cluster should be equal to the standard deviation of the noise. So after spike classification, any cluster whose S.D. differs significantly from the noise S.D. can be either further scrutinized or discarded. The χ^2 test tests the hypothesis that each cluster of spikes forms a D-dimensional Gaussian distribution. (In Section 2.5.2, however, we will examine whether the assumption of this test is valid or not.) The test is performed by first calculating the squared D-dimensional distance of every spike in a given cluster from its cluster mean and then by checking whether or not this distribution follows a χ^2 distribution with D degrees of freedom. A distribution that deviates significantly from the expected distribution may indicate the clustering of two similar units into one cluster. Finally, in the projection test, we again assume that the distribution of spikes in D-dimensional space should be a multivariate Gaussian with a covariance matrix equal to the identity matrix (assuming that noise whitening has been performed, as in Sec. 2.4.6), and that the projection of all spikes onto all possible axes joining any pair of units should form Gaussian distributions centered on the cluster centroids with standard deviations equal to 1. The "distinguishability" of any two given units can then be defined by setting a limit on the acceptable overlap between these two distributions, and a user can declare that units with less than a certain degree of distinguishability not be used in further analysis. This test also reveals when two clusters have been combined into one, as the projections between these two clusters will form a single Gaussian distribution centered around the true cluster mean.

2.5.2 Non-Gaussian Noise

Much of classical signal-detection theory is based on the assumption of channels having additive white Gaussian noise, and, as a result, most of the classical signal-detection techniques have been built around this assumption. Noise in extracellular recordings, on the other hand, has been shown to be both non-white [58] and non-Gaussian [32,74], so many of these classical techniques cannot be applied. Even signal-detection techniques that do not assume Gaussian noise, just that the distribution of the noise is known, are difficult to apply to neural data due to a lack of accurate noise models, especially models that are valid across various experimental setups.

2.5.3 Non-Stationarities

To make matters worse, neural data can be non-stationary. Fee et al. showed both that background noise is non-stationary and that a neuron's spike waveform varies as a function of the time since its preceding action potential [32]. This change in a neuron's spike waveform over time is especially dramatic during *burst firing* [44,81], during which the peak amplitude of the spike will decrease, since it is firing before completely returning to its resting state. Other causes of non-stationarites are *electrode drift* [44, 81], when the stiff electrode drifts within the fluid tissue with respect to the neurons being recorded, and cortical pulsation due to heartbeat or respiration [81]. Despite all of these known contributors to non-stationarity, data stationarity is still an assumption built into most spike-sorting methods. Until spike-sorting methods are developed to combat this problem, classification results will suffer. Efforts in this direction include [81,82].

2.5.4 Overlapping Spikes

A final, very tricky problem in spike sorting is that of overlapping spikes. The refractory period forces a neuron to rest for at least 1 ms between successive action potentials. But remember that our recorded signal is the sum of signals from several nearby neurons that are assumed to be firing independently. Thus it is possible for two different neurons to fire at or around the same time, such that their spikes overlap with one another in the recorded signal. At best, conventional spike-sorting methods may be able to identify such a detection as an outlier and, therefore, to classify it as noise. At worst, this overlap would be misclassified entirely. Ideally, we would like to be able to detect when an overlap occurs, and to resolve which neurons the overlapping spikes have come from. Some techniques have been developed towards this goal: [54,71,83–87].

2.6 Single vs. Multichannel Signal Processing

In multichannel recordings, adjacent channels sometimes receive activity from the same neurons. Examples of these types of multichannel recordings are stereotrode/ tetrode recordings, where the recording probes (made from microwires) have two/four closely spaced electrodes ($\sim 10\ \mu$m between centers). In these cases, correlations between channels can be exploited in order to separate single-unit activity, similarly to how triangulation can be used to determine the position of one object with respect to two other objects. Several algorithms have been developed to make use of this information, including **independent component analysis (ICA)** [88].

The idea behind ICA is that if N sources (neurons) have been mixed onto N detectors (electrodes) using a linear combination, a matrix can be found to "un-mix" the data such that each channel is independent from every other channel—that is, each channel contains the spike train from each neuron. The assumptions behind this method are that, ideally, the number of electrodes equals the number of neurons (or, in the less ideal case, the number of detectors is greater than the number of sources), and that each neuron is seen by at least two electrodes. This method, when it can be applied, has many benefits, including the ability to automatically detect artifacts and overlapping spikes and to correctly sort spikes from a neuron whose amplitude changes with time (i.e., to handle waveform non-stationarities).

Note that multichannel silicon arrays have much larger spacing between electrodes ($\sim$ 400 μm for the Utah array [89]), so it is much less likely for the same neuron to be recorded on two channels here. As such, algorithms exploiting correlations between channels can typically not be used for multichannel recordings of this type.

2.7 Controversy

There is an ongoing debate within the field of neural prosthetics—still a relatively immature field—over whether or not spike sorting is really necessary for reliable decoding. Spike sorting in the traditional sense seeks the single best spike train for each observed neuron. As a result, ambiguous spike trains often get discarded, which may be undesirable or unacceptable in some cases such as chronic recordings. To mitigate this problem, Wood and Black propose using an infinite Gaussian mixture model to instead generate a *distribution* of spike trains—that is, multiple different spike-sorting results with varying probabilities [90]. Then, this distribution of spike trains, rather than a single spike train, would serve as input to subsequent processing steps. The authors postulate that these results may be useful in certain types of neural signal analysis such as decoding algorithms which rely on cosine tuning. This approach has the benefit of quantifying the certainty of spike-sorting results and of improving single-unit yield. However, it is unclear how straightforward it would be to use this approach in neural signal analysis; downstream algorithms would likely have to be modified.

A number of other researchers have actually reported sufficient decoding performance when multiunit, rather than single-unit, activity is decoded. Ventura, for example, presented a paradigm for using multiunit spike trains in conjunction with existing decoding algorithms, such as the population-vector and maximum-likelihood decoding algorithms, to predict movement with comparable performance to traditional methods that use single-unit spike trains [91]. By bypassing spike sorting, this method saves time and computational effort, making it more appropriate for use in real-time neural prosthetics. This method has the added advantage of performing well in low SNR, where spike sorting can be unreliable. Actually, though, spike sorting is implicitly built in to this method, in that each constituent neuron's identity is revealed through information about tuning.

Stark and Abeles, on the other hand, came up with a decoding paradigm that involves no spike sorting at all, whether explicit or implicit [92]. They introduce a quantity called MUA (multiunit activity), which is calculated by bandpass-filtering the signal from 300–6000 Hz and taking the RMS. They then used the MUA as input to classification algorithms such as support vector machines, Fisher's linear discriminant analysis, Poisson probability density estimation, and artificial neural networks, which traditionally use single-unit activity. For each of these decoding methods, they found MUA to give better motion-prediction performance than either single-unit activity or LFP. Other advantages of this method are that MUA is more easily obtained than single-unit activity, MUA recordings are more stable over time, and MUA is informative even in the absence of spikes. An example of an MUA signal, compared to spikes and LFP, is shown in Fig. 2.14.

Another study showed that, while decoding is still better when single units are used, an acceptable level of performance can also be achieved using mulitple units [93]. A number of other researchers have also reported success in movement decoding using LFP [94, 95]. The primary advantages to using LFPs over spikes are that they are easier to acquire, are more stable over time, and are less susceptible to noise. Many other studies have suggested using a combination of LFPs and spikes to achieve high decoding performance [96–98].

Still, the majority of published studies in the field of neural prosthetics have used single-unit activity as input to their decoding algorithms [14, 15, 99, 100]. Furthermore, neural prosthetics is just one of many applications for spike sorting. Spike sorting will always be necessary for electrophysiological experiments that are designed to study the behavior of

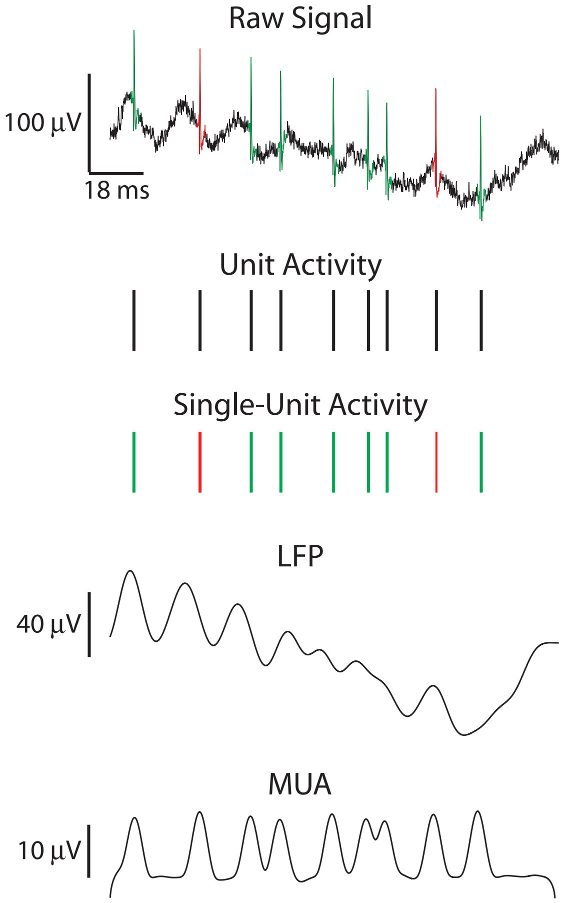


Figure 2.14: Example of an MUA signal, compared to the raw signal, unit activity, and LFP. Signals in this figure were generated by the authors and plotted in the same manner as in [92].

individual cells or networks of cells.

2.8 Conclusion

Spike sorting is an important processing step for many of the scientific and clinical applications that involve the extracellular recording of neuronal activity. Work still remains in finding optimal automatic, real-time, efficient, and accurate spike-sorting algorithms that address all the remaining challenges described in Sec. 2.5. Finding such a solution to the spike-sorting problem would finally allow reliable spike sorting to be performed in implantable hardware. Performing spike sorting in hardware, simultaneously on many channels, would provide researchers with whole new experimental paradigms. For example, on-site spike sorting would aid in providing experimenters with instantaneous information about the neurons, such as their tuning functions as a stimulus is varied. These signals could also be used to "close the loop" by delivering signals back to the brain, enabling a whole new class of neurophysiological and neuropsychological experiments. Performing spike sorting in hardware would also achieve enough data reduction to enable the wireless transmission of data, thereby eliminating the need for cables. This would open the door for new types of experiments in which the activity of the brain is investigated as animals move freely in enriched, and possibly even their natural, environments. It may also allow for recording from species that have never before been recorded, such as freely flying bats. And finally, implantable spike-sorting hardware would bring medical technologies for the treatment of disorders such as paralysis, epilepsy, and even cognitive and memory loss closer to a reality.

Acknowledgments

The authors thank Dr. R. Staba for reviewing the manuscript and providing invaluable feedback, especially with regards to the biology, and for providing access to human electrophysiological recordings. We also thank Mr. V. Karkare for help in developing the maximum-difference algorithm, as well as Mr. R. Chandler and Mr. M. Rachid for their comments. This work was supported in part by the National Science Foundation under Grant Numbers 0847088 and 0824275.

Permissions

Substantial portions of this chapter have been reprinted, with permission, from: S. Gibson, J. W. Judy, and D. Marković, "Spike Sorting: The First Step in Decoding the Brain," *IEEE Signal Process. Mag.*, vol. 29, no. 1, pp. 124–143, Jan. 2012. Copyright © 2012, IEEE.

Author contributions: J. W. Judy and D. Marković were the PIs.

CHAPTER 3

Algorithm Evaluations

3.1 Introduction

As described in Chapter 2, many spike-sorting methods have been published over the past 30 years. However, each time a new algorithm is published, it is demonstrated on a different dataset, which could be real or simulated, realistic or unrealistic, noisy or noiseless, etc. No standard dataset or methodology has been established for measuring a new algorithm's performance against existing algorithms. This makes it impossible for one to make fair comparisons between algorithms or to decide which algorithm is best for a given scenario (e.g., high or low SNR).

Here, we attempt to provide the needed dataset and methodology. Whereas many independent groups (e.g., [46, 48, 50, 51, 56, 60]) have evaluated individual algorithms using different, often biological, datasets, we have used the neural-signal simulator introduced in [46] to develop synthetic datasets in order to obtain an accurate, unbiased comparison between algorithms over a wide range of SNRs. Implantable spike-sorting hardware must be low-power in order to prevent heat-related tissue damage and to maximize battery life, as well as low-area in order to be implantable. The algorithms implemented in the hardware must be accurate, automatic, and real-time, as well as computationally simple in order to stay within the power limitations. Thus, we provide a methodology for evaluating algorithms in terms of accuracy versus computational complexity. We finally use this dataset and methodology to evaluate several spike-detection, feature-extraction, and dimensionality-reduction algorithms in order to determine which are most appropriate for use in a real-time, implantable neural-recording system.

This chapter is organized as follows. Section 3.2 describes the datasets that were created and Section 3.3 describes the methodology that was developed to evaluate the spike-sorting algorithms. Section 3.4 lists the algorithms that are evaluated in this chapter. Section 3.5 presents the results of the algorithm evaluation, including the accuracy, number of operations per second (NOPS), estimated area, and normalized cost of each algorithm. Section 3.6 provides a discussion of the reduction in data rate at each block of signal processing. Finally, conclusions and future work are discussed in Section 3.7.

3.2 Test Data

In order to evaluate the accuracy of spike-sorting algorithms, data with known spike times and classes is required. As described in Section 2.5.1, there are a few different ways to obtain such data. For this study, we chose to generate a semi-synthetic dataset using the data-driven neural-signal simulator introduced in [46]. The simulator contains a library of 594 average spike shapes taken from a collection of real physiological recordings. After the user selects the number of spikes, the spike shapes, the spike amplitudes, the firing rates, and the standard deviation of the noise, the system generates a waveform with the chosen spikes placed randomly in time (given a refractory period) at the chosen rate. Noise is automatically added by randomly choosing spike shapes from the library and adding them to the waveform at random times[1], which is a more biologically sound assumption than it would be to use only additive white Gaussian noise (see Section 2.5.2). Along with the test data ("raw data"), the simulator also generates a file containing the true spike time and the true spike class for each actual spike in the data file, which can be used to calculate accuracy (Fig. 3.1).

We wanted to produce datasets with varying degrees of difficulty to represent the diversity of data likely to be encountered in real recordings. Thus, we used this simulator to create

[1] The total firing rate of all spikes composing the background noise is 48 kHz. Henze et al. showed that extracellular electrodes can detect spikes within a 140-μm radius and that, in the hippocampus, about 1100 neurons are present within this volume [75]. A total firing rate of 48 kHz for these neurons is equivalent to each cell firing at about 43 Hz, a reasonable assumption. Note that the background spikes overlap with one another to a high degree, so individual spike shapes within the noise are generally not observable.

datasets with varying spike shapes, numbers of neurons, firing rates, amplitudes, and noise levels. We generated 96 datasets at 17 different noise levels, for a total of 1632 datasets. First, 24 sets of spikes were chosen from the library randomly (Fig. 3.2). Out of the 24 sets, 6 contained 2 neurons (Fig. 3.2a), 6 contained 3 (Fig. 3.2b), 6 contained 4 (Fig. 3.2c), and 6 contained 5 (Fig. 3.2d). From each of these 24 sets, 4 datasets were generated: 2 where the amplitudes of all spikes were equal (normalized to one) and 2 where the amplitudes were unequal. Each of these groups included one dataset where the firing rate of each neuron was equal (40 Hz) and one where the firing rates varied (from 5 to 40 Hz). Finally, we generated 17 versions of each of the 96 datasets at different noise levels, where the standard deviation of the noise ranged from $\sigma = 0$ to 0.4. This range allowed us to explore very high (∼20 dB) to very low (~ -10 dB) SNRs with the granularity needed in order to analyze at which SNRs the algorithms begin to break down. All datasets were 60 s in length and generated at a sampling rate of 24 kHz.

The SNR of each neuron in a given dataset was calculated as the peak-to-peak amplitude of the average spike shape from that neuron divided by 6σ, where σ is the standard deviation of the noise. Because the signal is sparse, we used the standard deviation of the raw data as an approximation for σ. The SNR of the entire dataset was then defined as the average SNR of all neurons in the dataset.

3.3 Testing Methodology

3.3.1 Accuracy Calculations

All detection, feature-extraction, and dimensionality-reduction algorithms were tested in MATLAB on the test data described above.

Spike detection using all of the methods was accomplished as follows: When a sample in the pre-emphasized signal crosses the threshold, a 3-ms window (1 ms before the threshold crossing and 2 ms after) is applied to the signal and the result is saved as a spike. This window length was chosen because a spike duration is unlikely to be longer than this and

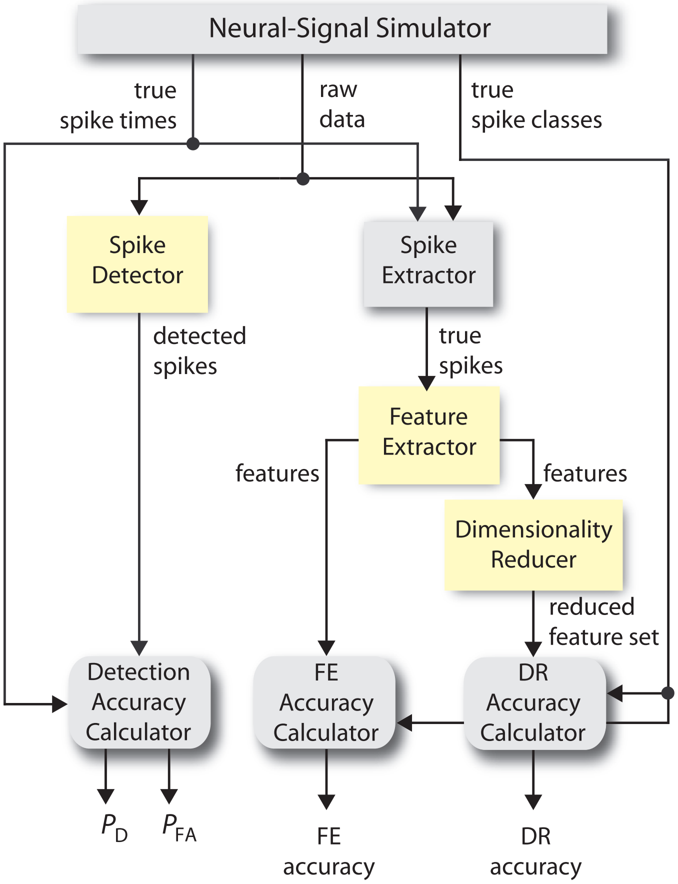


Figure 3.1: Signal-processing chain used to evaluate algorithm accuracies. The processing blocks that are being tested are highlighted in yellow.

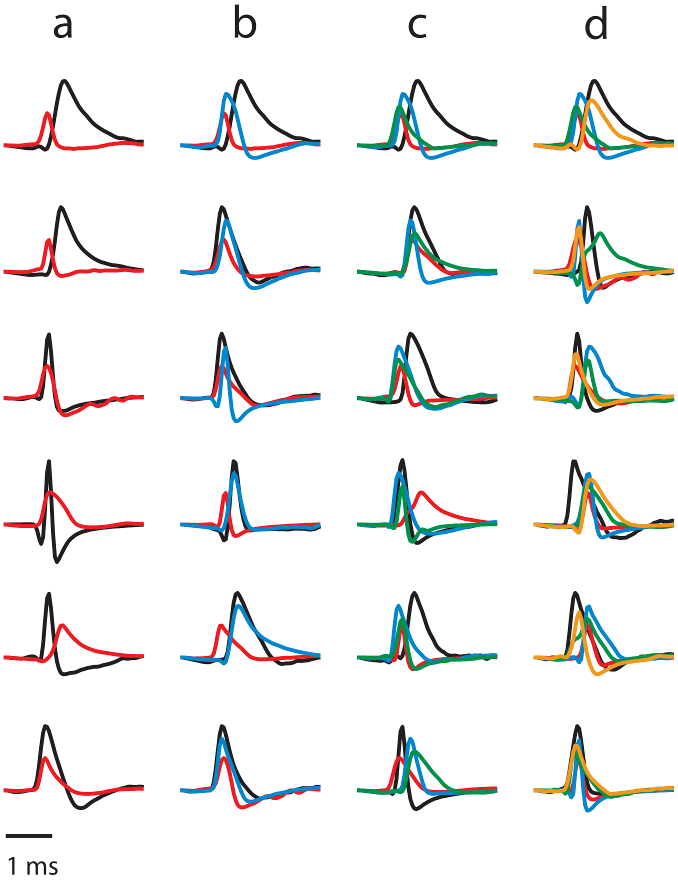


Figure 3.2: Average spike waveforms used as templates in the test datasets. Each set in column (a) contains 2 neurons, (b) 3 neurons, (c) 4 neurons, and (d) 5 neurons. For each set of spikes shown, 4 datasets were generated: 2 where the amplitudes of all spikes were normalized and 2 where the amplitudes were different (as shown). Within these pairs of datasets, the firing rate of each neuron was equal (40 Hz) in one and varied (from 5 Hz to 40 Hz) in the other.

because it minimizes the likelihood that we capture more than one spike in the window. In order to evaluate the quality of each detection method, the false-alarm and detection rates were calculated. In accordance with signal-detection theory, the false-alarm rate is defined as

$$P_{\mathrm{FA}} = \frac{\text{number of false alarms}}{\text{number of true negatives}}, \tag{3.1}$$

and the detection rate is defined as

$$P_{\mathrm{D}} = 1 - P_{\mathrm{M}}, \quad P_{\mathrm{M}} = \frac{\text{number of misses}}{\text{number of true positives}}. \tag{3.2}$$

We defined true positives as the samples within known spikes and true negatives as all other samples. False alarms are all the samples within a detected spike that are not part of a true spike, and misses are all samples of true spikes that are not part of detected spikes.

As depicted in Fig. 3.1, the accuracy of each feature-extraction method was calculated as follows. For each signal over the range of SNRs, true spikes were extracted using the file of true spike times. This assured that the accuracy calculations for the feature-extraction methods were not affected by the possible inaccuracies of spike-detection methods. Features from the detected spikes were then extracted using each of the feature-extraction algorithms being tested. These feature sets were then clustered using the MATLAB implementation of fuzzy c-means, the most accurate out of all clustering methods tried. The accuracy was then calculated by comparing the computed cluster assignment of each spike to the actual identities of each spike.

The accuracy of each dimensionality-reduction method was calculated similarly. For each test signal, the features extracted using each feature-extraction method were reduced using each of the dimensionality-reduction methods under test. These reduced features were then clustered using fuzzy c-means clustering, and the accuracy was again calculated by comparing the computed cluster assignment of each spike to the actual identities of each spike.

3.3.2 Receiver Operating Characteristic (ROC)

ROC curves were used to evaluate the performance of the various spike-detection algorithms. For a given method, the ROC curve was generated by first performing the appropriate pre-emphasis (as described in Section 3.4) and then systematically varying the threshold on the pre-emphasized signal from very low (the minimum value of the pre-emphasized signal) to very high (the maximum value of the pre-emphasized signal). At each threshold value, spikes were detected and P_{D} and P_{FA} were calculated in order to form the ROC curve. The area under the ROC curve (also called the "choice probability") represents the probability that an ideal observer will correctly classify an event in a two-alternative forced-choice task. Thus, a higher choice probability corresponds to a better detection method.

The ROC curves were also used to choose the spike-detection parameter C, which multiplies the automatically calculated thresholds (for example, in Eq. 2.4 and Eq. 2.7). For each method, the ROC curve of an initial training dataset was examined and the best threshold chosen given an acceptable error ($P_{\mathrm{D}} > 70\%$ and $P_{\mathrm{FA}} < 30\%$, determined empirically). C was then defined by dividing this threshold by the mean of those samples used to generate the ROC curves (i.e., the respective pre-emphasized signal).

3.3.3 Complexity Calculations

The number of operations per second (NOPS) is typically used to quantify an algorithm's computational complexity. Thus, one dimension to our complexity estimates was the NOPS, where an operation is defined as a 1-bit addition. For simplicity, a subtraction was considered to be equal to an addition in terms of number of operations, and multipliers and dividers were considered to require 10 times as many NOPS as an addition. In this chapter, this metric is reported in MOPS, millions of operations per second.

NOPS is a good indication of the power that an algorithm will require, but not necessarily of the area. Since we are concerned with both the power and area requirements of spike sorting, we also estimated the area requirement of each algorithm. The area can be divided into the area of logic and the area of memory. For logic, we estimated the required number of adders and multiplied by 20.46 μm^2 per 1-bit full adder (obtained from standard-cell estimates in a commercial 90-nm CMOS process). Subtractors and comparators were assumed to be the same size of an adder, while multipliers and dividers were assumed to be 10 times the size of an adder. For memory, we distinguished simple delays from memories that require addressing logic. For simple delays, we assumed 15 μm^2 per 1-bit register (obtained from standard cell estimates in a commercial 90-nm CMOS process). For memory, we estimated the memory requirement (in bits) for each algorithm and then estimated the size of this memory (in mm^2) in SRAM. In order to estimate the size of a memory using SRAM, we used the ARM-Artisan memory compiler to generate memories of different sizes and estimated their areas from the compiler reports. The memory depths supported by the tool vary from 1 kB to 8 kB (up to 8192 words, each word assumed to be 8 bits wide for our analysis). To estimate the area for memories larger than 8 kB, it is assumed that the memory is built from multiple smaller memories.

Finally, we combined these two metrics, NOPS and area, into a cost function so as to easily compare the complexities of different algorithms, as well as to provide a perspective on the complexity of individual processing steps (e.g., detection, feature extraction) relative to other processing steps. As shown in Eq. 3.3, first the NOPS for each algorithm was normalized by dividing by the maximum NOPS of any algorithm. Second, the same was done for area. Finally, these two numbers were summed. Thus, the cost function has a range from 0 to 2, 0 being least costly and 2 most costly.

$$\text{Normalized Cost}_i = \frac{\text{NOPS}_i}{\max\limits_i \text{NOPS}} + \frac{\text{area}_i}{\max\limits_i \text{area}} \tag{3.3}$$

To make these estimates, we assumed a sampling rate of 24 kSa/s, a maximum firing rate of 100 spikes/s, and 8-bit quantization for all signals. Also, since detection and dimensionality reduction both require a training period, we assumed 10 s of training. Finally, note that spike detection can be performed in either the analog or the digital domain. Because for 8 bits the power and area of a digital implementation are slightly lower than that of an analog implementation [101], we prefer to implement the circuit as a digital block. Thus, digital spike detection was assumed in the complexity estimations.

3.3.4 Assumptions

At this point we should point out a number of assumptions that were made in this analysis. The first is regarding the nature of the recording noise. Section 3.2 describes the way in which one source of noise, background spikes, was modeled and added to the test signals. Background activity is often the primary source of noise in an extracellular recording, but it is not the only source. Imperfections in the recording hardware, including electrodes, amplifiers, and ADCs, also contribute to the noise observed in extracellular recordings. Such hardware-dependent noise sources can vary significantly depending on the circuit implementations. Because it is difficult to generalize parameters such as thermal noise across these various circuit implementations, we chose to include only hardware-independent noise sources. Noise due to hardware should be small in comparison to background noise anyway, so we would not expect the inclusion of such noise sources to change the findings of this paper.

A second caveat is that all performance analyses were performed using floating-point arithmetic. We chose to go this route because we wanted a performance characterization of the algorithms themselves, in their purest forms, independent of their hardware implementations. The wordlengths for each signal can always be optimized during the design process to ensure whatever accuracy (relative to the floating-point performance) is needed.

The last assumption that was made is that the processing steps can be optimized somewhat independently. In this chapter, we will evaluate 3 detection, 4 feature-extraction, and 4 dimensionality-reduction algorithms. This gives us 48 possible combinations of algorithms. To simplify the analysis, we will treat detection independent from feature extraction and dimensionality reduction, since we expect minimal interaction between detection and feature extraction/dimensionality reduction. However, as will be discussed in Section 3.5.3, we suspect a higher degree of interaction between feature extraction and dimensionality reduction. Thus, we will first calculate the individual feature-extraction accuracies and then test the accuracies of combinations of the best feature-extraction algorithms with dimensionality-reduction algorithms.

3.4 Algorithms

The goal of this study was to evaluate a set of algorithms that are representative of the various principles used by, as well as of the range of complexity seen in, various spike-sorting methods. Thus, we investigated the following algorithms.

- For spike detection:
  - **Absolute value**, representing the class of detection algorithms based on voltage thresholding.
  - **Nonlinear energy operator (NEO)**, representing the class of detection algorithms based on energy thresholding; and
  - **Stationary-wavelet-transform product (SWTP)**, representing the class of detection algorithms based on wavelets.
- For feature extraction:
  - **Principal component analysis (PCA)**, the benchmark method in feature extraction;
  - **Discrete wavelet transform (DWT)**, representing the class of feature-extraction algorithms based on wavelets;
  - **Discrete derivatives (DD)**, representing the lower-complexity feature-extraction methods; and
  - **Integral transform (IT)**, also representing the lower-complexity methods.
- For dimensionality reduction:
  - **Lilliefors Test**, representing the class of statistical methods for dimensionality reduction;

- **Hartigan's Dip Test**, also representing the class of statistical methods for dimensionality reduction;
- **Maximum-Difference Test**, our own invention [64], representing a simple, intuitive method; and
- **Uniform sampling**, representing the most basic method.

Mathematical definitions of each of the above algorithms were provided in Chapter 2.

3.5 Results

3.5.1 Spike Detection

The ROC curves in Fig. 3.3 clearly show that the SWTP method is inferior to the other two methods. The curves corresponding to the absolute value and NEO methods, however, are too close to draw any conclusions. We next statistically compared the underlying choice-probability distributions, shown in Fig. 3.4, in order to determine which of these methods is better. The difference between each distribution is statistically significant (Kruskal-Wallis test, $p < 0.01$). The median of the NEO choice-probability distribution is highest, indicating that its performance is best. Figure 3.5 gives insight into the performance of each of these algorithms versus SNR and shows, more dramatically than Fig. 3.3 or Fig. 3.4, that NEO performs better than absolute value across SNRs. This result indicates either that the threshold calculation technique in Eq. 2.4 is more robust than the technique in Eq. 2.2, or that the NEO method is generally less sensitive to the choice of threshold than the absolute value method.

The complexity of each spike-detection algorithm, including the complexity for threshold calculation, is listed in Table 3.1. The absolute-value method is the least complex, followed by the NEO method. SWTP's cost of 2 indicates that it is most complex in terms of both OPS and area. Figure 3.6 shows the choice probability (median over all datasets and SNRs) versus cost. Methods that lie in the upper left corner (high accuracy, low complexity) are optimal for our application. Because the NEO method lies in the upper left corner, we

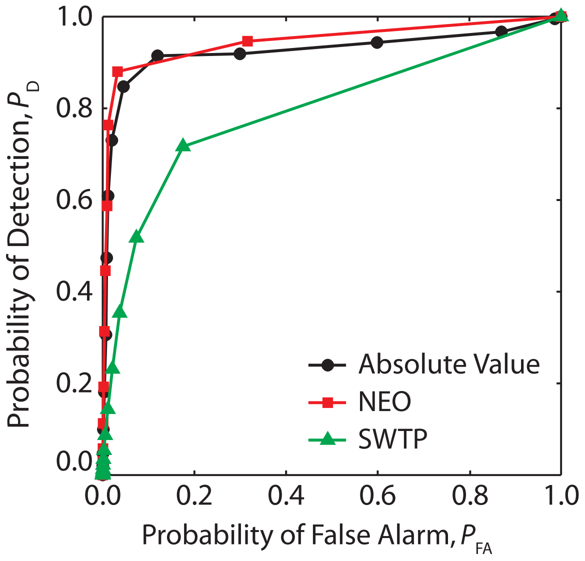


Figure 3.3: Median ROC curve for each detection method ($N = 1632$). The areas under the curves (choice probabilities) are as follows: Absolute Value, 0.925; NEO, 0.947; SWTP, 0.794.

Table 3.1: Complexity of Spike-Detection Algorithms

Algorithm	**MOPS**	**Area [mm^2]**	**Normalized Cost**
Absolute Value	0.4806	0.06104	0.0066
NEO	4.224	0.02950	0.0492
SWTP*	86.75	56.70	2

*Biorthogonal (3,1) wavelet. Smoothing operations not included.

identified it as the optimal spike detection method for our application.

3.5.2 Feature Extraction

Table 3.2 shows the complexity of each feature-extraction algorithm. Note that IT requires offline training in order to determine N_A and N_B (Eq. 2.10), which increases the complexity beyond the numbers in this table. More importantly, PCA requires offline training to determine the PCs, which significantly increases the complexity as it involves eigenvalue decomposition; the numbers in this table reflect only the complexity required to calculate the PC scores. Incidentally, since both IT and PCA require offline training, they are not completely unsupervised algorithms, and this in our view is not optimal for BMI applications

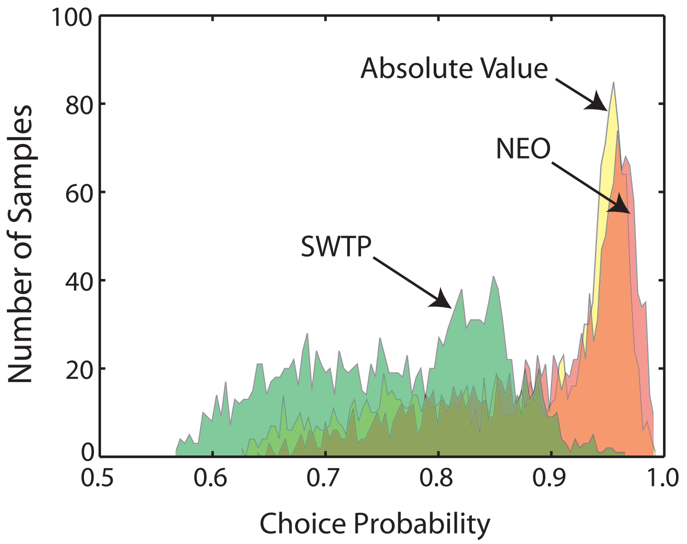


Figure 3.4: Histograms of choice probabilities for each detection method ($N = 1632$). The difference between each distribution is statistically significant (Kruskal-Wallis test, $p < 0.01$). The median of each distribution is: Absolute Value, 0.913; NEO, 0.926; SWTP, 0.772.

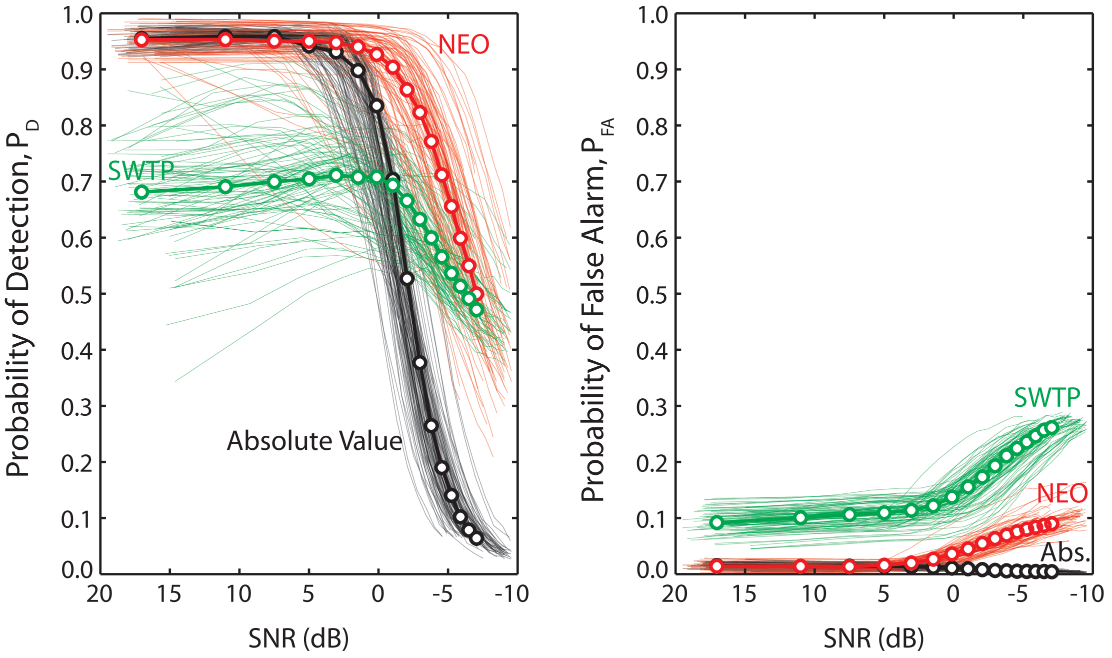


Figure 3.5: *Left*: Probability of detection (P_D) vs. SNR for each detection method. *Right*: Probability of false alarm (P_FA) vs. SNR for each detection method. Curves for each of the 96 datasets are shown. For each method, the median across all datasets is shown in bold.

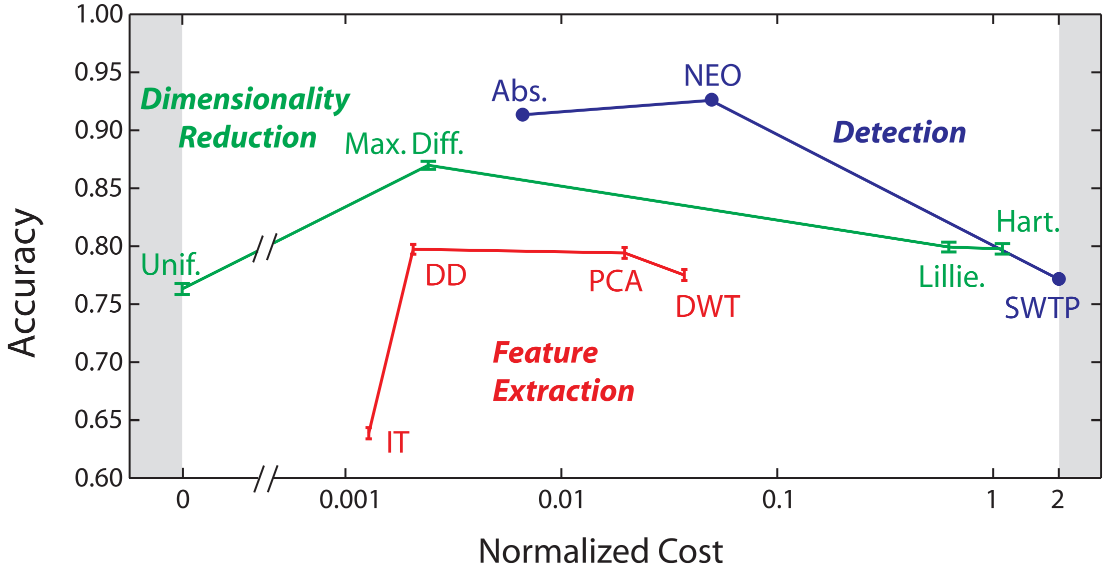


Figure 3.6: The accuracy of all datasets and noise levels ($N = 1632$) vs. normalized computational cost for each spike-detection, feature-extraction, and dimensionality-reduction algorithm. For spike-detection algorithms, "accuracy" represents the median choice probability. For feature-extraction and dimensionality-reduction algorithms, "accuracy" represents the mean classification accuracy after fuzzy c-means clustering, with error bars indicating the standard error of the mean.

anyway. Figure 3.6 shows that DD is the optimal feature-extraction method, since it lies at the knee point of the curve.

3.5.3 Dimensionality Reduction

Because the DWT and DD feature-extraction methods exhibited similar performance, we used both sets of features to test the dimensionality-reduction methods. We did this for two reasons: First, we anticipate some degree of interaction between feature extraction and

Table 3.2: Complexity of Feature-Extraction Algorithms

Algorithm	MOPS	Area [mm^2]	Normalized Cost
PCA*	1.265	0.2862	0.0196
DWT†	3.125	0.06105	0.0371
DD	0.1064	0.04725	0.0021
IT*	0.05440	0.03709	0.0013

*Requires offline training
†Level-5 Haar wavelet

dimensionality reduction. Second, although DD was shown to be less complex than DWT, it could be that in combination with the best dimensionality-reduction method the total complexity is less when DWT is used. Figure 3.7 shows the accuracy of each dimensionality-reduction method, for DWT and DD, as a function of the number of coefficients chosen. Note that uniform sampling is not shown here because its results were quite poor in comparison with the other methods, as will be shown in the next figure. It can be seen from this figure that the accuracy of each method levels off at a dimensionality of less than 39 coefficients. It can also be seen that for each dimensionality-reduction method, DWT reaches its maximum accuracy at a lower dimensionality than DD. However, the final accuracy of each dimensionality-reduction method when used with DD is higher than those when used with DWT. Finally, this figure shows that the peak accuracy is reached at the lowest dimensionality with the maximum-difference method, regardless of the feature-extraction method used.

Figure 3.8 shows a comparison of each dimensionality-reduction algorithm for dimensionalities up to 10, since memory issues in the next step of signal processing, clustering, make it desirable to reduce the dimensionality to at most 10 coefficients. All populations are statistically significantly different from one another (Friedman Test, $p < 0.001$ in each case). The performance of uniform sampling is obviously sporadic and in general lower than that of the other methods. For both feature-extraction methods, the maximum-difference method outperforms both Lilliefors and Hartigan's Dip tests, making it the optimal algorithm, in terms of accuracy, for dimensionality reduction. When a dimensionality of 10 DD coefficients is used, the maximum-difference method achieves an average accuracy of about 87%.

The complexity of each dimensionality-reduction algorithm is shown in Table 3.3, and Fig. 3.6 shows the accuracy of each method (averaged over all datasets and SNRs) versus the computational cost. While uniform sampling has a cost of 0, it also has the lowest accuracy. On the other hand, the maximum-difference algorithm has the highest accuracy, with a cost of around 3 orders of magnitude less than that of the Lilliefors Test and of Hartigan's Dip Test. Thus, we chose the maximum-difference algorithm as the optimal dimensionality-reduction method.

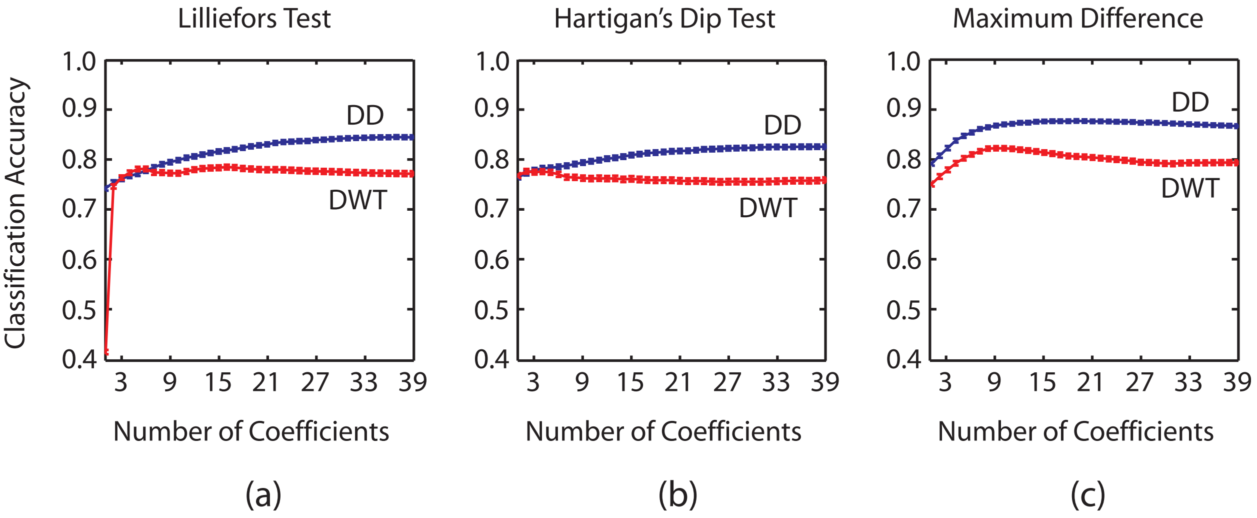


Figure 3.7: Mean classification accuracy of each dimensionality-reduction algorithm (a: Lilliefors Test, b: Hartigan's Dip Test, c: Maximum Difference test), averaged over all datasets and noise levels ($N = 1632$), after fuzzy c-means clustering, vs. dimensionality. Error bars show standard error of the mean.

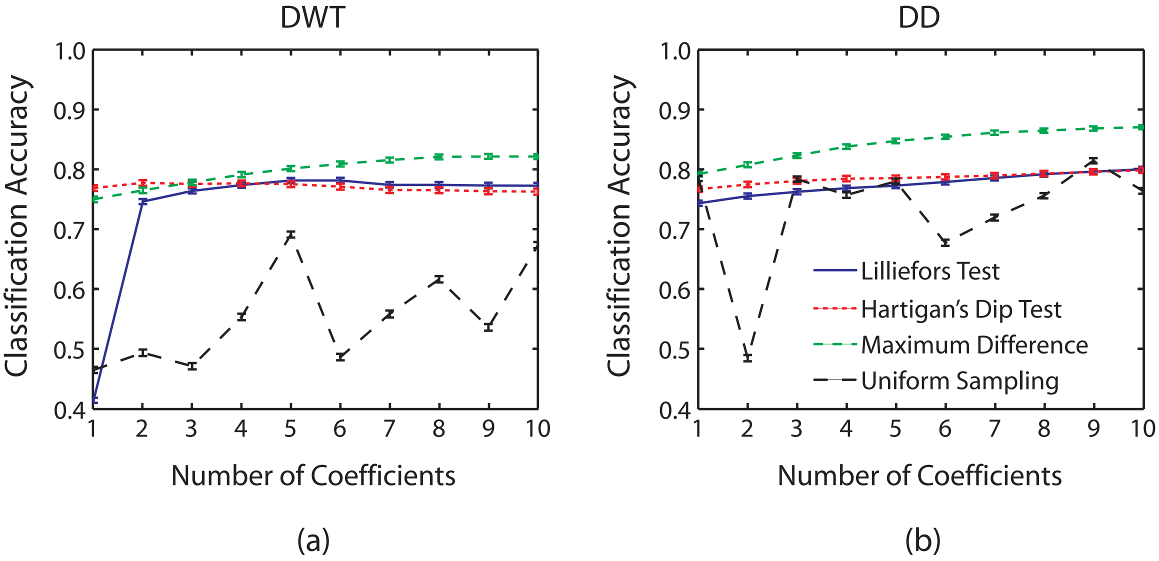


Figure 3.8: Mean classification accuracy of each dimensionality-reduction algorithm when using (a) DWT feature extraction and (b) DD feature extraction, averaged over all datasets and noise levels ($N = 1632$), after fuzzy c-means clustering, vs. dimensionality. Error bars show standard error of the mean. All populations are statistically significantly different from one another (Friedman Test, $p < 0.001$ in each case).

Table 3.3: Complexity of Dimensionality-Reduction Algorithms

Algorithm	MOPS	Area [mm^2]	Cost
Lilliefors Test	39.32	9.642	0.6233
Hartigan's Dip Test	77.72	11.81	1.1042
Maximum-Difference	0.1536	0.03679	0.0024
Uniform Sampling	0	0	0

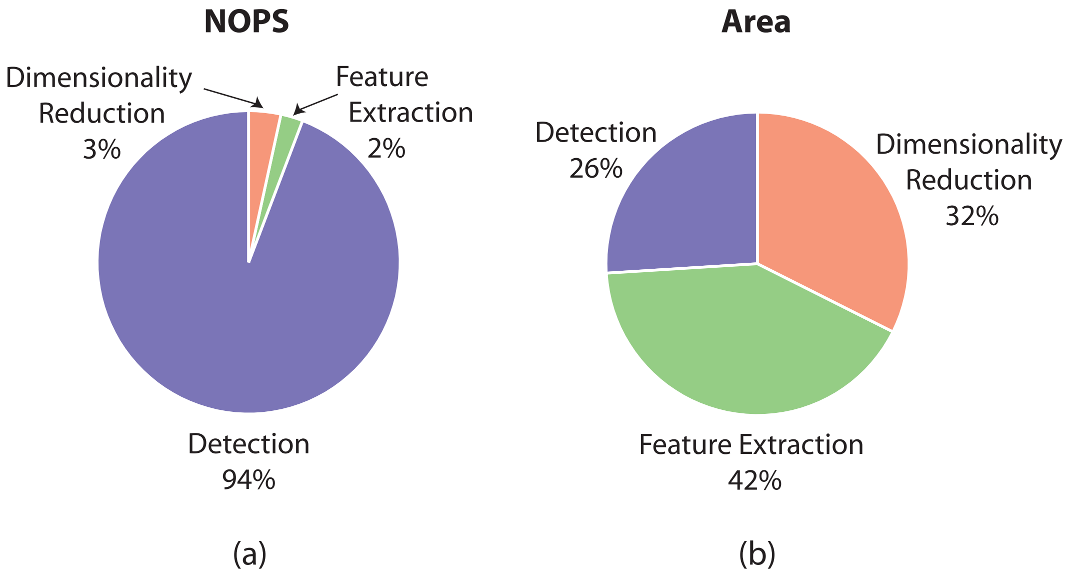


Figure 3.9: Pie charts indicating total system complexity: (a) OPS, (b) area.

3.6 Discussion

In Section 3.5, we determined that NEO, DD, and maximum-difference were the optimal algorithms for our application. Figure 3.9 shows a block-by-block breakdown of the overall system complexity when these methods are used. As shown in Fig. 3.9(a), the required NOPS is heavily dominated by detection. This is due to the fact that the detection hardware must operate on every single incoming sample, whereas other components only operate in the presence of a spike. Figure 3.9(b) shows that the total system area is more evenly distributed between components.

As suggested by Fig. 3.10, significant data reduction is achieved from spike sorting. Assume that we record from 100 channels, each channel having 3 neurons, each neuron having a firing rate of 20 spikes per second, an average firing rate for a cortical neuron

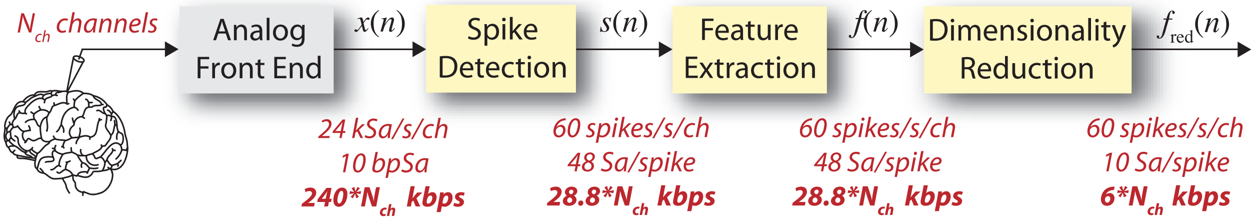


Figure 3.10: Example of data-rate reduction provided by each step of the spike-sorting process for an $N_{\rm ch}$-channel recording system. The data rate at the end of spike sorting is lower than that of the raw data. (Assumptions are annotated on the figure in red italics.)

Table 3.4: Overall System Complexity

	MOPS	Area [mm^2]	Data Rate* [Mbps]
Detection (NEO)	4.22	0.0295	2.88
FE (DD)	0.106	0.0473	2.88
DR (Max. Diff.)	0.154	0.0368	0.60
Overall System	4.48	0.114	0.60

*Assumptions: 100 ch, 3 cells/ch, 20 Hz/cell, 10 bpSa.

responding to its preferred stimulus. If in spike detection we take 48 samples per spike with a resolution of 10 bits per sample, this reduces our data rate from 24 Mbps to 2.88 Mbps—an 88% reduction. Feature extraction and dimensionality reduction can further reduce the data rate; when the dimensionality is reduced to 10 coefficients per spike, the data rates are effectively further reduced by 79%. Again assuming 100 channels and 3 neurons per channel at 20 spikes per second per neuron, the total data reduction achieved from detection, feature extraction, and dimensionality reduction is 95% (when 10 coefficients are selected). These results are summarized in Table 3.4.

3.7 Conclusion

Based on the present analysis, we advocate the NEO for spike detection, DD for feature extraction, and the maximum-difference test for dimensionality reduction. NEO would require an estimated area of about 0.03 mm^2, DD 0.05 mm^2, and maximum difference 0.04 mm^2, for a system total area of less than 0.12 mm^2 per channel. Assuming that area scales linearly with the number of channels, this equates to less than 12 mm^2 for 100 channels, which is feasible for implantable devices.

The area of the overall system is so far dominated by feature extraction, and the overall system power (as indicated by NOPS) by detection. We estimate that once clustering is added to the system, the overall area and power will both be dominated by clustering, which will be at least 3× more complex than spike detection, feature extraction, and dimensionality reduction combined.

The next step is to similarly evaluate existing methods for alignment at the front end and clustering at the back end of the spike-sorting process. A challenge will be to find an online, unsupervised, accurate, and computationally simple clustering algorithm for use in a hardware spike-sorter. Finally, verification methods, such as autocorrelograms and cross-correlograms, should be employed. Another direction to be explored is adaptive spike-sorting algorithms, which could adapt to moving electrodes/tissue and which would be more stable for long-term recordings.

Acknowledgments

The authors thank Dr. Y. Ben-Shaul for use of his neural-signal simulator, as well as Mr. V. Karkare for comparing register-bank and SRAM memories.

Permissions

Substantial portions of this chapter have been reprinted, with permission, from: S. Gibson, J. W. Judy, and D. Marković, “Technology-Aware Algorithm Design for Neural Spike Detection, Feature Extraction, and Dimensionality Reduction,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 18, no. 4, pp. 469–478, Oct. 2010. Copyright © 2010, IEEE.

Author contributions: J. W. Judy and D. Marković were the PIs.

CHAPTER 4

Analog vs. Digital Spike Detection

4.1 Introduction

Because of the strict limitations on power density for implanted electronics, all hardware must be carefully optimized in terms of power and area in order for them to be safely implantable. One question that arises when optimizing spike-detection hardware is whether spike detection should be performed in the analog or digital domain.

Many existing systems perform spike detection in the analog domain (e.g., [19, 20, 102–105]), while others choose to perform spike detection in the digital domain (e.g., [106–108]). The assumption is that analog spike detection is more power-efficient since the ADC would only need to run when there are spikes, whereas in digital detection the ADC must constantly be running since detection occurs only after sampling (Fig. 4.1). However, performing computations in the digital domain has the advantage that digital-design techniques that are not possible in the analog domain, such as voltage scaling and interleaving, can be employed. In this chapter, we investigate whether analog or digital spike detection is more efficient, with respect to both power and area. Circuits are simulated in 90-nm bulk CMOS, with thick oxide 250-nm transistors used for some parts of the analog designs.

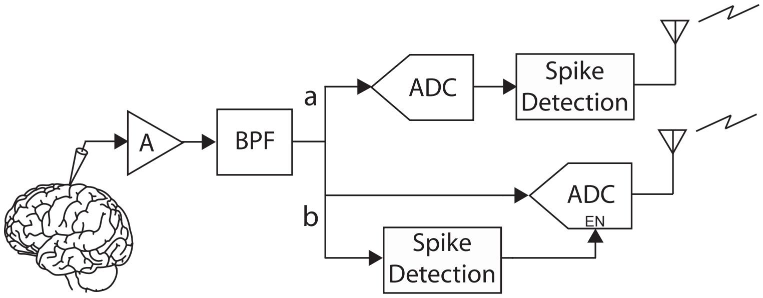


Figure 4.1: Block diagram for (a) digital spike detection and (b) analog spike detection.

4.2 Methods

4.2.1 Algorithms

4.2.1.1 Spike-Detection Algorithms

We wanted to include algorithms representing different levels of computational complexities in our analysis. Based on the study in Chapter 3, we chose absolute-value thresholding and the nonlinear energy operator, which differ in complexity by about one order of magnitude while maintaining similar performance.

As described in Chapter 2, in absolute-value thresholding, a threshold is applied to the absolute value of the waveform $x(n)$ [45], and in the nonlinear energy operator (NEO) method [45, 49–51], a threshold is applied to the NEO ψ:

$$\psi[x(n)] = x^2(n) - x(n+1) \cdot x(n-1). \tag{4.1}$$

4.2.1.2 Modes of Operation

We also chose to analyze each algorithm for two different modes of operation (Fig. 4.2). In `Pulse Output` mode, the spike detector just outputs a pulse when the signal ($|x|$ or ψ) crosses the threshold. Most published analog spike detectors operate in this mode. It is the simplest mode, since it requires neither memory nor an ADC (in the case of analog spike detection). However, in applications that require single-unit activity, spike sorting must

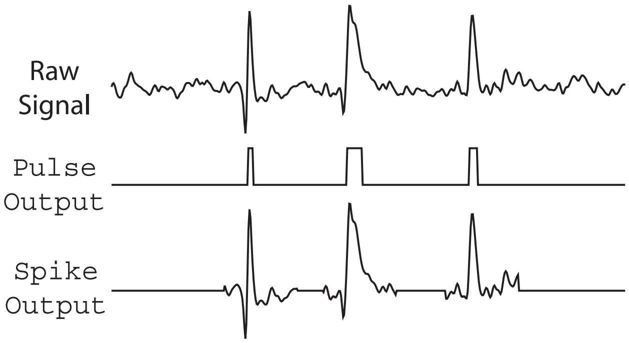


Figure 4.2: Outputs of spike detector for each mode of operation. In `Pulse Output` mode, a pulse is outputted each time the waveform crosses the threshold. In `Spike Output` mode, spike samples are outputted for subsequent spike sorting.

be performed following spike detection. If only spike times are outputted, then the spike shapes are lost, making subsequent spike sorting impossible. Therefore, the second mode of operation that we analyzed (`Spike Output`) transmits 1-ms-worth of waveform samples before the threshold crossing (the "spike preamble") and 2-ms-worth of waveform samples after the threshold crossing. Although 3 ms is much longer than a typical spike, this provides a sufficient number of samples for subsequent alignment.

4.2.2 Analog-to-Digital Converter (ADC)

The power (P_{ADC}) and area (A_{ADC}) of the ADC are modeled by Eq. 4.2 and Eq. 4.3. To compare the power of ADCs of different speeds (F_{S}), resolutions (B), and technology, the Figure-of-Merit (FoM) shown in Eq. 4.2 is used, where A_0 and B_0 are the area and resolution, respectively, of the baseline ADC described below.

$$P_{\mathrm{ADC}} = FoM \cdot F_{\mathrm{s}} \cdot 2^{B} \tag{4.2}$$

$$A_{\mathrm{ADC}} = A_0 \times 2^{2\cdot(B-B_0)} = 0.03 \times 2^{2\cdot(B-8)} \tag{4.3}$$

Based on recent published work (such as [109–111]), ADCs with FoMs on the order of 100 fJ/conv-step are readily achievable in the 8 to 12 bit range. Figure 4.3 shows a survey of

ADCs against different FoMs. As technology and circuit architectures improve, the FoM also improves. In the future, FoMs ten times lower may be commonplace [111]. Power increases linearly with speed, and exponentially with resolution (and often faster than 2^B). Area is primarily determined by the resolution. For an 8-bit, 24-kSa/s, 100-fF/conv-step ADC, the power consumption is expected to be 614 nW. Silicon area ranges from 0.021 to 0.24 mm^2; for the estimates below we use 0.03 mm^2 (A_0) and 8 bits (B_0) as our baseline ADC. To scale to other resolutions (B), we use Eq. 4.3. While the power dissipation of the high-efficiency ADCs previously mentioned is dominated by dynamic power, we allocate 10% of the reported power as standby power (for biasing and references). Including readout circuitry (Sec. 4.2.3.3) for buffered samples, the effective power is given by Eq. 4.4.

$$\begin{aligned} P_{\mathrm{ADC,eff}} &= (0.9 \cdot FoM \cdot 2^B \\ &\quad + P_{\mathrm{buf}}/F_{\mathrm{s}}) \cdot r_{\mathrm{D}} \\ &\quad + 0.1 \cdot FoM \cdot 2^B \cdot F_{\mathrm{s}} \end{aligned} \tag{4.4}$$

where F_{s} is the sampling rate (24 kSa/s) and r_{D} is the detection rate in samples per second, defined later in Eq. 4.5.

We also assume that interleaving the ADC with up to 64 channels has little impact on the overall performance (which is the case for fine-line CMOS).

4.2.3 Analog Spike Detection

Before we consider different implementations of the analog spike detectors, we will briefly consider the signal levels. With this information, we are able to specify tolerable degradation of the analog circuit (such as noise and offsets). As shown in Fig. 4.1, the detector circuitry is placed after amplification. Typical extracellular spike amplitudes are in the 50 to 500 μV range, with worst-case noise of 10 μV. With a preamp voltage gain of 100×, the noise at the detector and ADC input is 1 mV. The analog detector must keep its own electronic

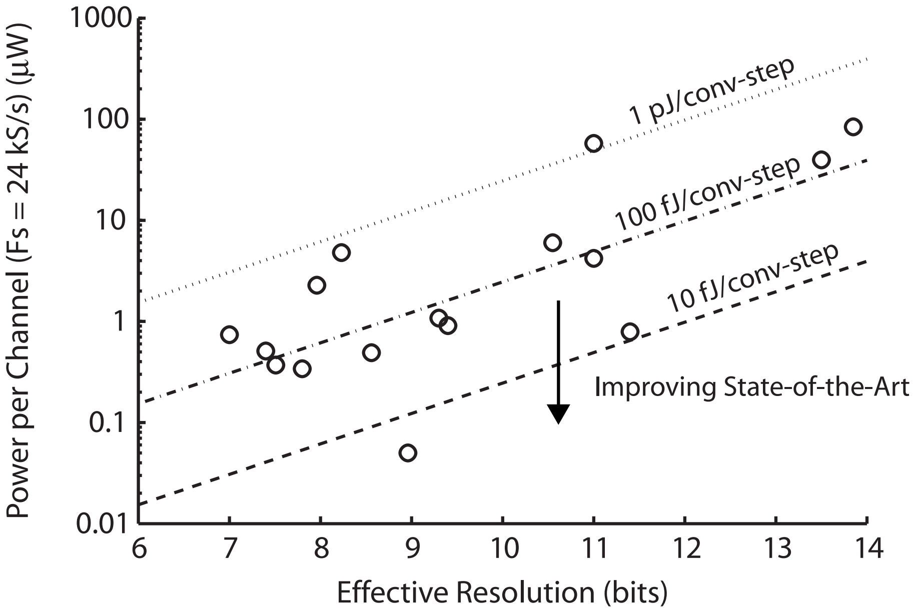


Figure 4.3: Performance of recent ADCs, and FoM contours. 100 fJ/conv-step is readily achievable, with better than 10 fJ/conv-step demonstrated.

thermal noise and offsets below this level. While this is achievable with modest power, we will compare the analog and digital implementations to determine which is optimal. We will only describe the dominant power and area contributors; clock power, for instance, was found to be negligible at the operation frequency of neural spike recording. The supply voltage assumed is fixed at 1 V for the analog portions. Area calculations are based on total active MOSFET area ($W \times L$) and capacitors with a specific capacitance of 1 fF/μm^2.

4.2.3.1 Absolute-Value Threshold Detector

Absolute-value thresholding can be performed with a clocked comparator and a Switched-Capacitor (SC) difference circuit. The primary error is the comparator offset voltage. Smaller offsets can be achieved by increasing the device area at the cost of power dissipation. The other significant error source is charge injection from the switches.

The comparator shown in Fig. 4.4 draws approximately 0.176 μA from a 1-V supply. The gate area of the input devices is 10 μm^2, yielding a random offset of 10.8 mV (in 90-nm CMOS). Assuming an ADC full-scale of 500 mV, this offset would result in a 10% error in

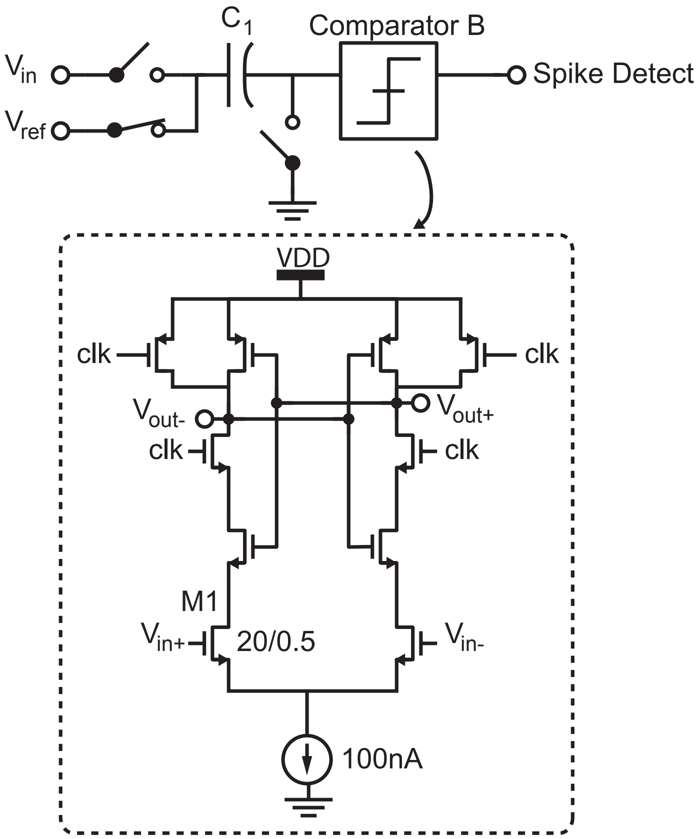


Figure 4.4: Schematic for a low-power dynamic comparator.

the desired threshold. However this could be overcome by circuit techniques such as auto-zeroing, or by increasing the device areas. The area is dominated by the sampling capacitor C_1 (500 fF).

A reference circuit is also required. The supply current of approximately 0.2 μA would drive the input capacitance of the comparator in its comparison phase. The total power of an analog spike detector is $2\times0.18 + 0.2 = 0.54$ μW.

4.2.3.2 Analog Nonlinear Energy Operator (NEO) Detector

Because analog differentiation is prone to being noisy, because its time constant is sensitive to process variation, and for a more direct comparison to the digital implementation, we implement the discrete-time version of the NEO as shown in Eq. 4.1.

In order to implement Eq. 4.1, we require an analog multiplier and analog memory. The

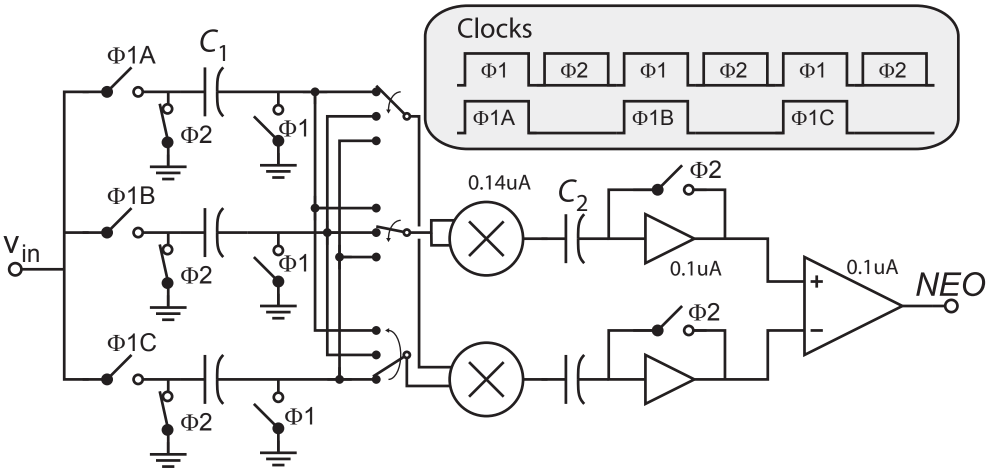


Figure 4.5: Implementation of the nonlinear energy operator in the discrete-time analog domain.

algorithm can be implemented with the circuit shown in Fig. 4.5. After the first half of the clock period (Φ1), a new sample is buffered in the C_1 array. During Φ2, the products $x^2(n)$ and $x(n-1) \cdot x(n+1)$ are computed from the corresponding capacitor voltages. A commutator after the capacitor array routes the correct sample to the multipliers, and the routing is updated each clock cycle. The products are stored on C_2. Auto-zeroing of the multipliers is achieved during Φ2, with C_2 implementing Output Offset Storage (OOS). This allows smaller devices to be used in the multipliers to save area.

Because the linear input range of a typical Gilbert multiplier is on the order of ± 50 mV, only limited gain can be applied to the signal. Hence thermal noise of the multiplier may cause excessive degradation of the SNR. Simulations show that 140-nA tail current for the multiplier is sufficient for 150-μV noise (1.5 μV at the preamplifier input).

The power of the combined circuitry (2 multipliers and 3 amplifiers, plus comparator) is $2 \times 0.14 + 3 \times 0.1 + 0.38 = 0.96$ μA. The analog NEO circuit requires a total of 5 pF of capacitance, so the total area is approximately 5000 μm^2.

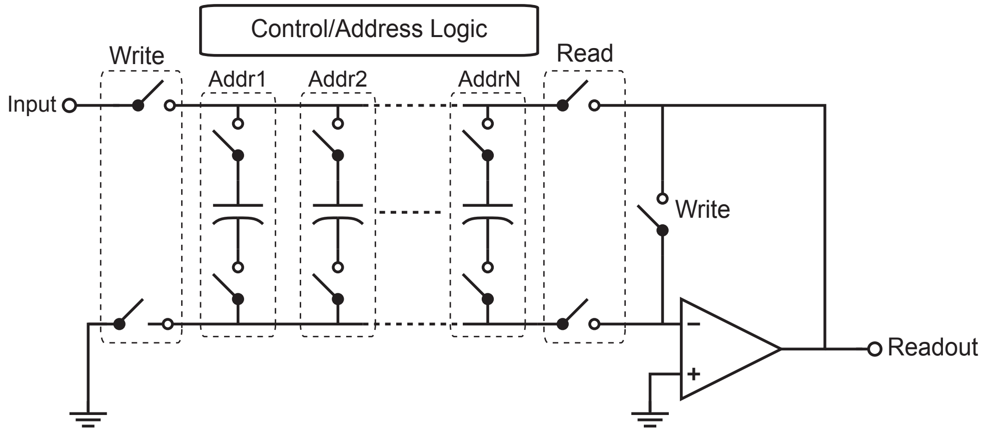


Figure 4.6: Implementation of analog memory for buffering of the signal before a spike has been detected.

4.2.3.3 Analog Memory

In some applications, it is advantageous to retain the samples before the spike detect event. This is straightforward in digital, but it is somewhat difficult to implement in analog. One solution, by Anelli [112], is shown in Fig. 4.6. In other applications that do not require the spike preamble, we can ignore this power/area.

Setting the storage capacitance (C_1) as 100 fF meets kT/C (thermal) noise requirements. The total area is computed as 24 samples × 100 fF × 1 fF/μm^2 × 2 (for a differential implementation) yielding a total area around 4800 μm^2 active area.

After a spike is detected, the memory must be read by the ADC. A buffer, in the form of a Flip-Around Track-and-Hold, provides good linearity. To estimate the power of this amplifier, we assume a two-stage OTA. Behavioral simulations show that $g_{\mathrm{m1}} = 0.564$ μS is required to meet the settling time requirement. The total opamp current, assuming $g_{\mathrm{m2}} = 3g_{\mathrm{m1}}$ is then $8 \times (g_{\mathrm{m1}}+g_{\mathrm{m2}}) \times V_{\mathrm{T}}/\kappa = 180$ nA. Our estimate for the total amplifier is 1.5× the area of the compensation capacitors (2 pF), for a total area of 3000 μm^2.

4.2.4 Effect of SNR and Firing Rate on Analog Detection Power

Since SNR and firing rates can vary significantly across neural recordings, it is important to check the validity of this analysis for a wide range of SNRs and firing rates. For example, if the rate of detection increases as SNR decreases due to false alarms, then the power of analog spike detection would also increase due to more frequent use of the ADC. If so, there could be a range of SNRs and firing rates for which digital detection is more efficient. Therefore, we estimated the variation in power consumption for the analog detection hardware with SNR and firing rate.

We generated data with the neural signal simulator used in [76] for SNRs ranging from about 15 dB to -10 dB. We then performed spike detection using both algorithms, using the automatic threshold calculation techniques described in [76], and calculated the probability of detection (P_{D}) and the probability of false alarm (P_{FA}) at each SNR. These rates were then used to calculate the detection rate r_{D} in samples per second for each SNR using Eq. 4.5:

$$r_{\mathrm{D}} = \max\{r_{\mathrm{N}} \cdot l \cdot P_{\mathrm{D}} + (F_{\mathrm{S}} - r_{\mathrm{N}} \cdot l) \cdot P_{\mathrm{FA}}, F_{\mathrm{s}}\}, \tag{4.5}$$

where r_{N} is the firing rate of the neurons (which can be the sum of firing rates of multiple neurons) in spikes per second and l is the length of a spike in samples per second. r_{D} can be thought of as the number of samples that the ADC must convert/quantize per second. Note that the maximum value that r_{D} can take is F_{s}. This equation was used in Eq. 4.4 to calculate the power of the ADC.

Figure 4.7 shows the variation in power of the analog implementation of NEO, `Spike Output` mode. The detection rate, and therefore the power, remain constant across SNRs until around -5 dB, when the number of detections (power) begins to decrease. This is due to the adaptive nature of the threshold, which is based on a multiple of the mean of the NEO, and which, therefore, increases as the noise increases. Figure 4.7 also shows that the power increases linearly with the firing rate, due to the linear increase in detection rate with firing rate, with a maximum variation in power of about 600 nW. (Similar results, not shown, were

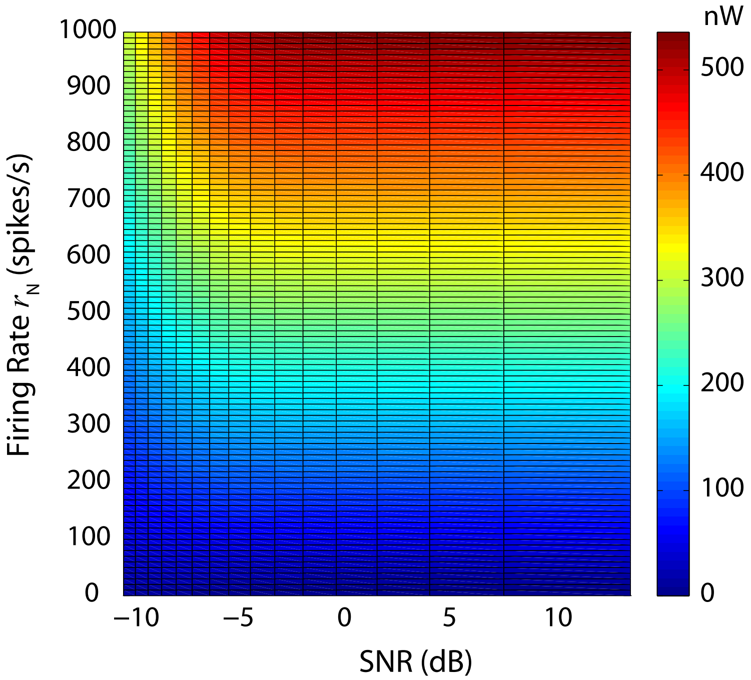


Figure 4.7: Variation in power of analog NEO spike detection, `Spike Output` mode, due to changes in SNR and neuronal firing rates. (Note: Firing rate can represent the sum of firing rates from multiple neurons.) The variation was calculated by subtracting the minimum power from each value. The power remains constant across SNRs until about -5 dB, at which point it begins to decrease, and the power increases steadily with firing rates, until saturation, when the ADC is operating at its maximum 24 kSa/s.

obtained for the absolute-value method.) To simplify the analysis, we will use an operating point of 1.3-dB SNR and 100-Hz firing rate when presenting the results in Section 4.3.

4.2.5 Digital Spike Detection

In order to obtain power and area estimates for the digital implementations of the spike-detection algorithms, both the absolute-value threshold and the NEO detection methods (`Spike Output` and `Pulse Output` modes as explained earlier) were implemented in the Matlab/Simulink-based design environment. Each of the above algorithms was implemented with 2-, 4-, 8-, 16- and 32-channel data-stream interleaving to determine a power–area efficient implementation. The RTL was auto-generated from the Simulink model using the Synplify DSP blockset. Power and area estimates were then obtained from the synthesis reports for these designs when synthesized with DC compiler from Synposys. Simulated neural data was input to RTL simulations to obtain switching activity estimates for the design. These estimates were then annotated into the synthesis flow to obtain power estimates for the digital spike-detection module.

Based on technology evaluation results for our design in 90-nm bulk CMOS process, we chose to operate the circuits at a reduced supply voltage of 0.4 V. Since standard-cell libraries are characterized for the nominal supply voltage (1 V), we specified a higher clock frequency for synthesis in order to account for the increase in delay due to supply voltage scaling. We also evaluated the reduction in leakage power due to supply voltage scaling for basic gates. The switching power and leakage power numbers obtained from synthesis were thus scaled down to their corresponding values at 0.4 V to make comparisons for power consumption at 0.4 V.

Figures 4.8 and 4.9 show the area and power per channel versus the number of channels interleaved. Interleaving usually increases the power due to increased switching activity of logic and a similar number of registers switching at a faster rate. However, if the supply voltage is scaled, savings in the leakage power of logic and the increase in switching power are comparable. Thus, the total power consumed per channel versus degree of interleaving has a global minimum. From the above results we found 8-channel interleaving to be a power–area efficient implementation for the detection algorithms considered. We also observed that area and power for `Spike Output` mode are significantly higher than those for `Pulse Output` mode. This is due to the additional logic and memory required to provide the detected spikes (with preamble) as the output. It should be noted that we used a register-based memory for our design to guarantee functional operation at 0.4 V. However, custom low-voltage memory would reduce the power difference between `Pulse Output` and `Spike Output` mode implementations. We found that variation in SNR and firing rate does not cause significant variations in the power consumed by the DSP. This result is expected, due to two major reasons: a) SNR and firing rate do not affect the ADC power for digital detection; and b) SNR and firing rate only affect the switching power of a portion of the DSP, which does not cause a significant change in the total power of the DSP scaled to 0.4 V. Hence, we expect the results of the above analysis to be valid for a wide range SNR and firing rates.

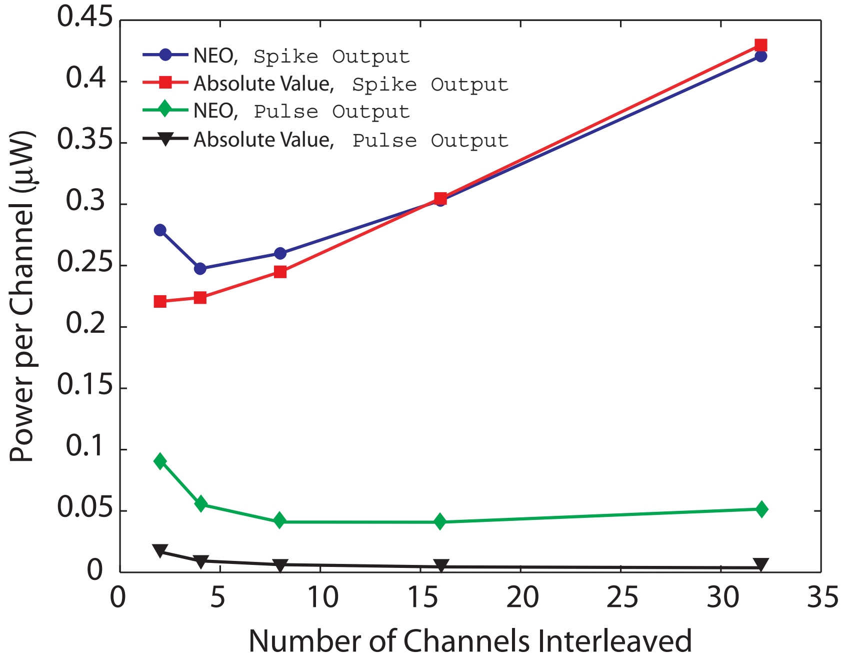


Figure 4.8: Power estimates obtained from Synopsys for NEO, `Spike Output` mode. The total power (P_{total}) is divided into switching power ($P_{\text{switching}}$) and leakage power (P_{leakage}).

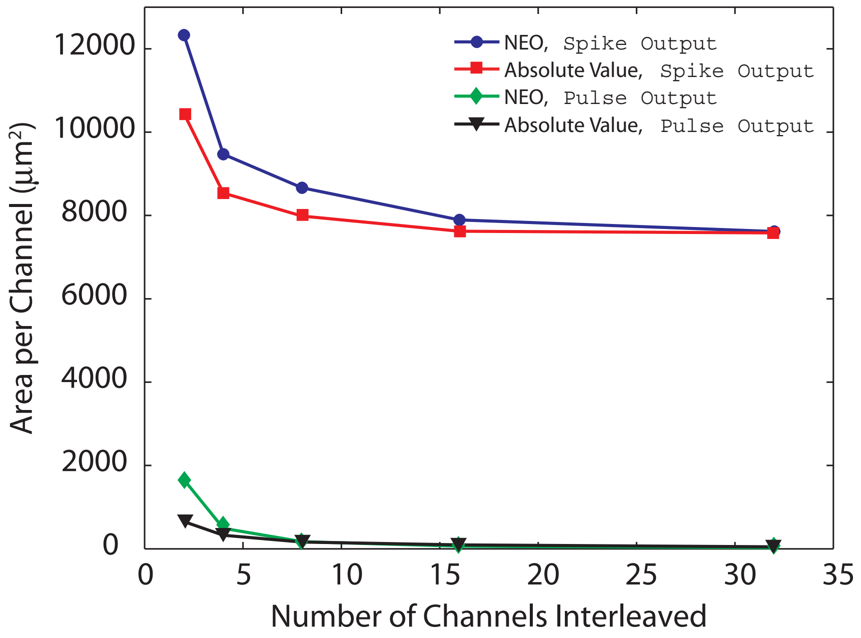


Figure 4.9: Area estimates for NEO, `Spike Output` mode, obtained from Synopsys as a function of the number of channels interleaved.

4.3 Results

Figure 4.10 shows the power per channel and area per channel for each algorithm and output mode. The first row of plots corresponds to power per channel, and the second row of plots corresponds to area per channel. Solid red (green) lines correspond to the total power/area per channel of analog (digital) detection, including the power/area of the ADC. The solid red (green) line can be decomposed into the power/area of detection alone, indicated by the dashed red (green) line, and the ADC power when operating at the maximum rate (F_s), indicated by the dashed black line.

The power for analog detection in `Pulse Output` mode is constant across bit resolution. In the case of the `Spike Output` mode, the power for analog detection does increase with bit resolution, since the ADC power increases. However, the increase in analog detection power due to the ADC is less than that for digital detection. This is because the ADC is only active for a limited time in case of analog detection. As for the area tradeoff, there is a crossover between analog and digital implementations in the case of `Pulse Output` mode. At higher bit resolution, the area for analog implementation is less since the area of the ADC dominates. For the `Spike Output` mode, however, the area of digital detection is less than that of the analog detection.

From these plots we can conclude that digital spike detection is better for resolutions of up to 8 or 9 bits, depending on the algorithm used. Until this point, the ADC power is small. Hence the saving in ADC power achieved by analog detection does not outweigh the lower power cost of the DSP implementation. However, since the power and area of the ADC scale exponentially with bit resolution, the ADC starts to dominate at higher resolution. The analog detection, therefore, is more power-efficient in this domain.

4.4 Conclusion

In this chapter, we have compared the power consumption and the area of analog and digital spike detection. We demonstrated that power is not a strong function of SNR or firing rates;

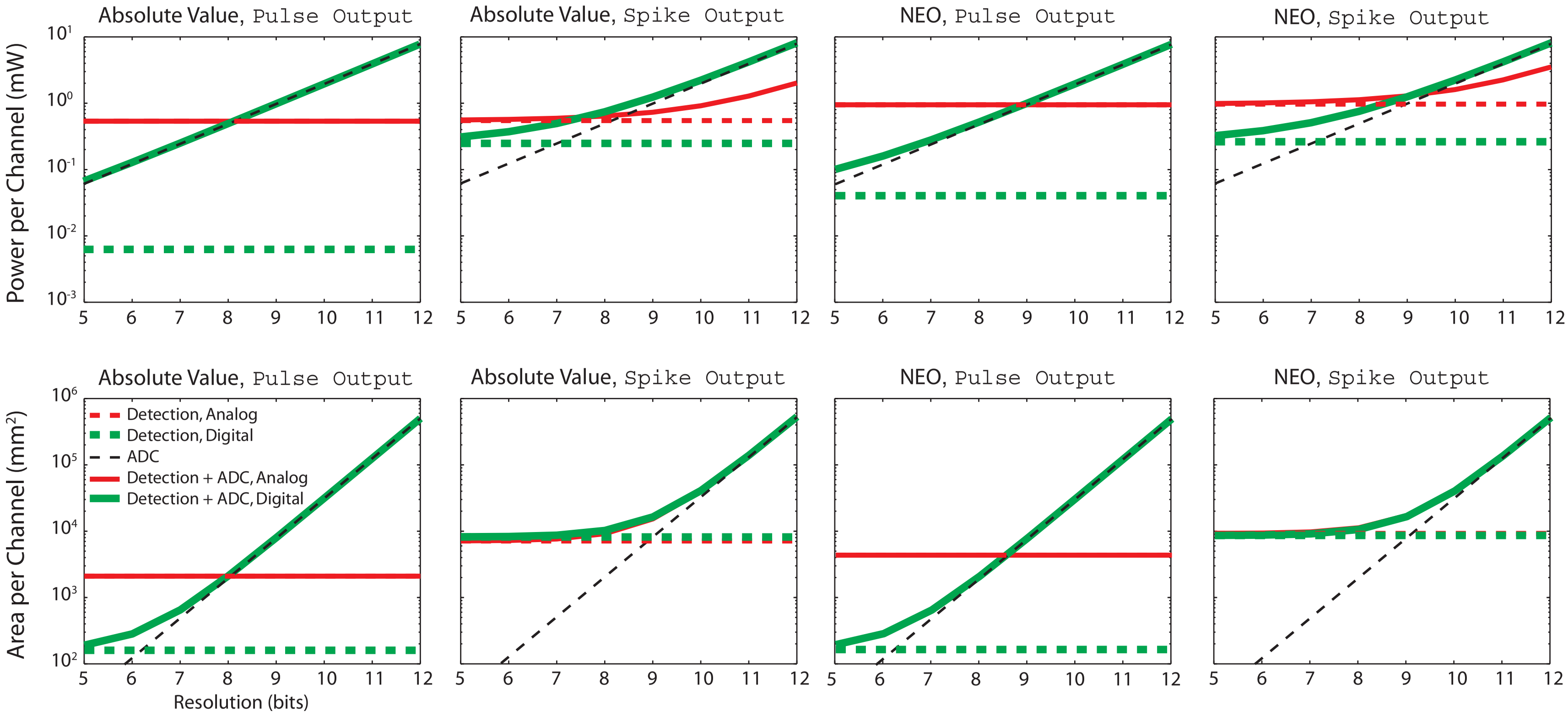


Figure 4.10: Power per channel and area per channel for each algorithm and output mode. The first row of plots corresponds to power per channel, and the second row of plots corresponds to area per channel. Solid red (green) lines correspond to the total power/area per channel of analog (digital) detection, including the power/area of the ADC. The solid red (green) line can be decomposed into the power/area of detection alone, indicated by the dashed red (green) line, and the ADC power/area when operating at the maximum rate (F_s), indicated by the dashed black line. For each algorithm and output mode, a crossover point exists between analog and digital, indicating that for lower resolutions digital spike detection is more power/area- efficient, while for higher resolutions analog spike detection is more power-efficient.

thus, the results shown for the operating point 1.3 dB, 100 Hz are valid across a wide range of SNRs and firing rates. We also showed that the tradeoff between digital and analog detection is a strong function of the bit resolution. For lower resolutions, digital implementations are more efficient, whereas for higher resolutions, analog implementations are more efficient. Therefore, the choice of whether to implement hardware spike detection in the analog or digital domain is dependent on the desired resolution.

Results in this work were based on an implementation in 90-nm bulk CMOS process. Future work could be to analyze the tradeoff between analog and digital detection across different technologies. The automatic-threshold-calculation block should also be incorporated into the analysis. We could also consider alternative circuit realizations that incorporate multiple recording channels.

Permissions

Substantial portions of this chapter have been reprinted, with permission, from: S. Gibson, R. Chandler, V. Karkare, D. Marković, and J. W. Judy, "An Efficiency Comparison of Analog and Digital Spike Detection," in *Proc. 4th Int. IEEE EMBS Conf. Neural Eng.*, Antalya, Turkey, 2009, pp. 423–428. Copyright © 2009, IEEE.

Author contributions: R. Chandler provided the analog circuit analysis, and V. Karkare provided the digital circuit analysis. D. Marković and J. W. Judy were the PIs.

CHAPTER 5

The Effects of Quantization on Spike Sorting

5.1 Introduction

In the previous chapter, we looked at minimizing the system power by optimizing the detection block. Another important design consideration that can have a significant impact on the total system power is the type and resolution of the quantizer used in data acquisition. For systems that are required to transmit the data wirelessly (Fig. 5.1a), the transmitter power will increase roughly linearly with the number of bits, and for systems that perform all processing on-chip (Fig. 5.1b), the power and area of the DSP will increase at least linearly with the number of bits. Yet there is no consensus within the recording community over the necessary resolution of neural signals. Most commercial (wired) data-acquisition systems provide 12- to 24-bit resolution [22–27]. A few researchers, however, have suggested that substantially fewer bits are necessary. For example, the authors in [107] performed an analysis of quantization noise versus total system noise and reported an optimal quantization level of 5 bits. The authors in [113] and [114] (independent studies) estimated the required number of bits by setting the dynamic-range-to-noise ratio (DNR) of the ADC equal to the "DNR" of the input signal, and concluded that 8 and 6 bits were needed for their respective systems.

The required resolution is likely application-dependent. Because in this thesis we are particularly interested in spike sorting, here we will assume that all data-acquisition hardware has been moved to the implant, where data must be processed via spike sorting on-site before wireless transmission of the results, and we will examine the effects of both uniform and optimal quantization on spike-sorting performance. We will begin by describing the

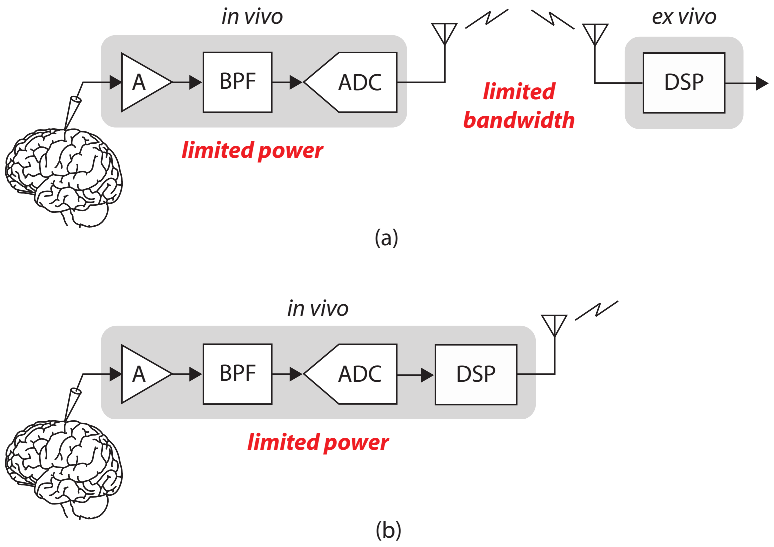


Figure 5.1: Block diagrams for two potential recording systems. (a) Raw data is transmitted wirelessly and processed off-site. (b) Data is processed on-site and only the results are transmitted wirelessly.

spike-sorting algorithms on which we tested these effects.

5.2 Spike-Sorting Algorithms

With the DSP now on the transmit side, all spike-sorting algorithms must be fully autonomous. Thus, we first looked for algorithms for each of these tasks that are completely unsupervised; from among the available unsupervised algorithms, we then chose algorithms that have demonstrated good performance in previous studies and that are realizable within the strict power budget to which implantable electronics are subject.

Spike detection was performed by applying a threshold to the absolute value of the amplitude of the waveform (Section 2.4.1). This method has been shown to be accurate for SNRs above 0 dB [64]. Because the spike-sorting system must be completely autonomous, the threshold must be calculated automatically. We used the automatic-threshold-calculation method described in Eq. 2.1 and Eq. 2.2, reprinted here for convenience:

$$Thr = 4\hat{\sigma}_{\mathrm{N}}, \tag{5.1}$$

$$\hat{\sigma}_{\mathrm{N}} = \text{median}\left(\frac{|x(n)|}{0.6745}\right), \tag{5.2}$$

where Thr is the detection threshold, $\hat{\sigma}_{\mathrm{N}}$ is an estimate of the noise standard deviation (SD), and $x(n)$ is a sample of the original signal x at time n. Spikes were extracted from data with a sampling rate of 24 kHz by taking 23 samples before the threshold-crossing sample and 48 samples after, resulting in a 3-ms spike of dimensionality $D = 72$. Following spike detection, each spike was aligned by placing a 48-sample window around the spike such that the maximum amplitude occurred at sample number 16, resulting in aligned spikes of dimensionality $D = 48$.

After spikes were detected and aligned, classification was performed using the Osort online sorting algorithm [69] (Section 2.4.5), which has been shown to have good clustering accuracy and is in use in several neuroscience laboratories. A summary of the algorithm is repeated here for convenience:

1. Initialization: Assign the first spike to its own cluster.
2. Calculate the Euclidean distance between the next spike and each cluster centroid.
3. If the smallest distance is less than the merging threshold T_M, assign the spike to the nearest cluster and recompute that cluster's mean using the X most recent spikes. Otherwise, start a new cluster.
4. Check the distances between each cluster and every other cluster. If any distance is below T_M, merge those two clusters and recompute its mean.

Steps 2–4 are then repeated indefinitely. In spike sorting, we typically assume that each spike observed is the sum of the signal (the deterministic spike shape for a given neuron) and a random noise variable. Thus, when viewing spikes in D-dimensional space (Fig. 5.2), we will see points clustered around each signal mean according to the noise probability distribution. Osort essentially limits the radius of each cluster to T_M, so it makes sense that T_M should be a function of the amount of noise in the system. The author of [69] suggests setting T_M equal to the noise SD, calculated as the SD of the raw data, again on the assumption of

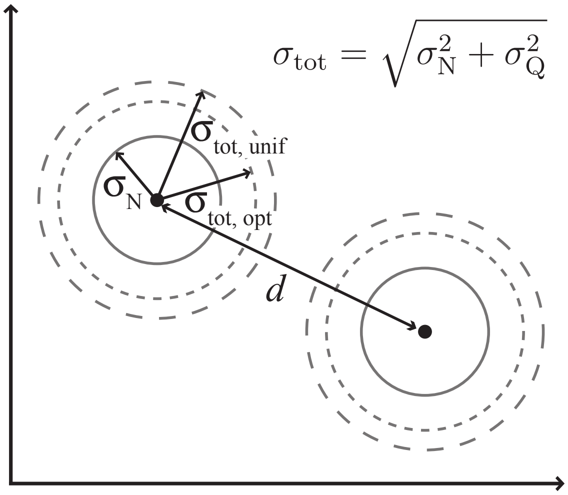

$\sigma_{\text{tot}} = \sqrt{\sigma_{\text{N}}^2 + \sigma_{\text{Q}}^2}$
σ tot, unif
σ N
σ tot, opt
d

Figure 5.2: Plot of received signals in 2-dimensional space. Each axis represents one dimension of the signal (arbitrary units). Each observed signal is the sum of the mean spike shape and a random noise variable. As a result, received points are spread around each mean according to the noise probability distribution. The two noise sources considered here are the system noise before quantization (SD = σ_{N}), which is composed of biological and electronics noise, and quantization noise (SD = σ_{Q}). We assume that each noise source is independent, so their variances add linearly, and the total noise SD is $\sigma_{\text{tot}} = \sqrt{\sigma_{\text{N}}^2 + \sigma_{\text{Q}}^2}$.

signal sparsity. However, in simulations we obtained better results when using the estimator given in Eq. 5.2.

5.3 Quantization

In Chapter 3, the theoretical performance of various spike-sorting algorithms was studied using floating-point precision. In practice, however, an N_{B}-bit ADC will limit the analog neural input signal to a finite resolution of $2^{N_{\text{B}}}$. The signal at the output of the ADC, V_{tot}, can be written as

$$V_{\text{tot}} = V_{\text{IN}} + V_{\text{N}} + V_{\text{Q}}, \tag{5.3}$$

where V_{IN} is the neural input signal, V_{N} is the pre-quantization system noise (i.e., biological and thermal), and V_{Q} is the quantization noise. The system noise and quantization noise are assumed to be independent, making the total noise variance

$$\sigma_{\text{tot}}^2 = \sigma_{\text{N}}^2 + \sigma_{\text{Q}}^2. \tag{5.4}$$

Figure 5.2 shows the effect of quantization noise on the signal. When viewed in D-dimensional space, the pre-quantization noise sources will lead to a distribution of points around each mean with a spread of SD $= \sigma_{\text{N}}$. Quantization effects will increase the spread to SD $= \sigma_{\text{tot}}$.

The most widely used ADC in practice is a uniform quantizer. The full-scale voltage, V_{fs}, is quantized to bins of a fixed width, $V_{\text{LSB}} = V_{\text{fs}}/2^{N_{\text{B}}}$. It is typically estimated that the probability of receiving the voltages within each quantized bin is approximately constant for moderate- to high-resolution quantizers. This results in a uniform distribution of the quantization noise from $-V_{\text{LSB}}/2$ to $V_{\text{LSB}}/2$ with $p(V_{\text{Q}}) = \frac{1}{V_{\text{LSB}}}$. The variance of V_{Q} is calculated by

$$\sigma_{\text{Q}}^2 = \int\limits_{-V_{\text{LSB}}/2}^{V_{\text{LSB}}/2} V_{\text{Q}}^2 \, p(V_{\text{Q}}) \, dV_{\text{Q}} = \frac{{V_{\text{LSB}}}^2}{12} \tag{5.5}$$

and represents the quantization noise power. Uniform ADCs are well suited for applications where (i) the probability distribution function (pdf) of the input is uniform or (ii) the pdf of the input is not available to the designer.

In systems where the input voltage pdf *is* available to the designer (and is non-uniform), we can obtain lower quantization noise variance for the same resolution by using an optimal quantizer [115]. For a given input pdf, an optimal quantizer optimizes x_i, the transition voltage, and y_i, the representative voltage of the i^{th} bin, $\forall\, i \in \{2^{N_{\text{B}}}\}$. The solution to the resultant optimization problem shows that the optimal signal-to-quantization-noise ratio, SQNR, is obtained when:

1. $x_i = (y_i + y_{i+1})/2$, and
2. y_i is the centroid of the bin bounded by x_i and x_{i+1}.

The difference in quantization noise can best be understood by looking at the quantization

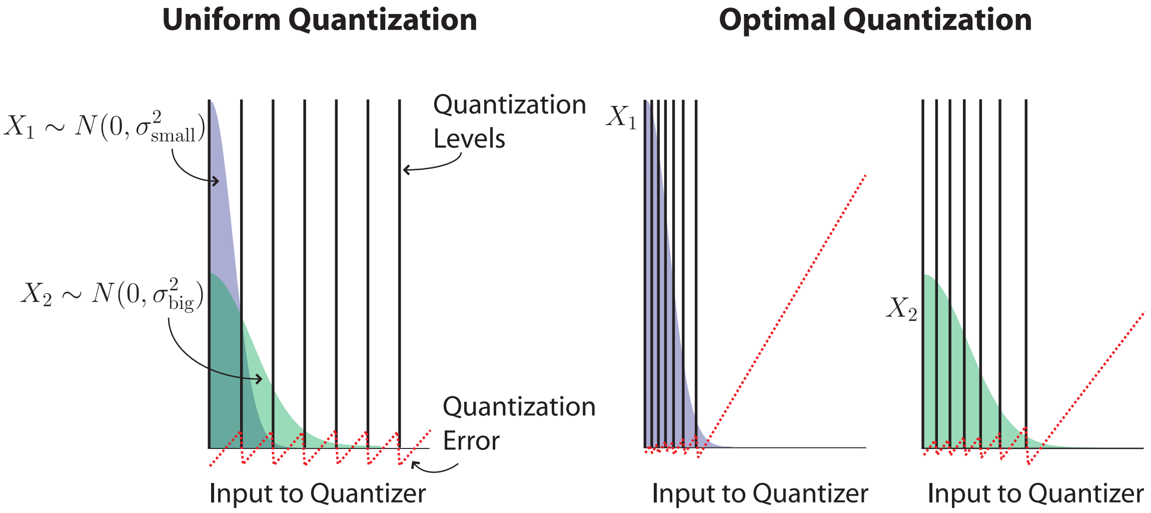


Figure 5.3: Examples of uniform and optimal quantization levels for narrow and wide Gaussian pdfs. The optimal quantizer minimizes the quantization error for each Gaussian by placing more bins in higher-probability regions and fewer bins in lower-probability regions.

noise curves of Fig. 5.3. In uniform quantization, each bin supplies the same amount of noise regardless of the input pdf. An optimal quantizer, on the other hand, selectively chooses the transition levels such that inputs with high probabilities will have a lower quantization noise than inputs with low probabilities. Figure 5.4 (left) shows the maximum SQNR for Gaussian and Laplacian pdfs quantized using both a uniform and an optimal quantizer. Depending on the resolution and pdf, the optimal quantizer can increase the SQNR by up to 10 dB. Using the approximation that each resolution bit increases the SQNR by about 6 dB, this implies that for a given SQNR, an optimal quantizer will use 1–2 fewer resolution bits than a uniform quantizer.

In spike-sorting applications, we have the benefit of a priori knowledge of the input pdf. Neural signals have a generalized normal distribution with a scale factor between 1 (Laplace) and 2 (Gaussian), as shown in Fig. 5.4 (right), and their distribution does not change rapidly over time. To evaluate the benefits of optimal quantization over uniform quantization, we will evaluate the performance of the detection and clustering algorithms using both types of quantization. But first, we will estimate the required resolution for uniform quantizers.

5.3.1 Detection

The basic idea in detection is to differentiate signal values above and below σ_{N}. To do this, the first quantization level must be less than or equal to σ_{N}. It follows that we can approximate a lower limit on the number of bits required for detection using uniform quantization to be

$$N_{\mathrm{min}} = \left\lceil \log_2 \left(\frac{V_{\mathrm{fs}}}{\sigma_{\mathrm{N}}} \right) \right\rceil . \tag{5.6}$$

This equation gives us the number of bits required to reach a noise level of σ_{N}, or the number of bits required to ensure the maximum performance for a known σ_{N}. Note that [113] and [114] are essentially using the same equation when they set the DNR of the ADC equal to the "DNR" of the input signal.

Because the accuracy of spike detection is highly dependent on the accuracy of the estimate of the noise SD (see Eq. 5.1 and Eq. 5.2), optimal quantization should improve the accuracy of spike detection. The optimal quantizer chooses the quantization levels and representative values to minimize the quantization error for a given distribution (Fig. 5.3). Minimizing the quantization error for V_{N} should allow us to estimate σ_{N} more accurately, thereby increasing the accuracy of detection.

5.3.2 Clustering

The separability of two clusters depends on the distance between their mean spike shapes d (Fig. 5.2). Quantization will increase the SD of each cluster to σ_{tot}, which increases the probability that the clusters overlap. Thus, in order to ensure that we will be able to separate highly similar spikes, we must minimize σ_{Q}.

Equations 5.4 and 5.5 show that as N_{B} increases, σ_{Q}^2 decreases, and as σ_{Q}^2 decreases, σ_{tot}^2 approaches σ_{N}^2. If we let σ_{Q}^2 equal σ_{N}^2, then $\sigma_{\mathrm{tot}} = \sqrt{\sigma_{\mathrm{N}}^2 + \sigma_{\mathrm{Q}}^2} = \sqrt{2\sigma_{\mathrm{N}}^2} = 1.4\sigma_{\mathrm{N}}$, corresponding to a degradation in SNR of 3 dB. Whether or not this SNR loss is tolerable is a function of the data, that is, of how small the distances are likely to be between pairs of mean spikes. Ideally one should determine how much SNR loss is tolerable for a particular

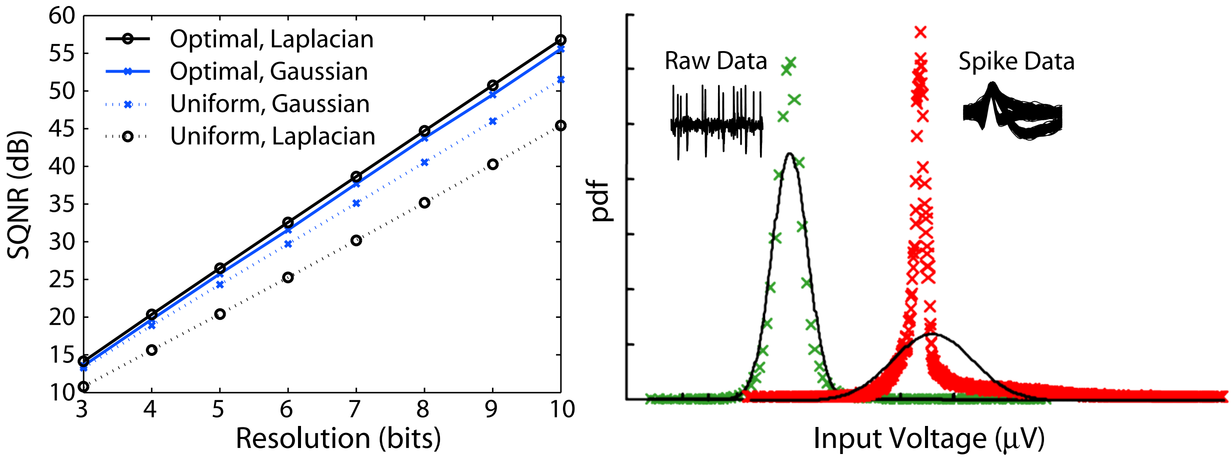


Figure 5.4: *Left*: SQNR vs. resolution for a uniform quantizer compared to the optimal quantizer. The optimal quantizer can increase the SQNR by up to 10 dB. *Right*: Input pdfs for spikes (red x's) and raw data (spikes plus noise, green x's). The black line shows a Gaussian fit using the sample mean and variance. Both non-uniform distributions suggest that non-uniform quantization would be more appropriate than uniform quantization.

dataset and then calculate how many bits are needed.

An optimal quantizer may also improve the accuracy of clustering for a given resolution. For clusters that are separable prior to quantization but that have a small d, it is critical to minimize σ_{Q} such that $\sigma_{\mathrm{tot}} \approx \sigma_{\mathrm{N}}$ so that the clusters are still separable after quantization. For a given resolution, an optimal quantizer can reduce the quantization noise compared to a uniform quantizer by up to 10 dB (Fig. 5.4, left), effectively reducing the noise around the cluster centroid. Thus, it may be able to maintain the separation between two clusters when a uniform quantizer cannot.

5.4 Simulation Results

We tested the effects of quantization on the synthetic datasets described in [64], which include a variety of difficulty levels (resulting from the degrees of similarities between mean spike waveforms) and SNRs. These datasets provided us with a ground truth to be utilized in the accuracy calculations. Data was generated at 24 kHz with floating-point precision. The maximum amplitudes of the mean spike waveforms in each dataset were normalized to 1

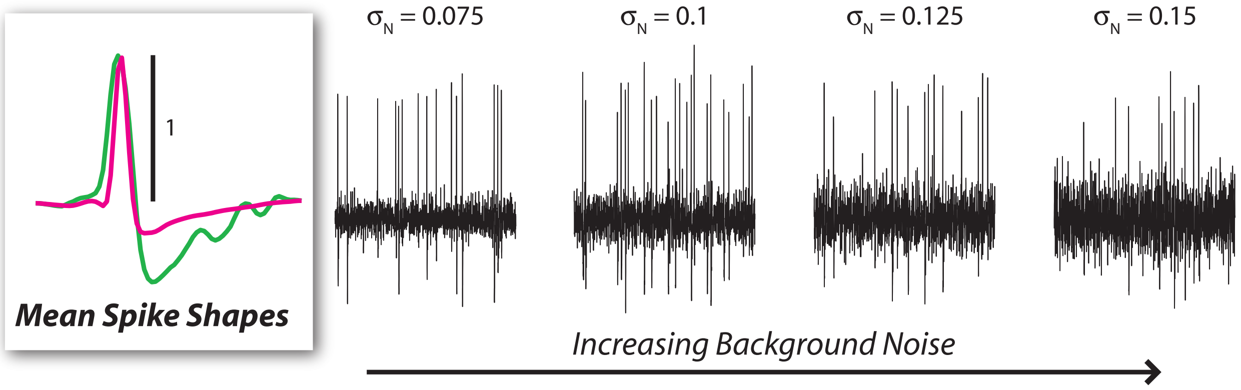


Figure 5.5: Example of a synthetic dataset composed of spikes from two neurons whose mean spike shapes are shown on the left. Data is generated with varying degrees of background noise. Note that only a few milliseconds of the 60-s datasets are shown here.

(unitless). An example of one dataset is shown in Fig. 5.5. The effects of quantization were examined by quantizing this raw data prior to spike sorting and calculating the accuracy after spike sorting. The quantizers were assumed to have $V_{\text{fs}} = 3.33$ (unitless) to allow for biphasic spikes and to accurately model worst-case conditions for practical front ends with adaptive gain control.

To calculate the accuracy of detection, we defined N as the total number of true spikes in the dataset and D as the number of correctly detected spikes. We also defined FA as the number of false alarms and TN as the number of true negatives (total number of samples not containing a spike, normalized to the length of a spike). The quantities D/N and FA/TN, then, represent the probability of detection and the probability of false alarm, respectively. We then calculated the clustering accuracy separately by using the known true spikes, rather than the detected spikes, as input to the clusterer; this allowed us to determine whether detection or clustering was the limiting factor in terms of accuracy/required resolution. We defined C as the number of spikes that were classified correctly; the quantity C/N, then, represents the probability of correct classification.

Figure 5.6 shows the results of detection and clustering for the dataset shown in Fig. 5.5. Note that $D/N = 0$ indicates cases where, because most of the values were quantized to 0, Eq. 5.2 evaluated to 0, and detection was terminated in order to avoid the situation where all samples trigger detection. Also note that for most simulation points (noise levels and resolutions), the probability of false alarm was negligible ($< 1\%$ in most cases, $< 5\%$ in every case); therefore, we did not include this figure in the plots.

First, let us look at the performance of the uniform quantizer. As expected, for both detection and clustering we achieve near-perfect performance at 10 bits, and the performance degrades as the resolution decreases, until we reach the point where the threshold can no longer be calculated and the accuracy drops to zero. Note the points in the clustering plots where the classification accuracy is 50%; this corresponds to chance performance, since we have two neurons in this dataset. In other words, at these points the threshold can still be calculated, but classification is random. Also notice that the accuracy is always limited by (or the required resolution is always dictated by) the clustering block.

Interestingly, as the noise in the system increases, the uniform quantizer requires a lower number of bits for accurate detection. Although this result may seem unintuitive, remember that the detection method requires an accurate estimate of the noise SD. As shown in Fig. 5.3, for a fixed resolution or bin width, noise with a large σ_{N} will quantize into multiple bins, resulting in an accurate estimate of the noise, while noise with a small σ_{N} will be quantized into the same bin, making estimation of σ_{N} impossible.

Next, let us compare our simulation results to the calculated values of N_{min} for the case of uniform quantization. Equation 5.6 predicts that, with known σ_{N}'s, we can guarantee maximum detection performance with 6, 6, 5, and 5 bits for $\sigma_{\mathrm{N}} = 0.075$, 0.1, 0.125, and 0.15, respectively; Fig. 5.6 shows that, for this example, we actually need 6, not 5, bits for perfect detection for $\sigma_{\mathrm{N}} = 0.125$ and $\sigma_{\mathrm{N}} = 0.15$. This indicates that Eq. 5.6 is not reliable for determining the required resolution of a system. (Remember, this very equation was used to determine the system resolution in [113] and [114].)

We can also use this plot to determine the minimum number of bits required to cluster at each noise level, and then back-calculate the maximum allowable degradation in SNR due to quantization. In this example, assuming that a resolution of 6 bits achieves acceptable classification accuracies in each case, we must limit the degradation in SNR to about 1.6 dB

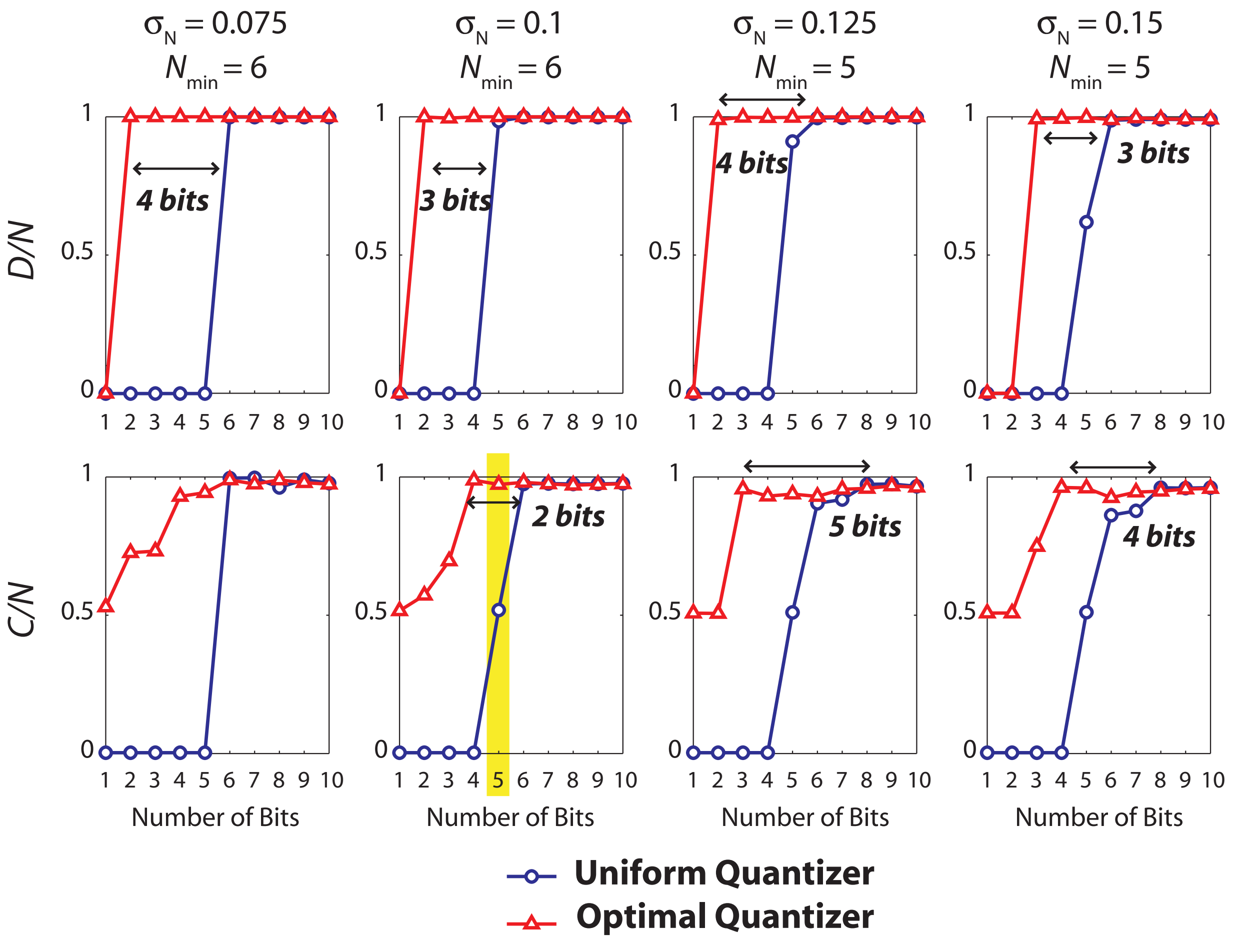


Figure 5.6: Comparison of detection (*top*) and classification (*bottom*) results for the example dataset shown in Fig. 5.5 when a uniform vs. an optimal quantizer is used.

($\sigma_{\text{tot}} = 1.2\sigma_{\text{N}}$), 1.2 dB ($\sigma_{\text{tot}} = 1.15\sigma_{\text{N}}$), 1.0 dB ($\sigma_{\text{tot}} = 1.12\sigma_{\text{N}}$), and 0.8 dB ($\sigma_{\text{tot}} = 1.1\sigma_{\text{N}}$), for $\sigma_{\text{N}} = 0.075$, 0.1, 0.125, and 0.15, respectively, in order to ensure good clustering results.

Next, let us compare the results of detection for the uniform quantizer and the optimal quantizer. Figure 5.6 shows that, across the board, the optimal quantizer requires 3 to 4 fewer bits for detection. This is likely because it minimizes the quantization error for V_{N}, thereby allowing us to estimate σ_{N} more accurately. Likewise, clustering requires as many as 5 fewer bits when the optimal quantizer is used versus when the uniform quantizer is used. In such cases, an N_{B}-bit optimal quantizer would be able to sort just as well as an $(N_{\text{B}} + 5)$-bit uniform quantizer.

An example of how using an optimal quantizer can benefit clustering is shown in Fig. 5.7.

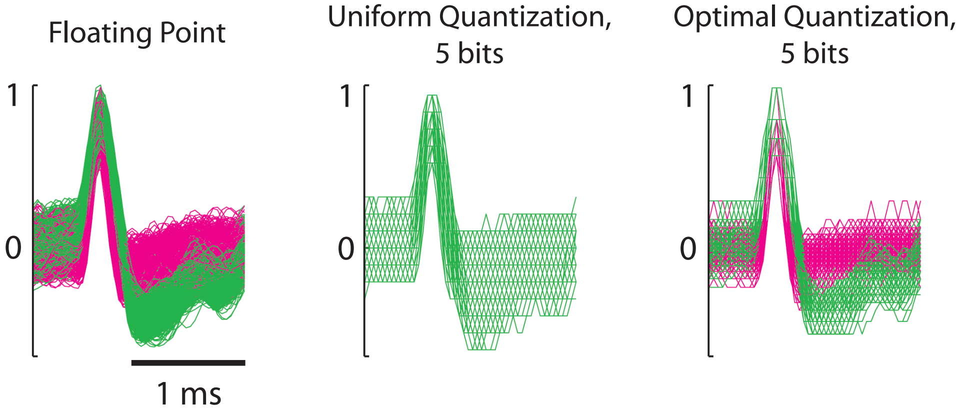


Figure 5.7: An example of the benefits to clustering of using an optimal quantizer, corresponding to the example for $\sigma_{\mathrm{N}} = 0.1$, $N_{\mathrm{B}} = 5$ (highlighted in Fig. 5.6).

This example corresponds to the example in Fig. 5.6 for $\sigma_{\mathrm{N}} = 0.1$, $N_{\mathrm{B}} = 5$. The clustering results when floating-point precision is used is shown on the left. The clustering results are good for both the uniform and the optimal quantizer from 10 down to 6 bits. At 5 bits, however, clustering of the uniformly quantized data breaks down by clustering all spikes into the same cluster (Fig. 5.7, center), whereas the optimally quantized data can still be clustered (Fig. 5.7, right). These results indicate that the quantization error introduced by uniform quantization increased σ_{tot} to a point where the clusters overlapped, whereas the quantization error introduced by optimal quantization was much smaller ($\sigma_{\mathrm{tot}} \approx \sigma_{\mathrm{N}}$) and the clusters were still well separated.

5.5 Conclusion

We have analyzed the effects of quantization on the performance of spike sorting. First, we provided intuitive explanations for how various quantization techniques affect various stages of spike sorting. We derived, for the uniform quantizer, an estimate for the number of bits needed to ensure the maximum performance for spike detection and a theoretical framework for calculating number of bits needed for clustering. We also provided evidence that optimal quantizers are appropriate for neural data. Finally, we demonstrated that optimal quantization provides a savings of 1–2 bits in theory and up to 5 bits in simulations.

Because optimal quantizers can improve the accuracy of spike sorting for a given resolution, or provide the same accuracy with fewer bits, they are well suited for ADCs in wireless neural recording systems.

Acknowledgments

The authors thank Mr. M. Rachid for helpful conversations about quantization noise. This work was supported in part by the National Science Foundation under Grant Numbers 0847088 and 0824275.

Permissions

Substantial portions of this chapter have been reprinted, with permission, from: S. Gibson, V. Wang, and D. Marković, "Effects of Quantization on Neural Spike Sorting," in *Proc. 2011 IEEE Int. Symp. Circuits Syst.*, Rio de Janeiro, Brazil, 2011, pp. 2099–2102. Copyright © 2011, IEEE. Note that the results section contains updates since the original publication.

Author contributions: V. Wang proposed the idea of using nonlinear quantization for spike sorting and provided the code for the quantizers. D. Marković was the PI.

CHAPTER 6

Spike-Sorting ASICs

6.1 Spike-Sorting Chip

Building on the algorithm evaluation work described in Chapter 3, we have developed two digital chips that perform automatic spike sorting. The first chip [116, 117] performs NEO detection, maximum-derivative alignment, and discrete-derivatives feature extraction, simultaneously for 64 channels. The chip has a modular architecture, which allows it to be configured to process 16, 32, 48, or 64 channels. The chip was implemented in a 90-nm CMOS process and has a power dissipation of 130 μW (power density of 30 μW/mm^2) when processing all 64 channels. Processing with this chip would reduce the data rate by 91.25% (11.71 to 1.02 Mbps). This data reduction could increase battery life from 1.6 to 18 hours (11x)[1].

A micrograph of the chip and a summary of its characteristics are shown in Fig. 6.1 and Table 6.1. An example showing the output of the chip is shown in Fig. 6.2. Data from two neurons are present in this example. We performed offline clustering on both the aligned spikes (bottom–left) and on the extracted features (bottom–right). The figure shows that the two spike classes are more separable in the feature domain than in the time domain. Indeed, for this example we can achieve a clustering accuracy of 92% for extracted features, compared to 77% for the original waveforms.

Table 6.2 shows a comparison of our work with previous spike-sorting DSPs [106–108, 118]. Our design performs simultaneous processing of 64 channels with a power consumption/power density lower than previous designs that perform only detection for multiple

[1] Assuming the transmitter in [20] with 20.45 nJ/b at 13 cm, an Energizer CR1632 battery with a capacity of 130 mAh at 3 V = 390 mWh, and an analog front-end power of 10 μW per channel [70].

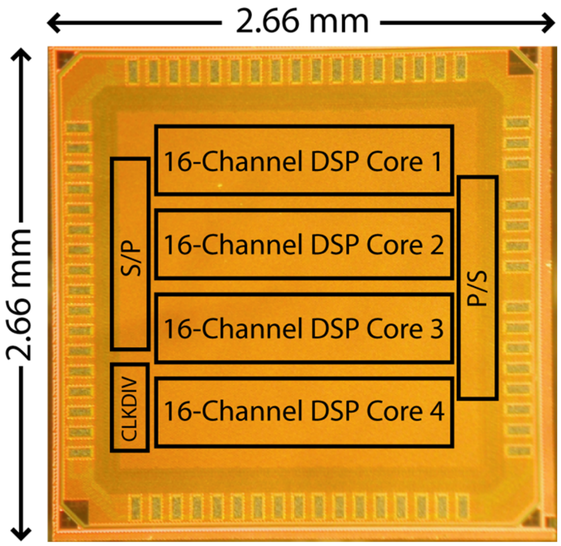


Figure 6.1: Micrograph of spike-sorting chip.

Table 6.1: Spike-Sorting Chip Summary

Technology	90-nm 1P8M CMOS
Core V_{DD}	0.55 V
I/O Voltage	1.2 V / 2.5 V
Gate Count	650 k
Clock Domains	0,4 MHz, 1.6 MHz
Power	2 μW/channel
Data Reduction	91.25% (11.71 to 1.02 Mbps)
No. of Channels	16, 32, 48, or 64
Median P_D	87%
Median P_{FA}	5%
Median CA	77%

channels or detection and feature extraction for a single channel. The design shows a 7-times reduction in power consumption and 2-times reduction in power density compared to the previous state-of-the-art spike-sorting DSPs.

6.2 Osort Chip

Previous spike-sorting DSP chips [106,107,117] have implemented spike detection and feature extraction, but none perform online multi-channel clustering. This is primarily because most of the clustering algorithms traditionally used for spike sorting are offline algorithms that

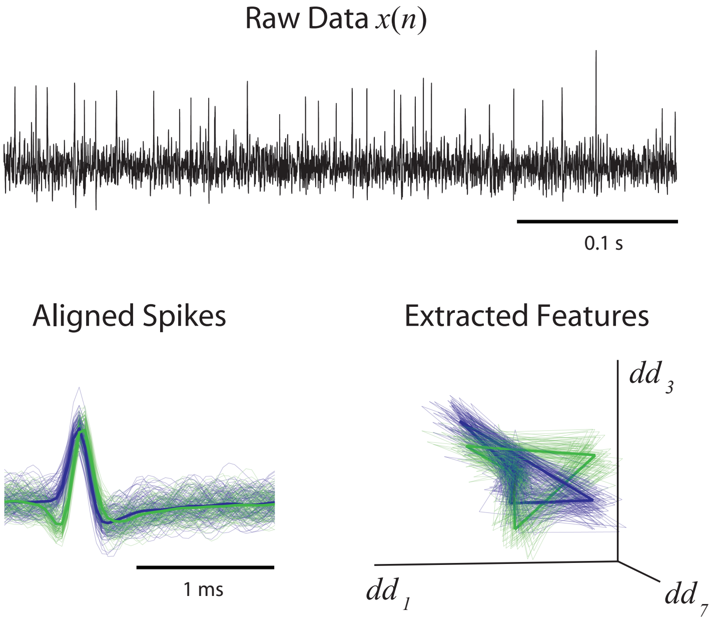


Figure 6.2: Sample of chip output. For the synthetic data shown on top, spikes were detected and aligned (*bottom–left*), and each spike was expressed by three disrete derivative "coefficients" (*bottom–right*). Spikes have been colored according to the ground truth. (Figure adapted from [116].)

Table 6.2: Comparison with Prior Work

Reference	[108]	[107]	[106]	[119]	**This work**
No. of Channels	96	32	1	128	**64**
Power (μW/channel)	104	75	100	14.6	**2.03**
Area (mm^2/channel)	–	0.11	1.58	0.01	**0.06**
Power Density (μW/mm^2)	–	680	60	1460	**30**
Process (nm)	FPGA	500	350	90	**90**
Core Voltage (V)	–	3	3.3	1.08	**0.55**
Detection	✓	✓	✓	✓	✓
Alignment	×	×	×	×	✓
Feature Extraction	×	×	✓	✓	✓

cannot be used for real-time data streams (see Section 2.4.5). Without clustering, however, on-chip spike-sorting cannot be considered complete. In our second chip [70], we implemented absolute-value spike detection along with multi-channel, unsupervised clustering, allowing us to meet the real-time spike-sorting requirement for applications like brain–machine interfaces. Additionally, because on-chip clustering reduces the output data rate by 20 times, we were able to support a higher number of channels for a given system power consumption.

6.2.1 Clustering Algorithm Evaluations

As mentioned in Section 2.12, the only automatic, online clustering algorithm (and, therefore, the best candidate for hardware implementation) known to the authors at this time is called Osort [69]. To benchmark its performance, we first evaluated the accuracy of Osort and compared it to other established clustering methods: k-Means, SPC, and Valley-Seeking (refer to Section 2.12 for descriptions of each method). The algorithm evaluation process was similar to the one described in Chapter 3. We used the same simulated datasets as described in Chapter 3, but we only used those with positive SNRs, as we found that many of the clustering methods broke down in negative SNR. Thus, we used 618 datasets with SNRs from about 20 dB down to 0 dB.

In testing all of the algorithms, we first performed perfect spike detection by extracting spikes using the file of known spike times. (Thus the detection was 100% accurate.) We then aligned spikes to their maximum values. In order to test k-means, SPC, and Valley-Seeking, we first performed feature extraction using PCA. Because Osort uses raw spikes as inputs instead of features, no feature extraction was performed prior to Osort clustering. Note that while SPC, Valley-Seeking, and Osort are all unsupervised algorithms, k-means requires the user to input k (the number of clusters/neurons) for each dataset; the correct value of k was always given in this analysis.

The results of this algorithm comparison are shown in Fig. 6.3. K-means has the highest accuracy, but it has the advantage of a priori knowledge of k, whereas all the other algorithms have to compute the number of clusters. SPC and Valley-Seeking both have higher accuracies

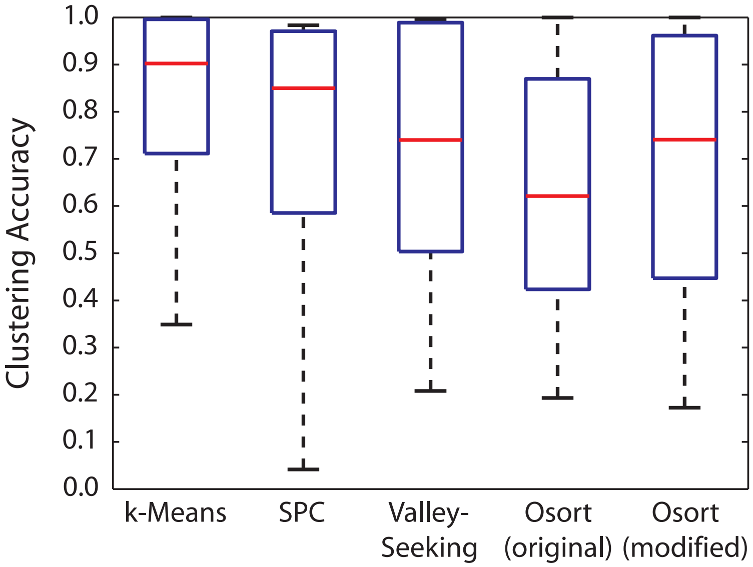


Figure 6.3: Box plots showing accuracy for each clustering algorithm tested. Red lines in the centers of the boxes represent the median accuracy, box edges the lower and upper quartiles, and whisker edges the maximum and minimum values ($N = 618$).

than Osort ("Osort (original)"), but they have the advantage of having information about the entire dataset before clustering is initiated, while Osort is causal (i.e., it makes decisions about each spike using only information about past spikes, not future spikes). In [70], we introduced several modifications to the Osort algorithm in order to reduce its complexity which actually turned out to improve its performance (Fig. 6.3, "Osort (modified)").

6.2.2 Chip Results

The die micrograph of the 16-channel spike-sorting chip is shown in Fig. 6.5. The chip occupies a core area of 1.23 mm^2 in a 65-nm CMOS process and consumes 75 μW of power from a 270-mV supply voltage. The power density of the chip is 67 μW/mm^2, which is 12-times lower than the power density known to damage brain cells. The algorithmic performance of the chip is characterized by a probability of detection (P_{D}) of 95%, a probability of false alarm (P_{FA}) of 1%, and a median classification accuracy (CA) of 75%. Table 6.3 summarizes the performance of the chip.

The chip was tested using human neural data recorded from one of nine 40-μm-diameter

Table 6.3: Osort Chip Summary

Technology	65-nm 1P9M CMOS
Core V_{DD}	0.27 V
I/O Voltage	1.2 V / 2.5 V
Clock Freq.	400 kHz
Power	4.68 μW/channel
Data Reduction	240x
No. of channels	16
Median P_D	95%
Median P_{FA}	1%
Median CA	75%

electrodes positioned in the hippocampal formation of a human epilepsy patient at UCLA. Figure 6.4(a) figure shows a sample raw data waveform on one of the channels along with the spike-detection thresholds. In this particular example, the detected action potentials were classified into three different clusters with the cluster means shown in Fig. 6.4(b). Figure 6.4(c) shows a raster plot of the detected action potentials. A two-dimensional projection of the 48-dimensional spike waveforms is shown in Fig. 6.4(d) to aid visualization of the clustering results. The approximate cluster boundaries are marked with the dotted lines and the cluster means are marked with bold colored markers.

Table 6.4 compares this work to previous spike-sorting DSP chips. As mentioned earlier, this is the only spike-sorting chip that provides real-time, multi-channel clustering. This chip can reduce the data rate from 3.072 Mbps to 12.8 kbps (240x), a 20-times higher reduction than previous work with only 2-times higher power consumption. While the DSP power is higher than previous work, the high data-rate reduction achieved due to online clustering would allow us to reduce the radio power, thereby reducing the total system power consumption. This could increase battery life from 6.2 hours to over 1 month [2].

[2] Assuming the transmitter in [20] with 20.45 nJ/b at 13 cm, an Energizer CR1632 battery with a capacity of 130 mAh at 3 V = 390 mWh, and an analog front-end power of 10 μW per channel [70].

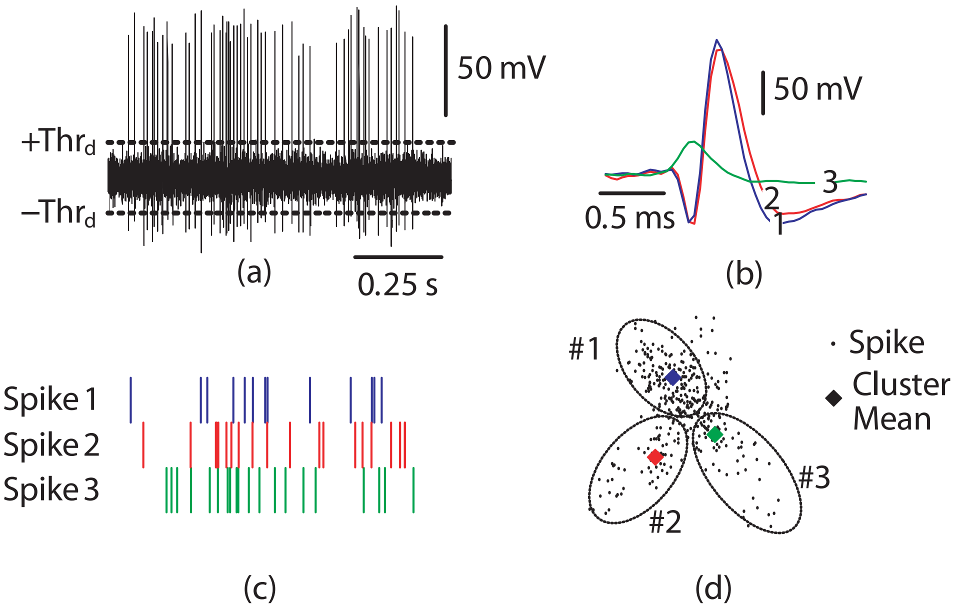


Figure 6.4: The chip was used to process multi-channel human neural data. The sample output from one of the channels is shown in this figure. (a) Sample raw data waveform. (b) Identified cluster means. (c) Raster plot of the detected action potentials. (d) A 2-D projection of the 48-dimensional cluster space presented to aid in visualization of the clustering results.

Table 6.4: Comparison with Prior Work

Reference	[107]	[106]	[116]	**This work**
No. of Channels	32	128	64	**16**
Detection	✓	✓	✓	✓
Feature extraction	×	✓	✓	**N/A**
Clustering	×	×	×	✓
Data-rate reduction	12.5x	80x	11x	**234x**
Power (μW/channel)	75	100	2.03	**4.68**
Area (mm^2/channel)	0.11	1.58	0.06	**0.07**
Power Density (μW/mm^2)	682	63.3	33.8	**66.8**
Process (nm)	500	350	90	**65**
Core Voltage (V)	3	3.3	0.55	**0.27**

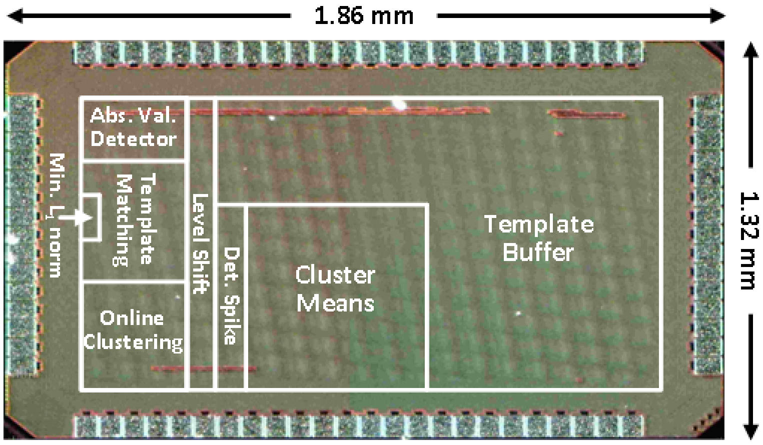


Figure 6.5: Die photo the 16-channel Osort chip.

Acknowledgments

We would like to thank Dr. Richard Staba at the UCLA Neurology Department for allowing access to human neural recordings for testing these chips. We would also like to thank the UCLA PDA group members for several helpful discussions. This work was supported in part by the National Science Foundation under Grant Numbers 0847088 and 0824275.

Permissions

Portions of Section 6.1 have been reprinted, with permission, from: V. Karkare, S. Gibson, and D. Marković, "A 130-μW, 64-Channel Neural Spike-Sorting DSP Chip," *IEEE J. Solid-State Circuits*, vol. 46, no. 5, pp. 1214–1222, May 2011. Copyright © 2011, IEEE.

Author contributions: V. Karkare performed all chip design. D. Marković was the PI.

Portions of Section 6.2 have been reprinted, with permission, from: V. Karkare, S. Gibson, C.-H. Yang, H. Chen, D. Markovic, "A 75μW, 16-Channel Neural Spike-Sorting Processor with Unsupervised Clustering," in *Proc. Int. Symp. on VLSI Circuits*, Kyoto, Japan, 2011, pp. 252–253. Copyright © 2011, IEEE.

Author contributions: V. Karkare performed all chip design, as well as the algorithm modifications. C.-H. Yang contributed to the memory-bank design. H. Chen helped with chip testing. D. Marković was the PI.

CHAPTER 7

FPGA-Based Spike-Sorting Platform

7.1 Introduction

7.1.1 Motivation

The work presented in Chapters 3–6 address the need to reduce the output data rate of neural recording systems in order to make wireless recording feasible. A separate issue with neural recording discussed in Chapter 1 was the need to reduce the offline processing time to spike-sort data that has already been recorded onto hard disks.

To review, currently most neural recording systems record raw data onto computer hard disks. This data is usually acquired with sampling rates of 20–30 kHz and resolutions of 12–24 bits per sample [22–27] for 64–128 channels simultaneously. For a human epilepsy study in which 64 channels of data are sampled at 27.777 kHz and quantized to 16 bits, an 8-hour experiment would accumulate about 100 GB of data. All data processing, including spike sorting, would be performed in software after the experiment. Conventional software tools[1] would require about 30 hours to sort the data from this one day of experiments (processing rate of 0.94 MBps).

If we could replace spike-sorting software with dedicated hardware running at 100 MHz (200 MBps), the processing time would be reduced from 30 hours to 8.5 minutes (212x). This example illustrates the clear need for the hardware acceleration of spike sorting.

[1] Osort software package [28] running in Windows on an Intel Core2 Duo Processor.

7.1.2 Proposed Solution

In this chapter, we present a hardware spike-sorting tool for accelerating the offline processing of existing neural data. In order to provide a good compromise between speed and flexibility, we chose to design our tool around an FPGA.

FPGAs, or field-programmable gate arrays, are a type of hardware that can be configured using software. FPGAs represent a tradeoff between speed and flexibility. At the high-speed/low-flexibility end of the spectrum we have ASICs (application-specific integrated circuits). ASICs have fixed architectures: each chip is designed and manufactured to perform only a particular function or set of functions. This allows the chip to be optimized in terms of power and area efficiency, but at the expense of flexibility. On the other end of the spectrum, high-flexibility/low-speed, we have general-purpose hardware such as microprocessors. Microprocessors are highly flexible—they can be programmed to perform virtually any operation—but this flexibility comes with a high overhead: Microprocessors also have fixed architectures that can support finite instruction sets, so performing even a simple operation takes many clock cycles (to fetch the instruction from program memory, decode the instruction, execute the instruction, write the results to memory). Additionally, microprocessors can only perform 3 to 4 operations in parallel, compared to ASICs which typically perform 10's to 100's of operations per clock cycle [120].

FPGAs fall somewhere in between these two extremes. Unlike both ASICs and microprocessors, they do not have a fixed architecture. They are manufactured with standard hardware units called “logic blocks”, which include basic units like lookup tables, full adders, and flip flops (memory), and a grid of reconfigurable interconnects, which allow blocks to be wired together in different ways to realize different architectures. The user can then program the FPGA using software that tells the FPGA how to configure the interconnects in order to achieve the desired architecture. In this way, the hardware can be configured to perform complex combinational logic functions. Like ASICs, FPGAs can also perform a high number of operations in parallel, which increases the efficiency, or effectively the speed, of computation.

FPGAs are a good choice of hardware for a spike-sorting platform because we have the flexibility to program various spike-sorting algorithms onto it during runtime, such that the user does not have to sacrifice flexibility of existing software, but because the hardware architecture is customized for spike sorting, it will run much faster than software can run on a general-purpose CPU. This flexibility also leaves room for modifying or improving the design at any point in the future.

7.1.3 Design Features

Flexibility is one of our primary concerns; users are unlikely to use this tool if they perceive that they are sacrificing on the functionality currently offered by their software tools. Thus, we chose to develop a library of algorithms for each step in the spike-sorting process that the user can choose from during runtime. Another way that flexibility was incorporated was in the spike-detection threshold. Detection thresholds are automatically calculated in hardware according to Eq. 2.2 and Eq. 2.4. However, we also allow the user the option of adjusting that threshold manually if they so choose. We also tried to maximize the compatibility of our tool with existing data acquisition hardware by providing support for data sampled at any sampling rate between 5 and 125 kHz.

Another important design consideration was to make the tool as user-friendly as possible. We do not want the user to perceive any difference between our tool and the software that they are accustomed to using. Thus, we designed a user-friendly GUI which can be run locally on any PC and serves to abstract the hardware away from the user. The GUI also provides visual feedback to the user by displaying the processing status along with portions of the sorting results.

7.2 Functionality of Current Prototype

7.2.1 Inputs

The input data is required to be in binary format (.bin), quantized to 16 bits with 8 integer and 8 fractional bits ([16,8]). In other words, a range of -128 to 128 is supported. The data is assumed to have been bandpass-filtered, for example from 100 to 6000 Hz, to remove LFP. Sampling rates from 5 to 125 kHz are supported. Scripts are provided to convert data from MATLAB format (.mat) to binary format.

7.2.2 Processing

The spike sorter performs spike detection, spike alignment, and clustering. For spike detection, the user can choose from absolute value and NEO, which were shown in Chapter 3 to be accurate and low in complexity. For alignment, the user has four different options: alignment to the spike maximum, spike minimum, absolute value maximum, or NEO maximum. Clustering is performed using Osort, which was shown in Chapter 6 to be the clustering algorithm most feasible for hardware implementation. Note that with Osort feature extraction is not needed; thus, the current prototype of the tool does not perform feature extraction.

7.2.3 Outputs

For each spike, the spike sorter outputs the aligned waveform, the timestamp, and cluster ID. The timestamp is generated on-chip using a counter, such that the timestamp represents the sample number of the beginning of the spike relative to the first sample of the file. This value can be used to calculate the absolute spike times during post-processing. Outputs are first written to a binary file. MATLAB then reads in this file, converts it to .mat format, and displays the sorting results graphically.

7.2.4 Modes of Operation

This prototype includes two modes of operation, ‘Test’ and ‘Run’. In ‘Test’ mode, the processor runs for the length of the training time (to calculate the detection threshold), plus one additional second, and displays the results. This allows users to experiment with different detection and alignment parameters until they are satisfied with the results, before processing on the entire dataset. In ‘Run’ mode, the entire input data file is processed.

7.2.5 GUI

The user interface is a GUI which runs in MATLAB. The GUI takes the following information from users: input file name, input file sampling rate, detection method, detection threshold scale factor (C), the length of training time for detection, the alignment method, and the directory for the output file (results). There are two buttons for executing the two modes of operation, ‘Test’ and ‘Run’. When a button is pressed, the FPGA is programmed and configured according to the user inputs, and the data processing begins. While the data is being processed, a progress indicator is printed in a display at the bottom of the GUI. At the end of a run, the GUI will display the aligned spikes, color-coded according to the clustering results, as well as a raster plot, showing the spike train for each neuron (Fig. 7.1). A count of how many spikes were detected during the run is also displayed next to the raster plot.

7.3 Implementation

7.3.1 Overview

The FPGA spike-sorting tool was implemented in three major layers: the FPGA layer, the Python layer, and the MATLAB layer. The FPGA is where the data processing actually occurs. A Python script is used to program the FPGA as well as to control the flow of data to and from the FPGA. MATLAB is used to run the GUI, to call the Python scripts, and to display the results.

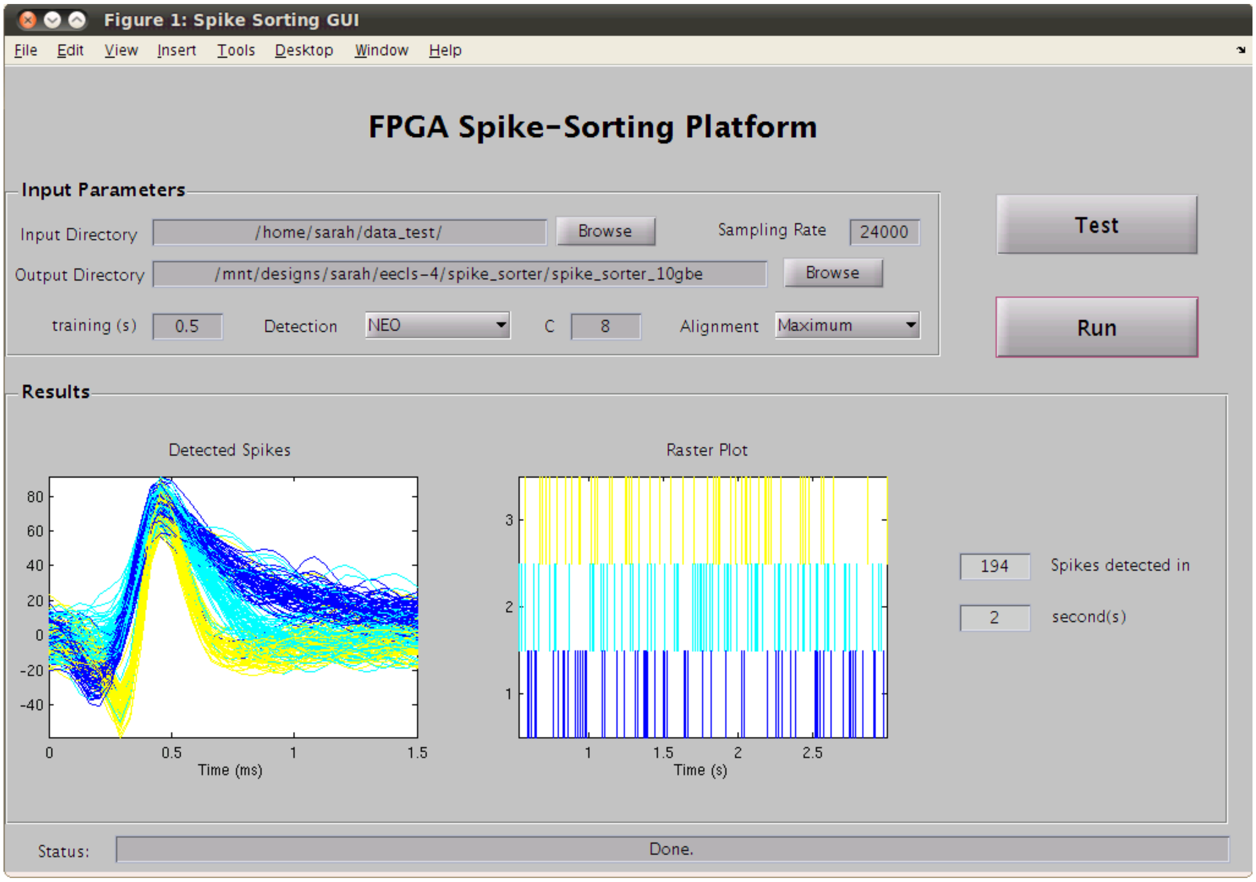


Figure 1: Spike Sorting GUI

FPGA Spike-Sorting Platform

Input Parameters

Input Directory /home/sarah/data_test/

Output Directory /mnt/designs/sarah/eecls-4/spike_sorter/spike_sorter_10gbe

Sampling Rate 24000

training (s) 0.5

Detection NEO

C 8

Alignment Maximum

Results

194 Spikes detected in

2 second(s)

Status: Done.

Figure 7.1: The user inputs sorting parameters into the 'Input Parameters' panel at the top of the window. Buttons for 'Test' and 'Run' are to the right. The 'Results' panel displays the aligned spikes color-coded according to the clustering results as well as a raster plot, showing the spike train for each neuron. A count of how many spikes were detected during the run is also displayed next to the raster plot. The progress is printed the 'Status' bar at the bottom.

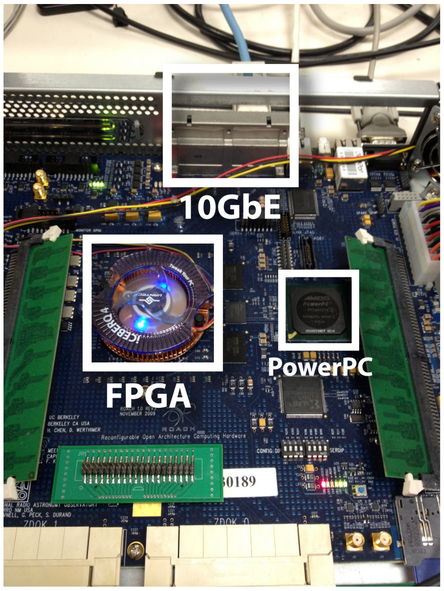


Figure 7.2: Photograph of the ROACH FPGA processing board [121].

In this section, we will first describe the hardware that was targeted in the design of this tool. We will then explain how the designs were implemented on the FPGA. Last, we will describe the software components of the tool, the Python script and the MATLAB script.

7.3.2 Targeted Hardware

We targeted the ROACH (Reconfigurable Open Architecture Computing Hardware) standalone FPGA processing board [121], which includes a Xilinx Vertex-5 FPGA and a separate PowerPC. The PowerPC runs an extended version of the Linux kernel called BORPH (Berkeley Operating system for ReProgrammable Hardware) [122], which can be used for interfacing between the FPGA and other external devices using Ethernet (Fig. 7.2). Instructions (e.g., program, write, read) can be sent from the local computer to the FPGA over Ethernet with the aid of KATCP (Karoo Array Telescope Control Protocol), a communications protocol designed specifically for the ROACH [123]. KATCP libraries have been developed both for MATLAB and for Python such that either environment can be used to communicate with the board via Ethernet. The board also includes a 10Gbps Ethernet (10GbE) interface. Note that the 10GbE data path bypasses the PowerPC and does not utilize KATCP routines.

7.3.3 FPGA Implementation

Algorithms were programmed in the Simulink environment using the Xilinx blockset, which allows designs to be mapped to Xilinx FPGAs. The BEE_XPS design tool [124] is a customized tool for mapping designs to the ROACH board. Given a Simulink model, this tool generates the bit file which can be loaded onto the PowerPC and then used to program the FPGA. The tool also provides a Simulink blockset with several memory elements, such as software registers and block RAMs, to/from which KATCP can directly write/read. The blockset also includes a 10GbE transceiver block for sending and receiving User Datagram Protocol (UDP) packets over the 10GbE interface.

The FPGA design is implemented in two major layers: the interface layer, which handles data I/O, and the spike-sorting layer, which processes the data.

7.3.3.1 Interface for Data I/O

The typical size of a file that may need to be processed by the tool at one time is on the order of GBs. Due to memory limitations of the FPGA, that entire amount of data cannot be written into FPGA memory at once. Thus, we designed a “streaming” processing scheme, where packets of data are continuously sent to the hardware and processed. To transfer data to and from the FPGA at the maximum possible speed, we used the ROACH’s 10GbE interface. Thus, the outermost layer of the FPGA design includes a 10GbE transceiver block that is used to receive packets of data from the computer and to send packets of results back to the computer (Fig. 7.3). We also included block RAM, both at the input and the output of the spike-sorting circuit, to buffer input and output data to ensure that the tranceiver’s internal buffer does not overflow. The effective processing scheme is as follows:

1. The computer sends a packet of data to the 10GbE transceiver.
2. The FPGA buffers data coming from the tranceiver into block RAM.
3. When the input buffer starts receiving valid data, a global “enable” signal is triggered which enables: (a) the sample-by-sample readout of data into the spike-sorting circuit,

and (b) the spike-sorting circuit itself[2].

4. As data is processed by the circuit, the results are written to the output buffers and simultaneously passed on to the 10GbE transceiver.

5. When the circuit has finished processing that packet, enable is set low and an "end of frame" signal is sent to the 10GbE tranceiver.

6. The tranceiver forms a packet and sends it back to the computer.

7. When the computer receives a packet, it sends a new packet to the FPGA.

This process is repeated until all the data has been processed.

The interface layer of the FPGA design also includes software registers for storing the desired spike-sorting parameters (i.e., the sampling rate, the detection algorithm, the detection threshold scale factor C, the length of training time, and the alignment algorithm) entered by the user via the GUI. These registers are written to over the Ethernet interface using KATCP. Note that we can afford to use this slower interface here because we only need to write to these registers once per input data file.

7.3.3.2 Spike Sorter

The spike sorter is composed of three main elements: the detection block, the alignment block, and the clustering block (Fig. 7.4). The detection block receives the raw data, as well as several of the configuration parameters (the detection method `det_meth`, the detection threshold scale factor `C`, and the signal indicating the training phase `training_phase_valid`). The outputs of detection are the detected spikes (`spk`) and a valid signal (`spk_valid`). When the alignment block detects a positive edge of `spk_valid`, it performs alignment on `spk` using the desired alignment method (`al_method`). When the clustering block detects a positive edge on `spk_al_valid`, it performs classification on `spk_al`. All blocks receive the global

[2] A global circuit enable is needed so that the circuit holds its state when it is not receiving valid data. For example, if the spike detection threshold calculation block were not disabled between blocks of valid data, it would process huge numbers of zeros as valid data, the effect of which would be to severely bias the threshold towards zero.

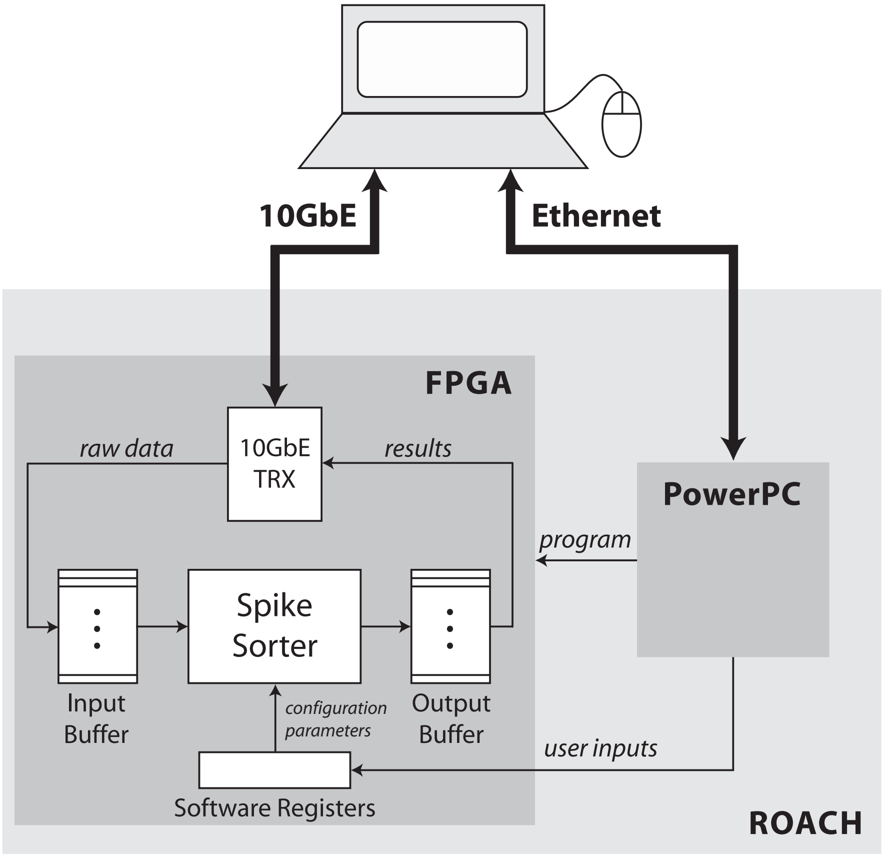


Figure 7.3: Data-streaming interface.

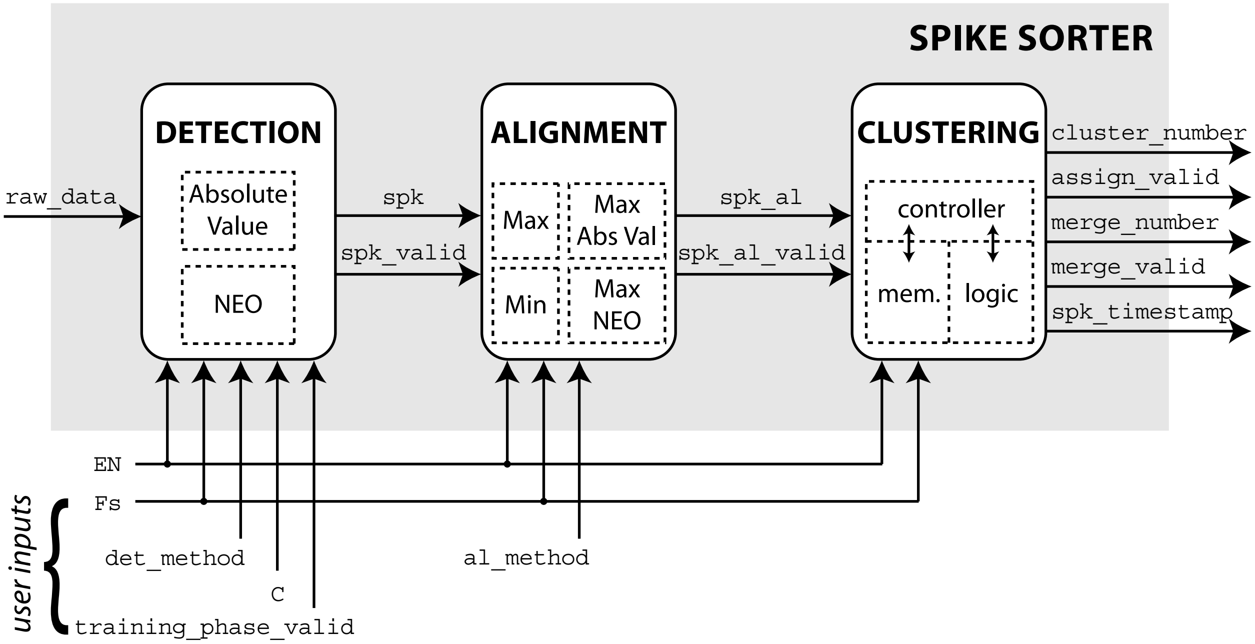


Figure 7.4: Block diagram of the processing elements on the FPGA.

enable `EN` and the sampling rate `Fs`. Following are some implementation details of each block.

Spike Detection

The detection algorithms were coded completely in Simulink using the above protocol. The detection block is composed of two sub-blocks, one for absolute-value and one for NEO, and a demux. To configure the hardware for the desired detection method, the code corresponding to the detection method chosen by the user (e.g. `det_meth` = 0 for absolute value, `det_meth` = 1 for NEO) is read from its software register and used as the select line to the demux, which passes the global enable signal through to the appropriate detection method block.

Both detection methods require a training phase in which the detection threshold is automatically calculated. The length of this training phase is also a user input, stored in a software register as an unsigned [12,6]-bit number (maximum value = 64 s, precision = 0.0156 s). This value is used to generate a `training_phase_valid` signal for the correct number of clock cycles, which is then used to enable the threshold calculation block during training while at the same time disabling all other circuitry. After the threshold has been calculated, it is multiplied by `C`, another user input stored in a software register as an unsigned [24,12]-bit number (maximum value = 4096, precision = 0.00024414).

The spike detection block also needs to know the sampling rate of the input data (F_s) because this determines how many samples to save as a spike (N_s). In spike detection, we buffer $N_s/3$ samples, and when a value crosses the detection threshold we declare all the $N_s/3$ samples in the buffer plus the following $2N_s/3$ samples as a spike. In order to implement this scheme for an arbitrary F_s, we created a variable-length delay. This was implemented using a dual-port block RAM (Fig. 7.5). A counter was used for addressing the RAM. Assume $c(n)$ represents the output of the counter at time n. Data is written to the RAM into address $c(n) + N_s/3$ (Port B), while data is read out from address $c(n)$ (Port A). In this way, every sample that we read at time n is a delayed version of x, namely $x(n - N_s/3)$. For example, if $N_s/3 = 10$, then we would write samples $x(0)$ through $x(10)$ into addresses 10 through 20. Then at sample time $n = 10$, we can read out the contents of the buffer one sample at a time, such that at sample time $n = 20$ we will have retrieved samples $x(0)$ through $x(10)$ again. Various assumptions made during hardware design require F_s to be in the range of 5 to 125 kHz (precision of 1 Hz).

Spike Alignment

All four alignment algorithms were also coded completely in Simulink using the above protocol, and the alignment hardware is configured in the same way as the detection hardware. Variable-length delays are also used for spike buffering in the alignment block.

Clustering

The implementation of Osort was more complex, as this algorithm involves not only arithmetic operations but also large amounts of memory and complex control logic. Thus Osort was broken down into three discrete modules: logic, memory, and control (Fig. 7.4). The logic module was implemented in Simulink and included blocks for: clustering threshold calculation, distance calculation, minimum distance calculation, cluster mean update, and cluster merge. The memory module was also implemented in Simulink using Xilinx block

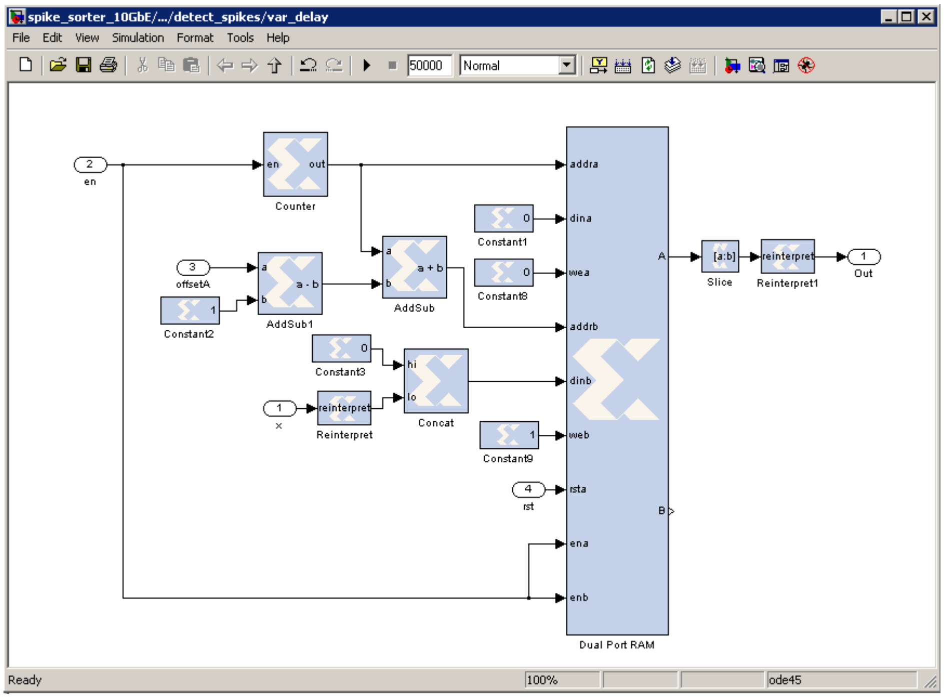


Figure 7.5: Implementation of a variable-length delay used in spike buffering for arbitrary sampling rates.

RAMs and included the memory for storing the cluster means and the cluster sizes. We included enough memory to accomodate 44 clusters, which we found in simulations to be the average maximum number of clusters existing at one time, but we also implemented logic in the controller that will delete the smallest cluster if all memory is full and there is a need to create a new cluster. The controller was written in Verilog and integrated with the Simulink model using a Xilinx “black box”.

The “per-spike latency” of the whole spike-sorting circuit—that is, the number of cycles required to process one spike—is dominated by clustering. This is because Osort is largely a serial algorithm; each spike has to go through a sequence of states (Fig. 7.6). Each state involves performing a different combination of operations, each with a different latency (Table 7.1). Each spike, then, can have a different per-spike latency depending on which path it takes through the state machine. The worst-case latency is represented by the path in which a spike is assigned to an existing cluster and a cluster merge is required (States 2 → 3 → 5 → 6 → 7 → 2). Such a case results in a per-spike latency of 266 clock cycles, corresponding to 11 ms in real-time or a real-time spike rate of 88 Hz if we assume a

Table 7.1: Latency of Clustering Operations

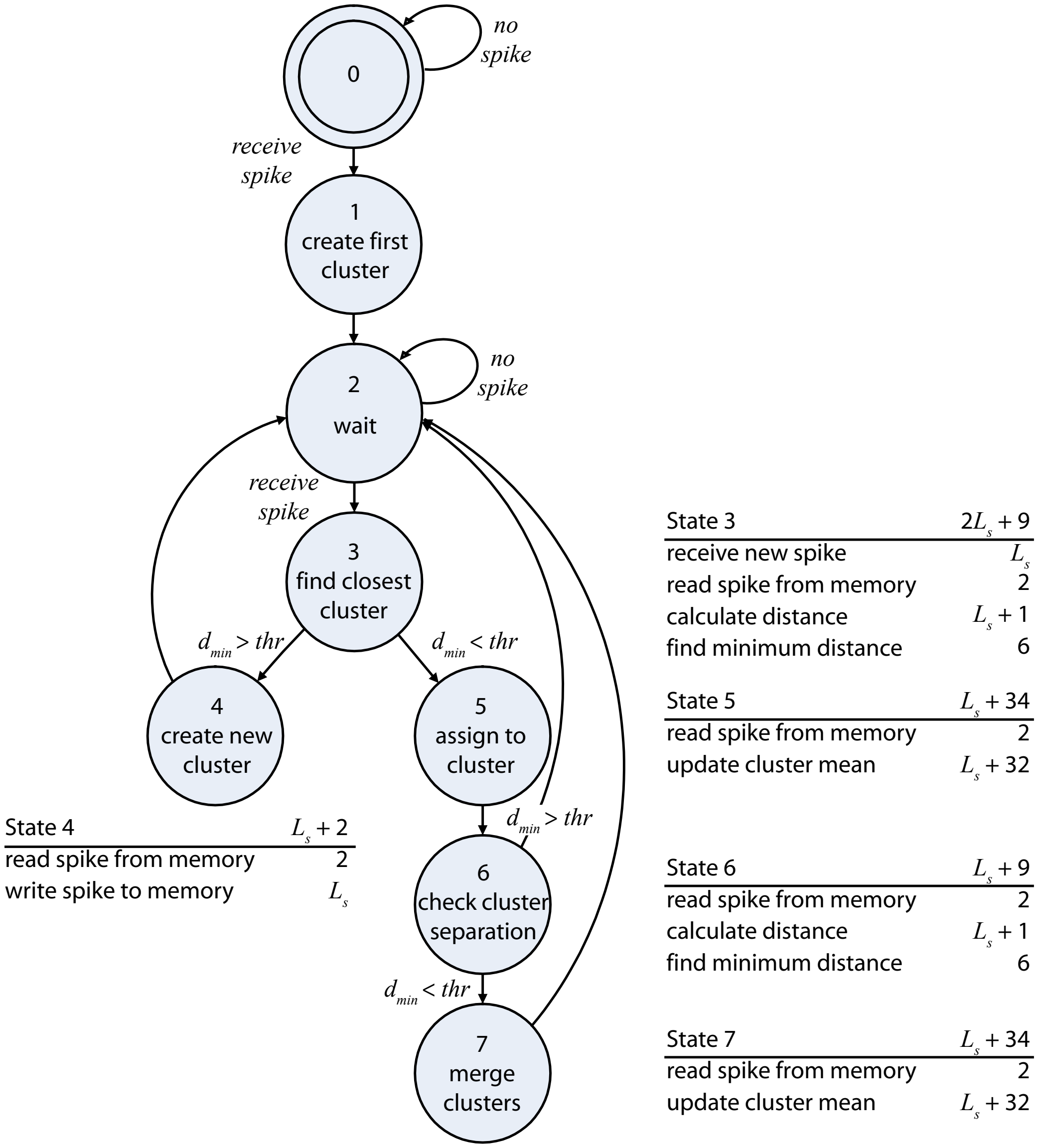
Operation	Latency [number of clock cycles]
receive new spike	L_s
write spike to memory	L_s
read spike from memory	2
calculate distances	$L_s + 1$
find minimum distance	6
update cluster mean	$L_s + 32$

sampling rate of 24 kHz. Since we would like to be able to support any arbitrary spike rate, we have logic that monitors the state of the clustering state machine. If the clusterer is in the "waiting" state (State 2), a new detected spike can be fed into the clusterer. If, however, the clusterer is still busy processing the previous spike, the current data sample is held in memory (i.e. the address on the input buffer is not incremented) until the clusterer returns to State 2.

7.3.4 Python Wrapper

We developed a custom Python script that programs the FPGA (using KATCP) and sends/ receives data to/from the FPGA via the 10GbE interface. The Python script creates a network socket and defines two threads: a transmitter and a receiver. The transmitter reads 8120 bytes[3] from the input binary file, creates a UDP packet, and sends the packet to the socket. This packet is received by the 10GbE transceiver on the FPGA. The FPGA processes the data and sends a packet of results back to the 10GbE transceiver. When the receiver thread, which has been listening for incoming packets, receives this packet, it writes the packet to a binary file and tells the transmitter to send a new packet to the FPGA. This process continues until the entire input file has been processed.

[3] We found experimentally that the size of the buffer in the 10GbE transceiver limits the size of a packet to 8120 bytes (1015 64-bit words).



State 3	$2L_s + 9$
receive new spike	L_s
read spike from memory	2
calculate distance	$L_s + 1$
find minimum distance	6

State 4	$L_s + 2$
read spike from memory	2
write spike to memory	L_s

State 5	$L_s + 34$
read spike from memory	2
update cluster mean	$L_s + 32$

State 6	$L_s + 9$
read spike from memory	2
calculate distance	$L_s + 1$
find minimum distance	6

State 7	$L_s + 34$
read spike from memory	2
update cluster mean	$L_s + 32$

Figure 7.6: State diagram of the Osort algorithm. Tables next to each state show which operations are performed in that state and the associated latencies in number of clock cycles, where L_s is the number of samples in a spike. The worst-case latency is incurred in the state transitions 2 → 3 → 5 → 6 → 7 → 2, which has a latency of $5L_s + 86$ clock cycles. For a sampling rate of 24 kHz, $L_s = 36$ and the worst-case latency is 266 clock cycles or 11 ms of real time.

7.3.5 MATLAB Wrapper

We also developed a MATLAB script to serve as the interface to the user. This ensures that everything running "under the hood"—both the hardware and Python—are abstracted away from the user. The user starts the program by typing a command into the MATLAB command window. This opens the GUI, where the user can input the desired spike-sorting parameters. When the user presses one of the start buttons ('Test' or 'Run'), MATLAB calls the Python script. When the Python script finishes running, indicating that the processing has completed, MATLAB reads the results from the binary files and plots the results in the GUI. The GUI (Fig. 7.1) was designed to be user-friendly and to resemble the Osort GUI in order to provide ease of use for our collaborators, who are accustomed to using the Osort software package.

7.4 Performance

7.4.1 Accuracy

Because we directly mapped each algorithm to hardware, the accuracy of spike sorting using this tool is the same as the accuracy using equivalent software tools. We provided an analysis of the accuracy of both detection algorithms in [64]; both algorithms were shown to have median accuracies of greater than 90% across all SNRs (see Chapter 3). An analysis of the accuracy of Osort can be found in the original paper [69], as well as in Chapter 6 of this thesis.

7.4.2 Processing Speed

We have achieved a processing time of 136 μs per 8120-byte packet, or a processing speed of about 60 MBps. To benchmark this performance against that of conventional software, we measured the speed at which the Osort software package [28], which runs in MATLAB, could process one data file on three different computers with varying processing powers. Figure 7.7(a) shows that this hardware tool can process 25-times faster than the equivalent

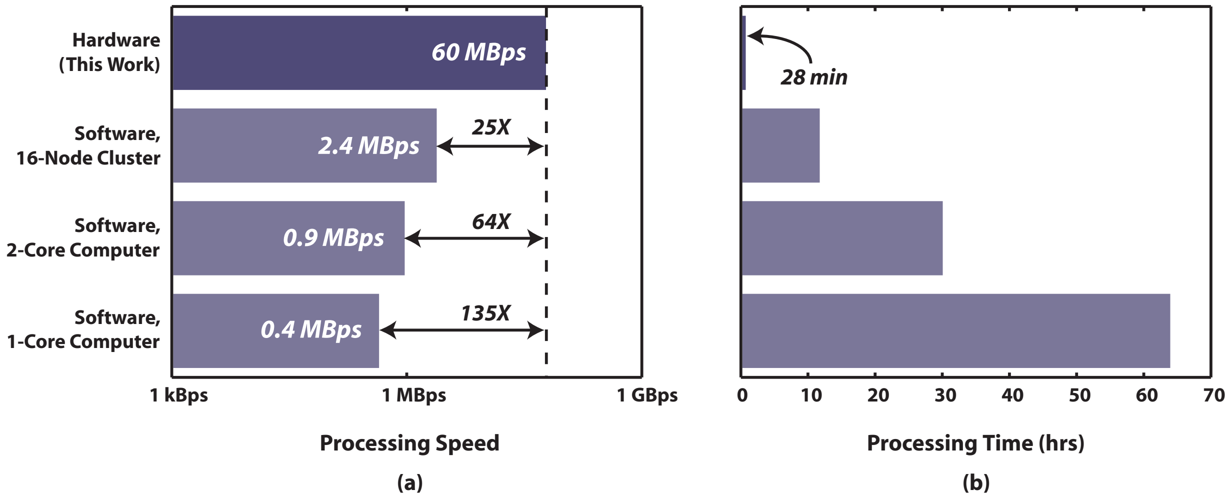


Figure 7.7: (a) Speed improvements of this hardware tool over the equivalent software running on three different computers with different processing powers. Note the logarithmic scale on the horizontal axis. Specifications of each computer are given in Table 7.2. (b) Time to process 8 hours of experimental data, assuming 64 channels sampled at 27.777 kHz and quantized to 16 bits.

Table 7.2: Computer Specifications

	Windows Server	**Windows Laptop**	**Linux Cluster**
Number of Nodes	1	2	16
Model	Intel Pentium D	Intel Core 2 Duo T7250	AMD Opteron Proc 6136
Clock Speed (MHz)	2790	2000	2400
L2 Cache (kB)	2048	2048	512
MATLAB Version	R2006b	R2009b	R2010b
MATLAB Wordlength	32	32	64

software running on the fastest computer available to us (a 16-node Linux cluster), and is over 100-times faster than a single-core computer. To continue the example given in Section 7.1.1, this tool can decrease the time required to process 8 hours of experimental data from about 30 hours on a duo-core computer (12 hours on a 16-node cluster, 64 hours on a single-core computer) to only 28 minutes (Fig. 7.7(b)).

Specifications for each computer used in this benchmark are provided for reference in Table 7.2. Note that in the case of the cluster, the entire 16-node computer was reserved for this experiment in order to ensure that we were allocated 100% of the computer's resources. Also, it was assumed that channels of data would have to be processed serially, but

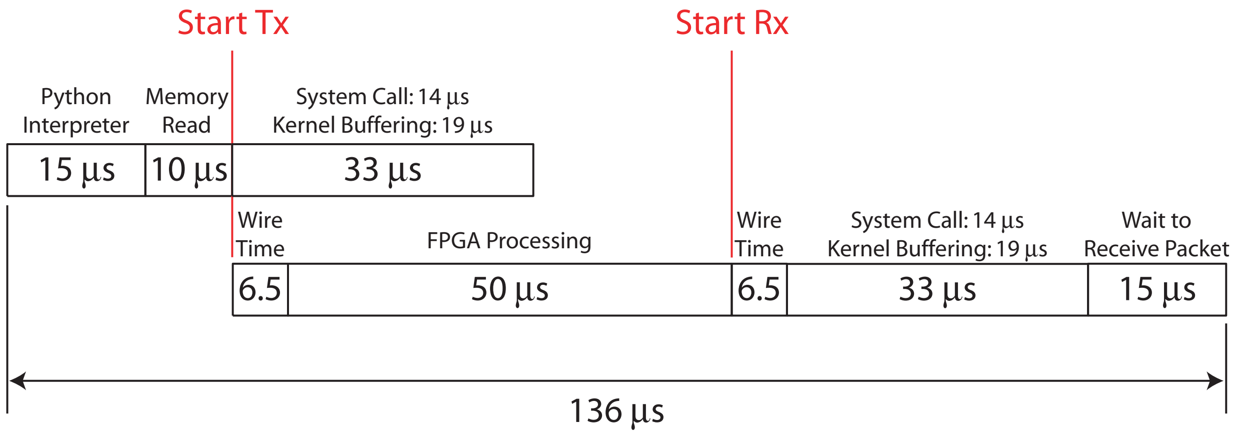


Figure 7.8: Breakdown of how various system components contribute to the total time to process one 8120-byte packet.

MATLAB's implicit multithreading was utilized to take advantage of the multiple cores.

7.4.3 Timing Analysis

Although the FPGA runs at 100 MHz (200 MBps), the total effective processing speed is currently limited to 60 MBps. Of the 136 μs processing time required per 8120-byte packet, almost two thirds is for data transfer and the rest is for FPGA processing. The bandwidth limitations in transferring data from the computer to the FPGA seems to be a result of software and operating-system overhead. Figure 7.8 provides a breakdown of how various components contribute to the data I/O speed. Sending each packet requires calling the Python interpreter (15 μs), reading data from the hard disk into main memory (10 μs), a system call to the socket (14 μs), and moving the UDP packet to the kernel (19 μs). We are incurring a penalty by using an interpreted language (Python) for the software wrapper; switching to a compiled language like C may allow us to reduce the transfer time. Requiring a system call for every individual packet also adds significant latency, especially since the maximum packet size is so small (8120 bytes). If the system call could be modified to move more than one UDP packet to the kernel at once, the transfer time would be greatly reduced.

7.5 Tool Availability

To use this tool, users will need the bit file, Python wrapper, and MATLAB wrapper developed here, all of which are available for free download at: http://icslwebs.ee.ucla.edu/dejan/researchwiki/index.php?title=FPGA_Spike-Sorting_Platform. This webpage contains all the necessary files for running the tool, as well as instructions for how to set up and use the tool. Note that users will also require the hardware on which to run the bit file: a ROACH FPGA processing board, which can be purchased from the CASPER group at UC Berkeley, and a 10GbE cable and network adapter. You may also contact the authors to arrange a demonstration with our hardware.

7.6 Conclusion

We have presented a new hardware accelerator that can increase the speed of offline spike sorting by at least 25 times, effectively reducing the time required to sort data from long experiments from many hours to just one hour. We attempted to preserve the flexibility of software by implementing several different algorithms in the design, and by providing user control over parameters such as the detection threshold. By implementing the tool using an FPGA, we leave room for future improvements and add-ons to the tool.

The tool has been posted online and is currently undergoing beta testing, where we hope to gather feedback from actual users in regards to both the user interface and the tool features. We plan to incorporate user suggestions into later versions of the tool. One idea for expanding the functionality of the tool is to include configurable digital filters at the front end to provide another dimension of flexibility for experimenters.

7.7 Acknowledgments

We thank Mr. V. Karkare for providing the Verilog code for the controller of the Osort ASIC, which formed the basis for the controller here. We also thank Ms. J. Guo for developing the

Python script for sending and receiving data to and from the FPGA and for performing the timing analysis of software overhead. Finally, we thank Mr. H. Chen for his invaluable help in setting up and teaching us how to use the ROACH infrastructure.

CHAPTER 8

Conclusion

8.1 Contributions

First, we developed synthetic datasets that can be used to obtain an accurate, unbiased comparison between spike-sorting algorithms. These datasets are biologically realistic, have known ground truths, and cover a wide range of SNRs and levels of difficulty. We also proposed a methodology for evaluating the tradeoffs between accuracy and computational complexity of spike-sorting algorithms. We then used these datasets and methodology to evaluate several published spike-sorting algorithms in order to determine which are most appropriate for implementation in real-time, low-power hardware. It is our hope that other researchers will also utilize our library of datasets and/or our methodology when publishing new algorithms in the future in order to present new work fairly against the context of existing work. We also presented an analysis of the tradeoffs between digital and analog spike detection. We found that the choice of whether to implement detection in the digital or analog domain is dependent on the desired resolution of the data. This analysis can serve as a reference for anyone designing spike-sorting systems in the future. All of this work led to the design of two low-power spike-sorting chips, both of which can be used in real-time implantable recording systems to reduce the data rate while at the same time providing useful computations.

We also provided an analysis of the effects of quantization on spike sorting, and of the effects of nonlinear quantization on spike sorting. We found that optimal quantizers can improve the accuracy of spike sorting at a given resolution, or can provide the same accuracy with fewer bits. We should note, however, that it remains to be seen what effect nonlinear quantization has downstream—for example, on the DSP. How would the design of the DSP have to change to accomodate nonlinearly quantized data? Would this increase or decrease the complexity of the DSP? If the complexity of the ADC decreases but the complexity of the DSP increases, have we gained or lost in terms of the overall system complexity?

Finally, we developed an FPGA-based spike-sorting platform that can increase the speed of offline spike sorting by at least 11 times, effectively reducing the time required to sort data from long experiments from many hours to just one hour. We attempted to preserve the flexibility of software by implementing several different algorithms in the design, and by providing user control over parameters such as spike detection thresholds. We hope that this tool finds use in the neuroscience community. We also hope to receive feedback from actual users regarding how the tool can be improved, in which case we may release add-ons with new features.

8.2 Looking to the Future

The work presented in this thesis can be continued, improved upon, or added to in many ways. In Chapter 3, we were able to evaluate only a subset of the many spike-sorting algorithms that exist; the same evaluation could be applied on the many algorithms that were not studied here. We could also evaluate the hardware feasibility of the "alternative" methods for spike sorting presented in Section 2.4.6. We could look for methods that can resolve overlapping spikes. We could perform the same type of analysis on alignment algorithms, and test whether or not interpolation improves the accuracy of spike sorting (and whether or not it is worth the increase in computational complexity).

Our spike-sorting algorithm analysis also assumed data that had been recorded with single-electrode probes. Many researchers, however, use tetrodes, which have four closely spaced electrodes on each probe. Data recorded using tetrodes can be processed using different techniques that exploit the dependence between channels. It could be useful to perform similar algorithm evaluations on these kinds of algorithms.

For BMIs, I think that the real excitement lies in the next stage of processing after spike sorting: decoding. Several independent labs have had success in decoding neural signals from various brain areas in various species [11–15]. But for this technology to become a therapeutic reality, these algorithms also need to be implemented in hardware. Which algorithms are most accurate? Which are most robust to noise? To changing recording conditions? Which algorithms are even feasible for hardware implementation? How sophisticated do these algorithms really need to be, and how much can we rely on the plasticity of the brain for learning to control the BMI? This is certainly an exciting time to be in the field of neuroengineering.

References

[1] N. Kipnis, "Luigi Galvani and the Debate on Animal Electricity, 1791-1800," *Ann. Sci.*, vol. 44, no. 2, pp. 107–142, 1987. 1.1

[2] A. L. Hodgkin and A. F. Huxley, "Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo," *J. Physiol.*, vol. 116, no. 4, pp. 449–472, 1952. 1.1, 2.1.1

[3] ——, "The components of membrane conductance in the giant axon of Loligo," *J. Physiol.*, vol. 116, no. 4, pp. 473–496, 1952. 1.1, 2.1.1

[4] ——, "The dual effect of membrane potential on sodium conductance in the giant axon of Loligo," *J. Physiol.*, vol. 116, no. 4, pp. 497–506, 1952. 1.1, 2.1.1

[5] ——, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol.*, vol. 117, no. 4, pp. 500–544, 1952. 1.1, 2.1.1

[6] D. H. Hubel and T. N. Wiesel, "Ferrier lecture: functional architecture of macaque monkey visual cortex," *Proc. R. Soc. Lond. B.*, vol. 198, no. 1130, pp. 1–59, 1977. 1.1

[7] K. Sameshima and L. A. Baccalá, "Trends in Multichannel Neural Ensemble Recording Instrumentation," in *Methods for Neural Ensemble Recordings*, M. A. L. Nicolelis, Ed. Boca Raton: CRC Press, 1999, ch. 3, pp. 47–60. 1.1

[8] A. Bragin, C. L. Wilson, R. J. Staba, M. Reddick, I. Fried, and J. Engel Jr., "Interictal high-frequency oscillations (80-500Hz) in the human epileptic brain: entorhinal cortex," *Ann. Neurol.*, vol. 52, no. 4, pp. 407–415, 2002. 1.1

[9] R. J. Staba, C. L. Wilson, A. Bragin, I. Fried, and J. Engel, "Quantitative analysis of high-frequency oscillations (80-500 Hz) recorded in human epileptic hippocampus and entorhinal cortex." *J. Neurophysiol.*, vol. 88, no. 4, pp. 1743–1752, Oct. 2002. 1.1

[10] R. J. Staba, L. Frighetto, E. J. Behnke, G. W. Mathern, T. Fields, A. Bragin, J. Ogren, I. Fried, C. L. Wilson, and J. Engel, "Increased fast ripple to ripple ratios correlate with reduced hippocampal volumes and neuron loss in temporal lobe epilepsy patients." *Epilepsia*, vol. 48, no. 11, pp. 2130–2138, Nov. 2007. 1.1

[11] M. A. L. Nicolelis, "Actions from thoughts," *Nature*, vol. 409, no. 6818, pp. 403–407, 2001. 1.1, 8.2

[12] L. R. Hochberg, M. D. Serruya, G. M. Friehs, J. A. Mukand, M. Saleh, A. H. Caplan, A. Branner, D. Chen, R. D. Penn, and J. P. Donoghue, "Neuronal ensemble control of prosthetic devices by a human with tetraplegia," *Nature*, vol. 442, no. 7099, pp. 164–171, 2006. 1.1, 8.2

[13] T. W. Berger, A. Ahuja, S. H. Courellis, S. A. Deadwyler, G. Erinjippurath, G. A. Gerhardt, G. Gholmieh, J. J. Granacki, R. Hampson, M. C. Hsaio, J. Lacoss, V. Z. Marmarelis, P. Nasiatka, V. Srinivasan, D. Song, A. R. Tanguay, and J. Wills, "Restoring lost cognitive function," *IEEE Eng. Med. Biol. Mag.*, vol. 24, no. 5, pp. 30–44, 2005. 1.1, 8.2

[14] D. M. Taylor, S. I. H. Tillery, and A. B. Schwartz, "Direct cortical control of 3D neuroprosthetic devices." *Science*, vol. 296, no. 5574, pp. 1829–1832, Jun. 2002. 1.1, 2.7, 8.2

[15] K. V. Shenoy, D. Meeker, S. Cao, S. A. Kureshi, B. Pesaran, C. A. Buneo, A. P. Batista, P. P. Mitra, J. W. Burdick, and R. A. Andersen, "Neural prosthetic control signals from plan activity," *Neuroreport*, vol. 14, no. 4, pp. 591–596, 2003. 1.1, 2.7, 8.2

[16] E. M. Schmidt, "Electrodes for Many Single Neuron Recordings," in *Methods for Neural Ensemble Recordings*, M. A. L. Nicolelis, Ed. Boca Raton: CRC Press, 1999, ch. 1, pp. 1–23. 1

[17] S. Suner, M. R. Fellows, C. Vargas-irwin, G. K. Nakata, and J. P. Donoghue, "Reliability of Signals From a Chronically Implanted , Silicon-Based Electrode Array in Non-Human Primate Primary Motor Cortex," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 13, no. 4, pp. 524–541, 2005. 1

[18] Plexon, "Offline Sorter," 2012. [Online]. Available: http://www.plexon.com/product/Offline_Sorter.html 1.3

[19] R. R. Harrison, "A Low-Power Integrated Circuit for Adaptive Detection of Action Potentials in Noisy Signals," in *Proc. 25th Ann. Int. Conf. IEEE EMBS*, Cancun, Mexico, 2003, pp. 3325–3328. 1.3.1, 4.1

[20] R. Harrison, P. T. Watkins, R. J. Kier, R. O. Lovejoy, D. J. Black, B. Greger, and F. Solzbacher, "A Low-Power Integrated Circuit for a Wireless 100-Electrode Neural Recording System," *IEEE J. Solid-State Circuits*, vol. 42, no. 1, pp. 123–133, 2007. 2, 4.1, 1, 2

[21] T. M. Seese, H. Harasaki, G. M. Saidel, and C. R. Davies, "Characterization of tissue morphology, angiogenesis, and temperature in the adaptive response of muscle tissue in chronic heating," *Lab. Invest.*, vol. 78, no. 12, pp. 1553–1562, Dec. 1998. 1.3.1

[22] Plexon, "Multichannel Acquisition Processor (MAP)," 2012. [Online]. Available: http://www.plexon.com/product/Multichannel_Acquisition_Processor__MAP__.html#Description 1.3.2, 5.1, 7.1.1

[23] ——, "OmniPlex," 2012. [Online]. Available: http://www.plexon.com/product/OmniPlex.html 1.3.2, 5.1, 7.1.1

[24] ——, "Recorder," 2012. [Online]. Available: http://www.plexon.com/product/Recorder.html 1.3.2, 5.1, 7.1.1

[25] Neuralynx, “Digital Lynx,” 2011. [Online]. Available: http://neuralynx.com/products/digital_data_acquisition_systems/ 1.3.2, 5.1, 7.1.1

[26] Blackrock Microsystems, “Cerebus,” 2012. [Online]. Available: http://www.blackrockmicro.com/content.aspx?id=13 1.3.2, 5.1, 7.1.1

[27] Ripple, “Grapevine NIP,” 2012. [Online]. Available: https://www.rppl.com/products/grapevine-neural-interface-processors/item/100-grapevine-nip 1.3.2, 5.1, 7.1.1

[28] U. Rutishauser, “Osort,” 2011. [Online]. Available: http://www.urut.ch/new/serendipity/index.php?/pages/osort.html 4, 1, 7.4.2

[29] G. Buzsáki, M. Penttonen, Z. Nádasdy, and A. Bragin, “Pattern and inhibition-dependent invasion of pyramidal cell dendrites by fast spikes in the hippocampus in vivo.” *Proc. Natl. Acad. Sci. U.S.A.*, vol. 93, no. 18, pp. 9921–5, Sep. 1996. 2.1

[30] Z. Nádasdy, J. Csicsvari, M. Penttonen, J. Hetke, K. Wise, and G. Buzsáki, “Extracellular Recording and Analysis of Neuronal Activity: From Single Cells to Ensembles,” in *Neuronal Ensembles: Strategies for Recording and Decoding*, H. B. Eichenbaum and J. L. Davis, Eds. New York: Wiley-Liss, 1998, ch. 2, pp. 17–55. 2.1.1, 2.2

[31] W. H. Freygang and K. Frank, “Extracellular potentials from single spinal motoneurons,” *J. Gen. Physiol*, vol. 42, no. 4, pp. 749–760, 1959. 2.2

[32] M. S. Fee, P. P. Mitra, and D. Kleinfeld, “Variability of Extracellular Spike Waveforms of Cortical Neurons,” *J. Neurophysiol.*, vol. 76, no. 6, pp. 3823–3833, 1996. 2.1.1, 2.5.2, 2.5.3

[33] F. W. Campbell, B. G. Cleland, G. F. Cooper, and C. Enroth-Cugell, “The angular selectivity of visual cortical cells to moving gratings.” *J. Physiol.*, vol. 198, no. 1, pp. 237–50, Sep. 1968. 2.1.1

[34] C. M. Gray and G. Viana Di Prisco, “Stimulus-dependent neuronal oscillations and local synchronization in striate cortex of the alert cat.” *J. Neurosci.*, vol. 17, no. 9, pp. 3239–53, May 1997. 2.1.1

[35] E. D. Adrian, “The impulses produced by sensory nerve-endings: Part 4. Impulses from pain receptors.” *J. Physiol.*, vol. 62, no. 1, pp. 33–51, 1926.

[36] D. H. Perkel and T. H. Bullock, “Neural Coding,” *Neurosciences Res. Prog. Bull.*, vol. 6, no. 3, pp. 221–348, 1968. 2.1.1

[37] J. J. Hopfield, “Pattern recognition computation using action potential timing for stimulus representation,” *Nature*, vol. 376, no. 6535, pp. 33–36, 1995. 2.1.1

[38] J. O’Keefe and M. L. Recce, “Phase relationship between hippocampal place units and the EEG theta rhythm.” *Hippocampus*, vol. 3, no. 3, pp. 317–330, Jul. 1993. 2.1.1

[39] Z. Nadasdy, “Information encoding and reconstruction from the phase of action potentials.” *Front. Syst. Neurosci.*, vol. 3, no. July, p. 6, Jan. 2009. 2.1.1

[40] G. Buzsáki and R. D. Traub, "Physiologic Basis of the Electroencephalogram and Local Field Potentials," in *Epilepsy: A Comprehensive Textbook*, 2nd ed., J. Engel Jr. and T. A. Pedley, Eds. Philadelphia, PA: Lippincott Williams & Wilkins, 2008, ch. 72, pp. 797–807. 2.1.2

[41] F. Wood, M. J. Black, C. Vargas-Irwin, M. Fellows, and J. P. Donoghue, "On the variability of manual spike sorting," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 6, pp. 912–918, 2004. 2.3, 2.4.5, 2.5.1

[42] K. D. Harris, D. A. Henze, J. Csicsvari, H. Hirase, and G. Buzsáki, "Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements." *J. Neurophysiol.*, vol. 84, no. 1, pp. 401–414, Jul. 2000. 2.3, 2.4.5, 2.1, 2.5.1

[43] W. L. G. Koontz, P. M. Narendra, and K. Fukunaga, "A Graph-Theoretic Approach to Nonparametric Cluster Analysis," *IEEE Trans. Computers*, vol. C-25, no. 9, pp. 936–944, Sep. 1976. 2.3

[44] M. S. Lewicki, "A review of methods for spike sorting: The detection and classification of neural action potentials," *Network: Comput. Neural Syst.*, vol. 9, no. 4, pp. R53–R78, Nov. 1998. 2.4, 2.4.1, 2.4.2, 2.4.3, 2.4.5, 2.5.1, 2.5.3

[45] I. Obeid and P. D. Wolf, "Evaluation of Spike-Detection Algorithms for a Brain-Machine Interface Application," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 6, pp. 905–911, Jun. 2004. 2.4.1, 2.4.1, 4.2.1.1

[46] R. Q. Quiroga, Z. Nadasdy, and Y. Ben-Shaul, "Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering." *Neural Comput.*, vol. 16, no. 8, pp. 1661–1687, Aug. 2004. 2.4.1, 2.4.3, 2.4.4, 2.4.5, 2.5.1, 3.1, 3.2

[47] M. J. Black, G. Sapiro, D. H. Marimont, and D. Heeger, "Robust anisotropic diffusion." *IEEE Trans. Image Processing*, vol. 7, no. 3, pp. 421–32, Jan. 1998. 1

[48] K. H. Kim and S. J. Kim, "A Wavelet-Based method for Action Potential Detection From Extracellular Neural Signal Recording With Low Signal-to-Noise Ratio," *IEEE Trans. Biomed. Eng.*, vol. 50, no. 8, pp. 999–1011, Aug. 2003. 2.4, 2.4.1, 3.1

[49] J. F. Kaiser, "On a simple algorithm to calculate the 'energy' of a signal," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP '90)*, vol. 1, Albuquerque, NM, 1990, pp. 381–384. 2.4.1, 4.2.1.1

[50] S. Mukhopadhyay and G. C. Ray, "A New Interpretation of Nonlinear Energy Operator and Its Efficacy in Spike Detection," *IEEE Trans. Biomed. Eng.*, vol. 45, no. 2, pp. 180–187, Feb. 1998. 2.4.1, 2.4.1, 2.4.1, 3.1, 4.2.1.1

[51] K. H. Kim and S. J. Kim, "Neural Spike Sorting Under Nearly 0-dB Signal-to-Noise Ratio Using Nonlinear Energy Operator and Artificial Neural-Network Classifier," *IEEE Trans. Biomed. Eng.*, vol. 47, no. 10, pp. 1406–1411, Oct. 2000. 2.4.1, 3.1, 4.2.1.1

[52] E. Hulata, R. Segev, Y. Shapira, M. Benveniste, and E. Ben-Jacob, “Detection and Sorting of Neural Spikes Using Wavelet Packets,” *Phys. Rev. Lett.*, vol. 85, no. 21, pp. 4637–4640, 2000. 2.4.1

[53] R. J. Brychta, S. Tuntrakool, M. Appalsamy, N. R. Keller, D. Robertson, R. G. Shiavi, and A. Diedrich, “Wavelet Methods for Spike Detection in Mouse Renal Sympathetic Nerve Activity,” *IEEE Trans. Biomed. Eng.*, vol. 54, no. 1, pp. 82–93, 2007. 2.4.1

[54] R. Chandra and L. M. Optican, “Detection, classification, and superposition resolution of action potentials in multiunit single-channel recordings by an on-line real-time neural network,” *IEEE Trans. Biomed. Eng.*, vol. 44, no. 5, pp. 403–412, May 1997. 2.4.2, 2.5.4

[55] J. H. Choi, H. K. Jung, and T. Kim, “A New Action Potential Detector Using the MTEO and Its Effects on Spike Sorting Systems at Low Signal-to-Noise Ratios,” *IEEE Trans. Biomed. Eng.*, vol. 53, no. 4, pp. 738–746, 2006. 2.4.2

[56] A. Zviagintsev, Y. Perelman, and R. Ginosar, “Algorithms and architectures for low power spike detection and alignment,” *J. Neural Eng.*, vol. 3, no. 1, pp. 35–42, 2006. 2.4.2, 3.1

[57] M. Sahani, “Latent Variable Models for Neural Data Analysis,” Ph.D. dissertation, California Institute of Technology, Pasadena, California, May 1999. 2.4.2, 2.4.6

[58] M. Abeles and M. H. Goldstein Jr., “Multispike Train Analysis,” *Proc. IEEE*, vol. 65, no. 5, pp. 762–773, May 1977. 2.4.3, 2.5.2

[59] Z. Nadasdy, R. Quian Quiroga, Y. Ben-Shaul, B. Pesaran, D. A. Wagenaar, and R. A. Andersen, “Comparison of unsupervised algorithms for on-line and off-line spike sorting,” in *32nd Annu. Meeting Soc. for Neurosci.*, 2002. [Online]. Available: http://www.vis.caltech.edu/~{}zoltan/ 2.4.3

[60] A. Zviagintsev, Y. Perelman, and R. Ginosar, “Low-Power Architectures for Spike Sorting,” in *Proc. 4th Int. IEEE EMBS Conf. Neural Eng.*, Arlington, VA, Mar. 2005, pp. 162–165. 2.4.3, 3.1

[61] H. W. Lilliefors, “On the Kolmogorov-Smirnov Test for Normality with Mean and Variance Unknown,” *J. Amer. Statistical Assoc.*, vol. 62, no. 318, pp. 399–402, Jun. 1967. 2.4.4

[62] J. A. Hartigan and P. M. Hartigan, “The Dip Test of Unimodality,” *Ann. Stat.*, vol. 13, no. 1, pp. 70–84, Mar. 1985. 2.4.4

[63] P. M. Hartigan, “Computation of the Dip Statistic to Test for Unimodality,” *J. Appl. Statist.*, vol. 34, no. 3, pp. 320–325, 1985. 2.4.4

[64] S. Gibson, J. W. Judy, and D. Marković, “Technology-aware algorithm design for neural spike detection, feature extraction, and dimensionality reduction.” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 18, no. 5, pp. 469–78, Oct. 2010. 2.4.4, 2.5.1, 3.4, 5.2, 5.4, 7.4.1

[65] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," in *Proc. 5th Berkeley Symp. on Math. Stat. and Prob.*, vol. 1, 1967, pp. 281–297. 2.4.5

[66] D. Novák, J. Wild, T. Sieger, and R. Jech, "Identifying Number of Neurons in Extracellular Recording," in *Proc. 4th Int. IEEE EMBS Conf. Neural Eng.*, Antalya, Turkey, 2009, pp. 742–745. 2.4.5

[67] C. Zhang, X. Zhang, M. Q. Zhang, and Y. Li, "Neighbor number, valley seeking and clustering," *Pattern Recogn. Lett.*, vol. 28, no. 2, pp. 173–180, 2007. 2.4.5

[68] M. Blatt, S. Wiseman, and E. Domany, "Data Clustering Using a Model Granular Magnet," *Neural Comput.*, vol. 9, no. 9, pp. 1805–1842, 1997. 2.4.5, 3

[69] U. Rutishauser, E. M. Schuman, and A. N. Mamelak, "Online detection and sorting of extracellularly recorded action potentials in human medial temporal lobe recordings, in vivo." *J. Neurosci. Methods*, vol. 154, no. 1-2, pp. 204–224, Jun. 2006. 2.4.5, 2.5.1, 5.2, 5.2, 6.2.1, 7.4.1

[70] V. Karkare, S. Gibson, C.-H. Yang, H. Chen, and D. Marković, "A 75μW, 16-Channel Neural Spike-Sorting Processor with Unsupervised Clustering," in *Proc. Int. Symp. on VLSI Circuits*, Kyoto, Japan, 2011, pp. 252–253. 2.1, 1, 6.2, 6.2.1, 2

[71] M. S. Lewicki, "Bayesian Modeling and Classification of Neural Signals," *Neural Comput.*, vol. 6, no. 5, pp. 1005–1030, 1994. 2.4.6, 2.5.4

[72] M. Sahani, J. S. Pezaris, and R. A. Andersen, "Extracellular recording from multiple neighboring cells: a maximum-likelihood solution to the spike-separation problem," in *Proc. 6th Ann. Conf. Comput. Neurosci. (CNS97)*. New York, NY, USA: Plenum Press, 1998, pp. 619–625. 2.4.6

[73] C. Pouzat, O. Mazor, and G. Laurent, "Using noise signature to optimize spike-sorting and to assess neuronal classification quality," *J. Neurosci. Methods*, vol. 122, no. 1, pp. 43–57, 2002. 2.4.6, 2.4.6, 2.5.1

[74] S. Shoham, M. R. Fellows, and R. A. Normann, "Robust, automatic spike sorting using mixtures of multivariate t-distributions," *J. Neurosci. Methods*, vol. 127, no. 2, pp. 111–122, 2003. 2.4.6, 2.5.2

[75] D. A. Henze, Z. Borhegyi, J. Csicsvari, A. Mamiya, K. D. Harris, and G. Buzsáki, "Intracellular features predicted by extracellular recordings in the hippocampus in vivo." *J. Neurophysiol.*, vol. 84, no. 1, pp. 390–400, Jul. 2000. 2.5.1, 1

[76] S. Gibson, J. W. Judy, and D. Marković, "Comparison of Spike-Sorting Algorithms for Future Hardware Implementation," in *Proc. 30th Ann. Int. Conf. IEEE EMBS*, Vancouver, Canada, 2008, pp. 5015–5020. 2.5.1, 4.2.4

[77] L. S. Smith and N. Mtetwa, "A tool for synthesizing spike trains with realistic interference," *J. Neurosci. Methods*, vol. 159, no. 1, pp. 170–180, 2007. 2.5.1

[78] J. Martinez, C. Pedreira, M. J. Ison, and R. Quian Quiroga, “Realistic simulation of extracellular recordings.” *J. Neurosci. Methods*, vol. 184, no. 2, pp. 285–293, Nov. 2009. 2.5.1

[79] D. H. Perkel, G. L. Gerstein, and G. P. Moore, “Neuronal spike trains and stochastic point processes: I. The single spike train,” *Biophys. J.*, vol. 7, no. 4, pp. 391–418, 1967. 2.5.1

[80] A. Tankus, Y. Yeshurun, and I. Fried, “An automatic measure for classifying clusters of suspected spikes into single cells versus multiunits.” *J. Neural Eng.*, vol. 6, no. 5, p. 056001, Oct. 2009. 2.5.1

[81] R. K. Snider and A. B. Bonds, “Classification of non-stationary neural signals,” *J. Neurosci. Methods*, vol. 84, no. 1-2, pp. 155–166, 1998. 2.5.3

[82] A. Bar-Hillel, A. Spiro, and E. Stark, “Spike sorting: Bayesian clustering of non-stationary data,” *J. Neurosci. Methods*, vol. 157, no. 2, pp. 303–316, 2006. 2.5.3

[83] V. J. Prochazka and H. H. Kornhuber, “On-line multi-unit sorting with resolution of superposition potentials,” *Electroencephalogr. Clin. Neurophysiol.*, vol. 34, no. 1, pp. 91–93, 1973. 2.5.4

[84] A. F. Atiya, “Recognition of Multiunit Neural Signals,” *IEEE Trans. Biomed. Eng.*, vol. 39, no. 7, pp. 723–729, 1992. 2.5.4

[85] S. Takahashi, Y. Anzai, and Y. Sakurai, “Automatic sorting for multi-neuronal activity recorded with tetrodes in the presence of overlapping spikes.” *J. Neurophysiol.*, vol. 89, no. 4, pp. 2245–2258, Apr. 2003. 2.5.4

[86] W. Ding and J. Yuan, “Spike sorting based on multi-class support vector machine with superposition resolution,” *Med. Bio. Eng. Comput.*, vol. 46, no. 2, pp. 139–145, 2008. 2.5.4

[87] K. Y. Kwon, S. Eldawlatly, and K. G. Oweiss, “NeuroQuest: A comprehensive tool for large scale neural data processing and analysis,” in *Proc. 4th Int. IEEE EMBS Conf. Neural Eng.*, Antalya, Turkey, 2009, pp. 622–625. 2.5.4

[88] G. D. Brown, S. Yamada, and T. J. Sejnowski, “Independent component analysis at the neural cocktail party,” *Trends Neurosci.*, vol. 24, no. 1, pp. 54–63, 2001. 2.6

[89] E. M. Maynard, C. T. Nordhausen, and R. A. Normann, “The Utah Intracortical Array: A recording structure for potential brain-computer interfaces,” *Electroencephalogr. Clin. Neurophysiol.*, vol. 102, pp. 228–239, 1997. 2.6

[90] F. Wood and M. J. Black, “A nonparametric Bayesian alternative to spike sorting.” *J. Neurosci. Methods*, vol. 173, no. 1, pp. 1–12, Aug. 2008. 2.7

[91] V. Ventura, “Spike train decoding without spike sorting.” *Neural Comput.*, vol. 20, no. 4, pp. 923–963, Apr. 2008. 2.7

[92] E. Stark and M. Abeles, "Predicting Movement from Multiunit Activity," *J. Neurosci.*, vol. 27, no. 31, pp. 8387–8394, 2007. 2.7, 2.14

[93] J. M. Carmena, M. A. Lebedev, R. E. Crist, J. E. O'Doherty, D. M. Santucci, D. F. Dimitrov, P. G. Patil, C. S. Henriquez, and M. A. L. Nicolelis, "Learning to control a brain-machine interface for reaching and grasping by primates." *PLoS biology*, vol. 1, no. 2, p. E42, Nov. 2003. 2.7

[94] B. Pesaran, J. S. Pezaris, M. Sahani, P. P. Mitra, and R. A. Andersen, "Temporal structure in neuronal activity during working memory in macaque parietal cortex," *Nat. Neurosci.*, vol. 5, no. 8, pp. 805–811, 2002. 2.7

[95] N. F. Ince, R. Gupta, S. Arica, A. H. Tewfik, J. Ashe, and G. Pellizzer, "Movement direction decoding with spatial patterns of local field potentials," in *Proc. 4th Int. IEEE EMBS Conf. Neural Eng.* Antalya, Turkey: Ieee, Apr. 2009, pp. 291–294. 2.7

[96] C. Mehring, J. Rickert, E. Vaadia, S. Cardosa De Oliveira, A. Aertsen, and S. Rotter, "Inference of hand movements from local field potentials in monkey motor cortex." *Nat. Neurosci.*, vol. 6, no. 12, pp. 1253–1254, Dec. 2003. 2.7

[97] H. Scherberger, M. R. Jarvis, and R. A. Andersen, "Cortical local field potential encodes movement intentions in the posterior parietal cortex." *Neuron*, vol. 46, no. 2, pp. 347–354, Apr. 2005. 2.7

[98] M. Mollazadeh, V. Aggarwal, N. V. Thakor, A. J. Law, A. Davidson, and M. H. Schieber, "Coherency between spike and LFP activity in M1 during hand movements," in *Proc. 4th Int. IEEE EMBS Conf. Neural Eng.* Antalya, Turkey: Ieee, Apr. 2009, pp. 506–509. 2.7

[99] J. Wessberg, C. R. Stambaugh, J. D. Kralik, P. D. Beck, M. Laubach, J. K. Chapin, J. Kim, S. J. Biggs, M. A. Srinivasan, and M. A. L. Nicolelis, "Real-time prediction of hand trajectory by ensembles of cortical neurons in primates," *Nature*, vol. 408, no. 6810, pp. 361–365, 2000. 2.7

[100] M. D. Serruya, N. G. Hatsopoulos, L. Paninski, M. R. Fellows, and J. P. Donoghue, "Instant neural control of a movement signal," *Nature*, vol. 416, no. 6877, pp. 141–142, 2002. 2.7

[101] S. Gibson, R. Chandler, V. Karkare, D. Marković, and J. W. Judy, "An Efficiency Comparison of Analog and Digital Spike Detection," in *Proc. 4th Int. IEEE EMBS Conf. Neural Eng.*, Antalya, Turkey, 2009, pp. 423–428. 3.3.3

[102] P. T. Watkins, G. Santhanam, K. V. Shenoy, and R. R. Harrison, "Validation of Adaptive Threshold Spike Detector for Neural Recording," in *Proc. 26th Ann. Int. Conf. IEEE EMBS*, San Francisco, CA, USA, 2004, pp. 4079–4082. 4.1

[103] A. M. Sodagar, K. D. Wise, and K. Najafi, "A Fully Integrated Mixed-Signal Neural Processor for Implantable Multichannel Cortical Recordings," *IEEE Trans. Biomed. Eng.*, vol. 54, no. 6, pp. 1075–1088, Jun. 2007. 4.1

[104] C. L. Rogers and J. G. Harris, "A Low-Power Analog Spike Detector for Extracellular Neural Recordings," in *Proc. 11th IEEE Int. Conf. Electronics, Circuits, and Systems*, Tel-Aviv, Israel, Dec. 2004, pp. 290–293. 4.1

[105] C. L. Rogers, J. G. Harris, J. C. Principe, and J. C. Sanchez, "An Analog VLSI Implementation of a Multi-Scale Spike Detection Algorithm for Extracellular Neural Recordings," in *Proc. 2nd Int. IEEE EMBS Conf. Neural Eng.*, Arlington, VA, USA, Mar. 2005, pp. 213–216. 4.1

[106] M. Chae, W. Liu, Z. Yang, T. Chen, J. Kim, M. Sivaprakasam, and M. Yuce, "A 128-Channel 6mW Wireless Neural Recording IC with On-the-Fly Spike Sorting and UWB Tansmitter," in *IEEE Int. Solid-State Circuits Conf.*, San Francisco, CA, USA, Feb. 2008, pp. 146–603. 4.1, 6.1, 6.2, 6.2, 6.4

[107] R. H. Olsson and K. D. Wise, "A three-dimensional neural recording microsystem with implantable data compression circuitry," *IEEE J. Solid-State Circuits*, vol. 40, no. 12, pp. 2796–2804, Dec. 2005. 4.1, 5.1, 6.1, 6.2, 6.2, 6.4

[108] M. Rizk, I. Obeid, S. H. Callender, and P. D. Wolf, "A single-chip signal processing and telemetry engine for an implantable 96-channel neural data acquisition system," *J. Neural Eng.*, vol. 4, no. 3, pp. 309–321, Sep. 2007. 4.1, 6.1, 6.2

[109] A. Agnes, E. Bonizzoni, P. Malcovati, and F. Maloberti, "A 9.4-ENOB 1V 3.8μW 100kS/s SAR ADC with Time-Domain Comparator," in *IEEE Int. Solid-State Circuits Conf.* IEEE, 2008, pp. 246–610. 4.2.2

[110] V. Giannini, P. Nuzzo, V. Chironi, A. Baschirotto, G. Van der Plas, and J. Craninckx, "An 820μW 9b 40MS/s noise-tolerant dynamic-SAR ADC in 90nm digital CMOS," in *IEEE Int. Solid-State Circuits Conf.* IEEE, 2008, pp. 238–610. 4.2.2

[111] M. Van Elzakker, E. Van Tuijl, P. Geraedts, D. Schinkel, E. Klumperink, and B. Nauta, "A 1.9μW 4.4 fJ/Conversion-step 10b 1MS/s Charge-Redistribution ADC," in *IEEE Int. Solid-State Circuits Conf.* IEEE, 2008, pp. 244–610. 4.2.2

[112] G. Anelli, "Design and Characterization of Radiation Tolerant Integrated Circuits in Deep Submicron CMOS Technologies for the LHC Experiments," Ph.D. dissertation, CERN, 2000. 4.2.3.3

[113] Z. S. Zumsteg, C. Kemere, S. Member, S. O. Driscoll, G. Santhanam, R. E. Ahmed, K. V. Shenoy, and T. H. Meng, "Power Feasibility of Implantable Digital Spike Sorting Circuits for Neural Prosthetic Systems," *Rehabilitation*, vol. 13, no. 3, pp. 272–279, 2005. 5.1, 5.3.1, 5.4

[114] M. S. Chae, W. Liu, and M. Sivaprakasam, "Design optimization for integrated neural recording systems," *IEEE J. Solid-State Circuits*, vol. 43, no. 9, pp. 1931–1939, 2008. 5.1, 5.3.1, 5.4

[115] J. Max, "Quantizing for minimum distortion," *IEEE Trans. Inform. Theory*, vol. 6, no. 1, pp. 7–12, Mar. 1960. 5.3

[116] V. Karkare, S. Gibson, and D. Marković, "A 130-μW, 64-Channel Spike-Sorting DSP Chip," in *IEEE Asian Solid-State Circuits Conf.*, Taipei, Taiwan, 2009, pp. 289–292. 6.1, 6.2, 6.4

[117] ——, "A 130-W, 64-Channel Neural Spike-Sorting DSP Chip," *Solid-State Circuits, IEEE*, vol. 46, no. 5, pp. 1214–1222, 2011. 6.1, 6.2

[118] T.-C. Chen and L.-G. Chen, "128-Channel Spike Sorting Processor With a Parallel-Folding Structure in 90Nm Process," *IEEE Int. Symp. Circuits and Syst. (ISCAS 2009)*, pp. 1253–1256, May 2009. 6.1

[119] T.-C. Chen, K. Chen, Z. Yang, K. Cockerham, and W. Liu, "A biomedical multi-processor SoC for closed-loop neuroprosthetic applications," in *IEEE Int. Solid-State Circuits Conf.*, San Francisco, CA, 2009, pp. 434–435, 435a. 6.2

[120] D. Marković and R. W. Brodersen, *DSP Architecture Design Essentials.* Springer, 2012. 7.1.2

[121] "ROACH," 2011. [Online]. Available: https://casper.berkeley.edu/wiki/ROACH 7.2, 7.3.2

[122] H. K.-H. So, "BORPH: Berkeley Operating system for ReProgrammable Hardware," 2010. [Online]. Available: http://www.eee.hku.hk/~{}hso/borph.html 7.3.2

[123] "KATCP," 2012. [Online]. Available: https://casper.berkeley.edu/wiki/KATCP 7.3.2

[124] "BEE_XPS," 2009. [Online]. Available: https://casper.berkeley.edu/wiki/BEE_XPS 7.3.3