Highlights

Adaptive spike detection and hardware optimization towards autonomous, high-channel-count BMIs

Zheng Zhang,Timothy G. Constandinou

- Adaptive spike detection method for autonomous operation.
- Avoids the need for training data or threshold re-calibration.
- Low complexity hardware-efficient design for real-time, low power applications.
- Low memory requirements. No floating-point, multiplication or division operations.
- Desirable for chronic or high channel recordings where signal fidelity may vary.

Adaptive spike detection and hardware optimization towards autonomous, high-channel-count BMIs

Zheng Zhang[a], Timothy G. Constandinou[a,b]

[a]*Department of Electrical and Electronic Engineering, Imperial College London, South Kensington Campus, London SW7 2AZ, UK*
[b]*UK Dementia Research Institute (UKDRI) Care Research & Technology Centre, based at Imperial College London and the University of Surrey*

ARTICLE INFO

Keywords:
Spike Detection
Adaptive Threshold
Multi-unit Activity
Hardware-Efficient
Low Power
Embedded System
Neural Interface

ABSTRACT

Background: The progress in microtechnology has enabled an exponential trend in the number of neurons that can be simultaneously recorded. The data bandwidth requirement is however increasing with channel count. The vast majority of experimental work involving electrophysiology stores the raw data and then processes this offline; to detect the underlying spike events. Emerging applications however require new methods for local, real-time processing. New Methods: We have developed an adaptive, low complexity spike detection algorithm that combines three novel components for: (1) removing the local field potentials; (2) enhancing the signal-to-noise ratio; and (3) computing an adaptive threshold. The proposed algorithm has been optimised for hardware implementation (i.e. minimising computations, translating to a fixed-point implementation), and demonstrated on low-power embedded targets. *Main results:* The algorithm has been validated on both synthetic datasets and real recordings yielding a detection sensitivity of up to 90%. The initial hardware implementation using an off-the-shelf embedded platform demonstrated a memory requirement of less than 0.1 kb ROM and 3 kb program flash, consuming an average power of 130 μW. *Comparison with Existing Methods:* The method presented has the advantages over other approaches, that it allows spike events to be robustly detected in real-time from neural activity in a completely autonomous way, without the need for any calibration, and can be implemented with low hardware resources. *Conclusion:* The proposed method can detect spikes effectively and adaptively. It alleviates the need for re-calibration, which is critical towards achieving a viable BMI, and more so with future 'high bandwidth' systems' targeting 1000s of channels.

1. Introduction

With the advent of *high bandwidth* Brain Machine Interfaces (BMIs) [33, 35, 37] and new and emerging research tools for electrophysiology [22, 10, 1, 40], implantable neural interfaces are now capable of recording from 100s to 1000s of channels. This has been made possible by advances in microsystems technology that also brings the opportunity of large scale integration with microelectronics [31]. This increase in channel count however poses a major challenge in communication power for both wired and wireless systems. The availability of front-end electronics here allows for neural signal conditioning and processing that can facilitate feature extraction. This can lead to massive reductions in communication bandwidth and thus overall power requirements, but requires the on-node processing to be achievable in low power. It is also essential that any data reduction or feature extraction (e.g. [39]) maintains the underlying information in the neural signal.

A common approach is to select a specific signal band, for example, 0.1–100 Hz for Local Field Potentials (LFPs) or 300 Hz–5 kHz for Extracellular Action Potentials (EAPs) and optimise the amplifier performance as well as data converter sampling rate and resolution to reduce the output bit rate [4, 5, 15]. For EAPs further reduction is possible due to the sparse nature of neural spiking signals. Single Unit Activity (SUA), Multi Unit Activity (MUA), or Entire Spiking Activity (ESA) are three means of reducing a raw recording to a spike rate, or event-based output. This can reduce output bandwidth by to 2-3 orders of magnitude [14, 32, 15]. One of the simplest such schemes is to do spike detection, and transmit spike timings (for MUA [20, 30, 11, 15]) or spike snippets (for subsequent SUA processing, e.g. spike sorting [19, 24, 39, 32]).

Although the spike detection itself is a relatively simple operation (i.e. a comparison to a threshold), its overall performance (sensitivity, specificity, selectivity and false positive rate) [6] can be significantly improved by pre-processing the signal. This pre-processing includes filtering, to ensure complete removal of lower frequency LFPs; spike enhancement, to boost the Signal-to-Noise Ratio (SNR); and threshold computation, to ensure a good balance between noise being incorrectly detected (false positives), and spikes being missed (false negatives).

The filtering often starts in the analogue domain, with the front-end providing a high pass operation. This typically consists of a low order filter with cut-off frequency below the band of interest, and thus avoid introducing phase distortion (that would change the shape of the spike) [41, 5, 36]. This is typically proceeded with a (causal) digital filter, typically a finite impulse response (FIR) to remove any remaining lower frequency activity or noise. Then the spike enhancement that emphasizes spikes in preference to noise can be achieved using a wavelet transform [52, 51, 26], template matching [2, 27, 50, 21, 32] or non-linear function such as the Non-linear Energy Operator (NEO) also known as the Teager/Kaiser Energy Operator (TEO) [23] and variations

zheng.zhang18@imperial.ac.uk (Z. Zhang); t.constandinou@imperial.ac.uk (T.G. Constandinou)
ORCID(s):

[34, 9, 14].

Before comparing this emphasised/enhanced signal to the threshold it is essential to set its level based on the specific recording channel. This is because the individual electrode properties, tissue environment (e.g. density of surrounding neurons, proximity to electrode, scar tissue growth), in addition to electronic properties (e.g. instrumentation noise) varies for each channel, and over time, resulting in variable signal and noise levels. The threshold is thus calculated based on the statistics of the underlying signal, typically using the mean, median or standard deviation [42]. The cross-correlation of NEO-processed data is also used in [34], which achieves a good spike detection performance. However, such a threshold tends to be unstable since the appearance of spikes can disturb the local statistics, and in [13] a spike elimination technique has been proposed to reduce such interference. Additionally, for a real-time system, the threshold needs to be pre-calculated using training data, or ideally even being adaptive to track changes in the signal statistics. Adaptive threshold methods implemented in hardware have used local minima, maxima or peak detection [7, 28], Sigma-Delta estimation on half-normal distribution [17], the wavelet transform [52, 51], or estimations of root mean square (rms) [48] and/or the standard deviation [20].

This work presents a novel, highly hardware efficient (low complexity, low computation) adaptive spike detection algorithm for implantable BMI applications. This includes a mean subtraction filter to first eliminate any lower frequency components (e.g. LFP) from the signal without introducing additional phase distortion. We also propose a novel operator, the Amplitude Slope Operator (ASO) as a hardware-efficient alternative to NEO for enhancing the SNR of the EAP signals. The adaptive threshold is calculated periodically by taking a running mean that excludes any detected spikes that have been detected, but also runs a concurrent subthreshold detection to exclude a portion of the undetected spikes within the background activity. The algorithm has originally been developed in Matlab using floating point arithmetic, and has been ported to a C implementation using fixed-point arithmetic. This has been applied to two embedded targets (ARM Cortex M0+ and ARM Cortex M4 microcontrollers) to demonstrate the real-time capability, spike detection performance and low power/low complexity implementation.

The remainder of this paper is organised as follows: Section 2 describes the methods, algorithm and implementation; Section 3 present and discusses both the simulated and measured results; and Section 4 concludes this work.

2. Material and Methods

The algorithm for adaptive spike detection has been developed in three phases: (1) initial methods conceived, tested and optimised using MATLAB with floating point arithmetic; (2) then translated to a fixed point representation; and (3) migrated to an embedded target (two different microcontrollers within the ARM Cortex family) using C programming language.

The workflow of the proposed spike detection system is illustrated in Fig. 1.

Evaluating spike detection algorithms has the challenge in that real data has no ground truth. One can either use synthetic or real data to evaluate their algorithms. The great advantage with using synthetic data, is that the ground truth is known. The noise level, however, and any drifting/fading of the signal does not behave in a realistic way. Additionally, noise characteristics across different electrodes and instrumentation vary, and also within probes or multielectrode arrays across recording channels. It is therefore challenging to utilise established synthetic datasets and achieve results that are representative of a real recording. Often there can be some inconsistency between the evaluation results and practical performance. On the other hand, real data provides realistic signals but the ground truth is unknown. To balance the trade-off between these approaches (integrity of a realistic signal vs. integrity in the evaluation), we use both types of datasets.Synthetic data [42] is first used to quantitatively evaluate our proposed algorithm, and make a fair comparison with different algorithms that have been previously published. Experimental recordings (with an annotated ground truth) [46] is then used for demonstrating the practical performance of the proposed algorithm with realistic noise levels and signal quality.

2.1. Test Datasets

2.1.1. Synthetic dataset

For the initial evaluation, and quantitative comparison, we used a commonly used dataset with known ground truth provided by Quian Quiroga et al. [42]. This dataset contains 4 groups of synthetic recordings (easy1, easy2, difficult1, difficult2), with noise levels ranging between 0.05 to 0.2. Each synthetic recording is 60 s in duration and sampled at 24,000 Hz. Across each synthetic recording, there are three different single units, each with a mean firing rate of 20 Hz, whose arrival follows a Poisson distribution. Fig.2 shows one portion of the signal. The validation on such dataset is included in Section 3 as a baseline comparison.

2.1.2. Experimental data

The neural recordings were collected by Cortex Lab, University College London and made publicly available [46]. These recordings contain 384 channels of neural signals from the visual cortex, sampled at 30 kHz. We have chosen to develop and validate our algorithm using this real data such that the signal dynamics and channel-to-channel variability are as realistic as possible including broadband data (both local field potential and extracellular action potentials). Previous work in the literature has used synthetic spike data [34, 42, 52] to ensure the ground truth is known. This is however not the case for real-world data, i.e. the ground truth is not certain. We therefore applied an automatic spike labelling method to estimate the ground truth of spikes in this experimental dataset. We achieved this first by labelling the real recordings with a “gold standard” spike detection/sorting tool "Wave_Clus" [8] treating each channel independently. We

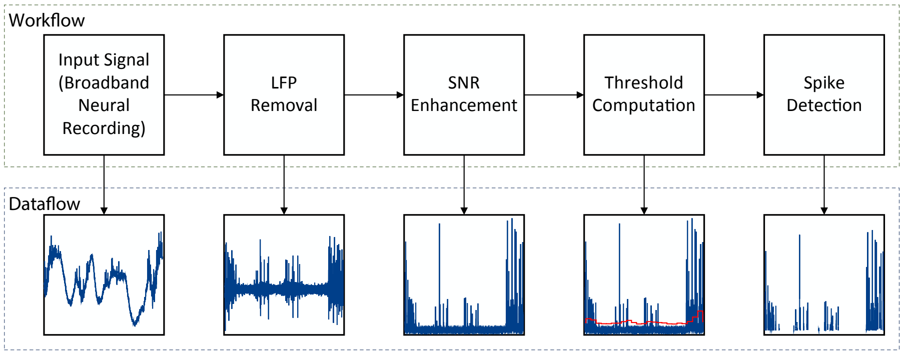

Figure 1: Overview of the proposed adaptive spike detection algorithm.

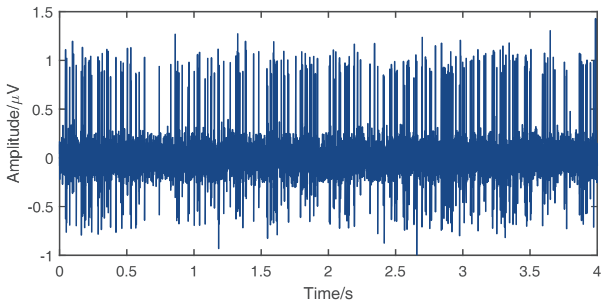

Figure 2: A Snapshot of the synthetic dataset

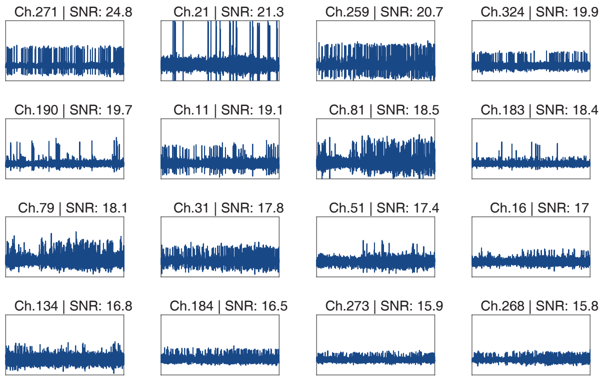

Figure 3: A sample of 16 channels shown in descending order of SNR. Each waveform is annotated with channel number and SNR level.

then use fact that the neuropixels probes provide spatially-oversampled observations, with spikes events appearing across adjacent electrodes. By exploiting this cross-channel information, any missing spikes can be recovered and isolated events removed, to achieve good integrity for the ground truth estimation. Moreover, the detected spikes can either be genuine spikes or MUA noise. We then merge the different spike type clusters into two clusters, i.e. real spikes and MUA noise, using the fact that these two clusters should have the max spike peak difference. This resulted in a dataset with 65 channels of recorded data for 33.3s each, annotated with ground truth estimations. The variety of the data has also been assessed in which the spike peak range is varying from over 300 μV to 50 μV and the noise ground range is varying from 50 μV to below 20 μV, while the SNR is varying from 8 dB to 19dB. A snapshot of some recordings are given in Fig. 3.

2.1.3. *Realistic data with varying noise levels*

Although the recordings already contain realistic noise levels, we add further noise to rigorously evaluate the robustness of our proposed methods. Specifically, two types of noise are added (for SNRs between 5 dB to 20 dB) to simulate different noise environments. Firstly Gaussian noise is added to represent the variability in thermal noise observed in recording electrodes, instrumentation and other natural processes. Secondly, background activity (also referred to as multi-unit activity (MUA) noise) is added to represent the variability in the 'undetectable' signals originating from the tissue itself. We follow the simulation methods described in [42, 44] that uses a spike arrival time following a Poisson Distribution, with average frequency of around 50 spikes/sec. The arrival interval duration between two spikes, therefore, follows an exponential distribution with an average interval length for 20 ms/spike. Noise spike samples are extracted from the recordings different from the ones used for spike detection to ensure independence. The spike peaks are normalised according to SNR levels before being added to the recordings.

2.2. Algorithm Development

The algorithm was initially developed using floating point arithmetic using MATLAB 9.5 (R2018b).

The proposed algorithm has been designed to operate three phases: (1) *mean subtraction*, to remove the LFP signal; (2) *signal enhancement*, to make the spikes more distinguishable compared to the noise, and (3) *threshold comparison*, to detect the occurrence of a spike. The block diagram of the proposed algorithm is given in Fig. 4.

2.2.1. *Mean Subtraction*

The aim of the mean subtraction is to remove the LFP – the low-frequency 'background noise' in signals observed due to

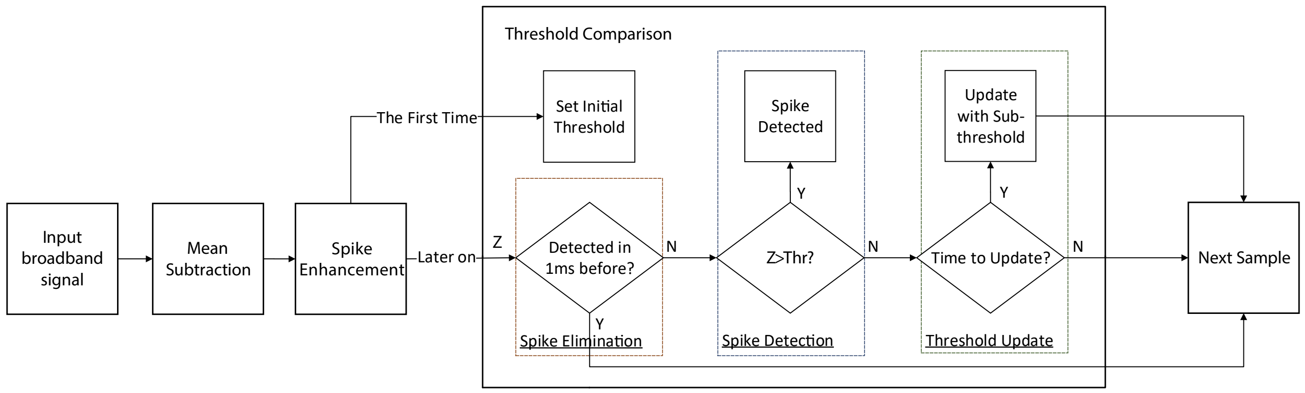

Figure 4: Block diagram of proposed algorithm.

the aggregated network activity across the tissue. The LFP mainly lies at frequencies below 100 Hz. To remove this, we propose using a 16-point moving average filter to find the local mean and subtract this mean from the current value to align the signal.

Instead of calculating the mean by summing a series of successive samples, we achieve this by incrementally updating a weighted average as described in Eq. 1. This approach both simplifies the required computation and reduces memory requirements. The result of the LFP removed signal is obtained by Eq. 2.

$$\mu_n = \mu_{n-1} - \frac{x_{n-16} - x_n}{16} \tag{1}$$

$$y_n = x_n \; - \; \mu_{n-1} \tag{2}$$

where μ_n is the n^{th} mean of the signal, x_n is the n^{th} read value and y_n is the mean-removed x_n. With this design, the summing operation in a filtering operation can then be replaced with 2 additions (or subtractions). The number of required operations is therefore reduced by 8×. The complexity is reduced from $O(N)$ to $O(1)$, which means that using a longer filter does not increase computation. The captured mean and resulting signals are shown in Fig. 5

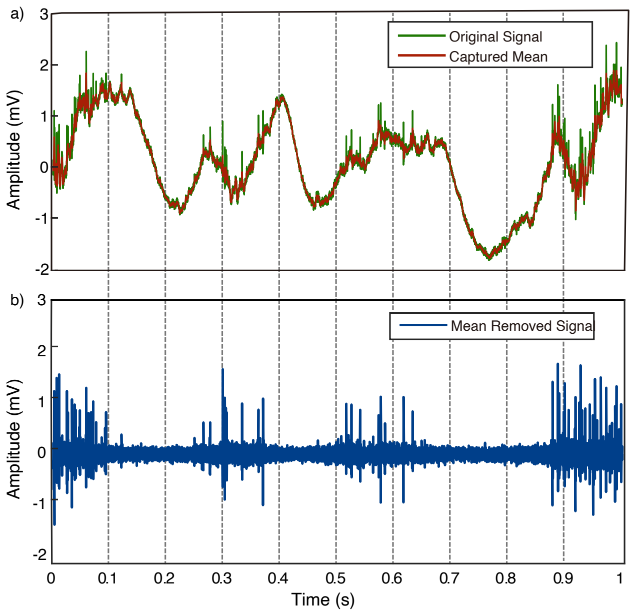

Figure 5: Results of mean subtraction filter; after the mean subtraction, the recording is aligned at zero. Shown are: a) Original recording with the captured Mean value. b) Signal after the LFP removing.

2.2.2. Spike Enhancement

Inspired by the MNEO [9], described by Eq. 3, we propose a new operator, the Amplitude Slope Operator (ASO), formulated in Eq. 4

$$z_n = {y_n}^2 - y_{n+k}y_{n-k} \tag{3}$$

$$z_n = y_n\,(y_n - y_{n-k}) \tag{4}$$

where z_n is the emphasised data and y_n is the LFP-removed input data. In this equation, y_n stands for the amplitude and $y_n{-}y_{n-k}$ stands for the slope at this point. Referring to Fig. 6, a spike is different from noise because it has a higher amplitude and a more significant slope or gradient, i.e. abrupt changes. Therefore, intuitively the ASO amplifies the signal intervals that satisfy both these conditions whilst suppress signal intervals where only one or neither of these conditions are satisfied.

Each ASO operation requires one multiplication and one subtraction. The computation is reduced by half, compared to the widely used non-linear energy operator (NEO) [12, 14, 28], since this requires two multiplication operations and one subtraction. Additionally, no future sample is needed in this operator, which makes it suitable for real-time applications. The results are shown in Fig. 7.

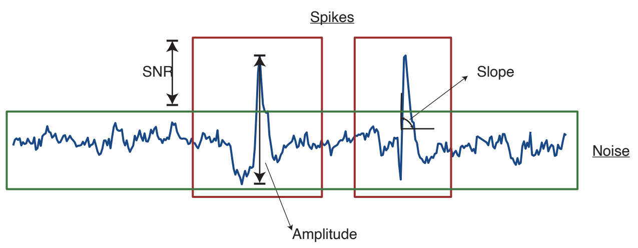

Figure 6: The differences of spikes and noise, where the spikes are likely to have more distinguishable amplitudes and slopes.

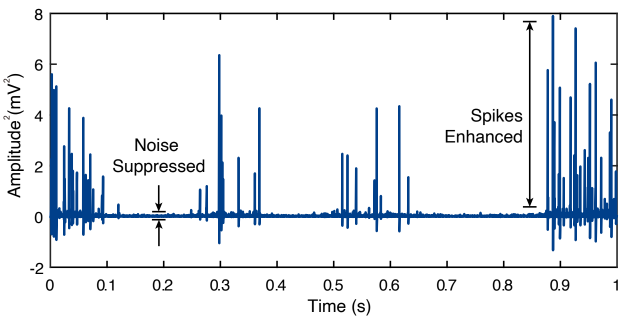

Figure 7: The ASO emphasised signal, where the noise is suppressed and the spikes are enhanced, comparing to Fig. 5.b

2.2.3. *Threshold Comparison*

The spike detection step itself is performed by comparing the emphasised signal to a threshold value. This threshold should be determined by considering the local statistics, providing a measure of the local noise level. A spike is detected when the input signal crosses this threshold value.

There are several challenges in computing this threshold value, particularly for real-time hardware application. Firstly, the inherent need for statistics in computing the threshold places significant memory requirements in a low power/low complexity implementation. Secondly, the fact that each recording electrode observes its own unique SNR means each channel needs to be individually calibrated or trained. Thirdly, the signal observed at any given electrode itself changes over time requiring repeated re-calibration. These challenges motivate the development of an adaptive threshold method that using an algorithmic approach to avoid the need for statistics.

Developing a iterative function that estimates the noise level without requiring prior history poses its own challenges. Samples that contain spikes for example can contribute to erroneously raising the threshold value. It is thus essential to provide blanking to capture a robust noise level.

The overall threshold comparison process is described in Fig. 4. The block labelled *'Spike Detected'* reports the detected spike locations, and details of the block labelled *'Update with Sub-threshold'* are shown in Fig. 8.

The algorithm operates as follows: an initial threshold value is set by calculating the median value across the first 64 samples. Using the median is essential here because the buffered values are likely to contain spikes, and this provides some robustness to outliers. We compute the initial threshold by adapting the method reported in [42], with threshold

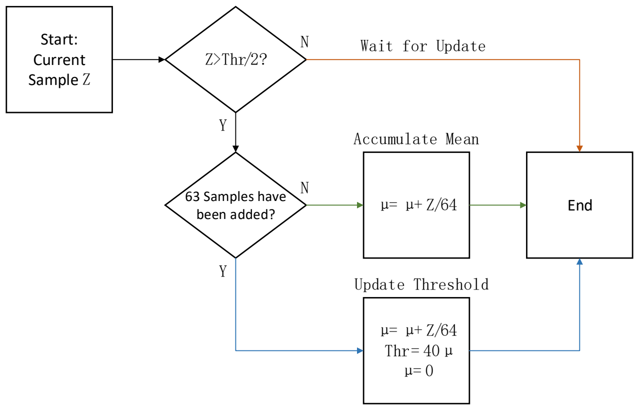

Figure 8: Block diagram of the update algorithm

described by Eq. 5. This has been modified (multiplier increased to ×22) to operate on the enhanced/emphasized signal (after ASO pre-processor) instead of raw data.

$$Thr = 22\sigma, \quad \sigma = median[\frac{y(n)}{0.6745}] \tag{5}$$

Although the median is essential for initially setting the threshold, the trade-off between accuracy and complexity needs to be considered for the operation thereafter (i.e. regular threshold update). Considering the median, mean and standard deviation measures, the mean is selected for the regular threshold update. This is performed by taking an average across a 64-sample window, as described in Eqs. 6 and 7.

$$\mu_{thr} = \frac{1}{64}\sum_{i=n-64}^{n-1} z_i \tag{6}$$

$$Thr = 40 * \mu_{thr} \tag{7}$$

The 40× multiplier has been determined through empirical test (i.e. an exhaustive search) also considering hardware implementability. This revealed that although the value selected was the optimum, any variability to this provided good robustness (i.e. low sensitivity to data dependence). The threshold update is duty cycled to occur every 0.6 s, striking a trade-off between the 'attack' rate and average power requirements. This duty cycle can potentially be tuned to be significantly lower (as signals do not generally drift over seconds, but more on the scale of hours or days).

Since using the signal mean to determine the threshold value is sensitive to outliers, e.g. the presence of spikes would add error to the noise estimation, we implement two techniques to eliminate the effect of spikes on the threshold computation:

1. *Spike Exclusion*: This effectively blanks the samples associated to any detected spikes when calculating the local mean. More specifically, this is implemented by

invoking a 1 ms 'refractory period' whenever a spike is detected (it is observed that the vast majority of spikes have a positive phase of approximately 0.7 ms). This disables any updates to the threshold computation during this period and as such excludes the spike samples from the mean computation. Implementing this feature additionally avoids multiple detections of a single spike (e.g. if the positive phase of the spike is noisy).

2. *Sub-threshold Exclusion*: For spikes that are not detected but clearly above the 'noise' level, the previous *spike exclusion* step does not exclude this from the mean computation. Therefore, a separate spike detection is performed using half the threshold value to further exclude any distinct spike signals. This has been adopted in preference to reducing the spike detection threshold level to maintain a good balance between overall sensitivity and accuracy

2.3. Fixed-Point Implementation

To improve the efficiency of hardware implementation (i.e. reducing complexity and power consumption), a fixed point representation is highly desirable.

To convert the previously described algorithm that is originally developed in MATLAB, all floating point variables are replaced with 16-bit integer variables. This furthermore provides the opportunity to replace multiplication (and division) operations to logical bit shifts wherever scaling ratios are appropriate. The reduced precision in the arithmetic however poses some issues that need to be addressed.

2.3.1. Mean Subtraction

The fixed point conversion first impacts the mean subtraction operation (to filter out LFP signal), where the reduction in precision leads to the mean value fading to a relatively small value that can lead to a significant error. This can be resolved by regularly recalculating the mean such that its precision is restored before it is allowed to 'fade'. Given the 16-bit precision, the recalculation interval is found to be 40 ms (determined empirically).

However, as the moving average filter is based on a 16-sample sliding window, its implementation can be simplified by replacing the division operation by a 4-bit logical right shift operation.

2.3.2. Amplitude Slope Operator

The ASO function can also be simplified in a similar manner to use a logical shift. The ASO operator is thus modified to:

$$z_n = (y_n - y_{n-1}) << \varphi(y_n) \tag{8}$$

where φ refers to the operator that finds the last power of 2 and $<<$ stands for bit left-shifting (A smaller number is used in preventing bit overflowing.) Therefore, the need for the multiplication can be altogether eliminated.

2.3.3. Threshold comparison

The computation of the initial threshold is modified to account for the fixed point representation, described in Eq. 9.

$$thr_{init} = median[y(n)] << 5 \tag{9}$$

The threshold value here is rounded down to 32 times of the local median value (22 / 0.6745 ≈ 32), $implementedusinga5-bitleft-shiftofthemedian.$

The updated threshold value (during detection process) is then calculated according to Eq. 10.

$$thr = \mu_{thr} << 5 + \mu_{thr} << 3 \tag{10}$$

For the mean computation required for the *sub-threshold elimination*, we reverse the order of operation (i.e. first right shift by 6-bits and then accumulate) to avoid the possibility of an overflow given the 16-bit representation.

The algorithm was re-evaluated after all the above mentioned changes, demonstrating a negligible degradation to overall performance.

2.4. Hardware Implementation

A key objective in the hardware implementation of a spike detection algorithm is to minimise its power requirements. For an implantable application (e.g. BMI) this is essential for two reasons: firstly, the energy budget is constrained by the energy source (e.g. battery) and wireless power transfer efficiency. Secondly, any power consumed is ultimately dissipated as heat that can lead to damage of neural tissue. Additionally it is highly desirable to reduce complexity to allow for a high integration density.

2.4.1. Hardware and Measurement Setting

To assess the hardware efficiency of the proposed algorithm we implement the algorithm on a commercially-available embedded platform. Here, the power consumption can provide a relative measure of computational complexity, whereas the memory requirements can provide a measure of hardware complexity (i.e. silicon area).

We have implemented the proposed algorithm on two different microcontroller families (ARM Cortex M0+ and ARM Cortex M4), using appropriate development platforms (FRDM - KL05Z and FRDM - K64F respectively), for the power consumption, memory requirement, and run times measurement of the proposed algorithm. The FRDM - KL05Z platform features a 32-bit ARM - M0 + CPU operating at 48 MHz with 32 KB flash and 4 KB RAM. It has one 12 - bit ADC and one 12 - bit DAC. This MCU is designed with efficiency in mind, which is ideal as an ultra-low power demonstrator. The FRDM - K64F platform features a 32 - bit ARM - M4 CPU operating at 120 MHz. It has 1 MB flash memory, 256 KB ROM, two 16 - bit ADCs and one 12 - bit DAC and it also supports low power timer. This board is a powerful MCU with low power consumption.

However, since the development boards only contain one digitial to analogue converter (DAC), only one signal can be directly observed at a time (ASO output and threshold value). We therefore have additionally implemented a pulse width modulated (PWM) based DAC to enable a second analogue output to be concurrently observed.

Since the typical frequency range of observed spikes (measured extracellular action potentials) is 300 Hz to 3 kHz [36], the sampling frequency is set to be 7 kHz to avoid aliasing and keep the power consumption as low as possible. For spike sorting application however a higher sampling frequency would be desirable.

The firmware has been programmed in C using the MBED OS online compiler development environment. All variables are defined using either unsigned short or short data types (16-bit integers). The test data has been transferred from MATLAB to an arbitrary waveform generator, with output observed using an oscilloscope. The power consumption is determined by measuring the voltage V_{op} drop across a 10 Ω current sense resistor(R) that is placed in series the MCU power line.The runtime t_{op} is observed using the oscilloscope. The incremental power W_{op} (beyond the idle running power W_{idle}) and extra energy E_{op} for each operation, average power W_{avg} across the duty cycle, and total energy E_{sample} per sample has been calculated based on Equ. 11-14. The memory utilisation is directly reported within the MBED development environment.

$$W_{op} = \frac{V_{op}^2}{R} - W_{idle} \tag{11}$$

$$E_{op} = W_{op} * t_{op} \tag{12}$$

$$W_{avg} = \frac{\sum_{all\ ops} E_{op} * N_{op}}{\sum_{all\ ops} t_{op} * N_{op}} \tag{13}$$

$$E_{total} = \frac{\sum_{all\ ops} E_{op} * N_{op}}{N_{samples}} \tag{14}$$

Where N_{op} stands for the number of times one operation should be taken in one duty cycle and $N_{samples}$ stands for the number of samples been taken in one duty cycle.

2.4.2. Algorithm Migration

The algorithm itself has been developed with computational complexity and hardware efficiency in mind. The previous section focused on translating this to fixed point arithmetic to further reduce complexity. One addition consideration however is to achieve an approximately constant computational load (operations per sample) to ensure efficient hardware utilisation (e.g. clock optimisation).

Although the threshold update itself happens once each 0.6 s, it is essential to spread the processing operations required throughout this 0.6 s period. The threshold update process achieves this by interleaving operations during each sample period. This is implemented by executing one of 3 different branches (shown previously in Fig. 8): 'Wait for update', 'Accumulate mean' and 'Update threshold'. This approach both alleviates the need for loops and additionally reduces memory requirements.

The subroutine for detection (mean subtraction, spike enhancement and threshold comparison) is invoked at the sampling rate (i.e. 7 kHz), controlled by a timer (low power timer in K64F). The MCU is set to a low power sleep mode between samples and woken up on each new sample. The timing for the threshold comparison duty cycle and spike elimination are achieved by polling. This is to avoid using additional timers. The threshold is then updated every 3,500 samples (7,000×18,000 /30,000 = 4,200), and 5 samples (20×7,000 /30,000 ≈5) will be skipped after a spike is detected for applying spike elimination.

The ASO implementation is different on each of the MCUs. This is because the K64F has a dedicated assembly instruction (Count Leading Zeros, CLZ) that can be leveraged to determine the last power of 2. This reduces the multiplication and division operations to bit shifting. The KL05Z however, does not support CLZ so needs to be implemented in software, therefore requiring more instructions. The multiplication therefore remains in the KL05Z implementation with ASO calculated according to Eq. 4.

The threshold comparison is implemented identically to the MATLAB implementation.

3. Results and Discussion

This section presents results demonstrating the operation of the algorithm, outright spike detection performance (e.g. sensitivity, accuracy, FDR), and hardware efficiency. The power consumption, run time, and memory requirement for the hardware are then measured to demonstrate the suitability for implantable BMI applications.

3.1. Software Evaluation

To show the effectiveness of the proposed algorithms, We have first tested our proposed algorithm on a synthetic dataset as a baseline comparison. We then compare the proposed algorithm with an IIR filter for LFP removal, NEO in spike emphasising, and different thresholding methods using a real dataset to demonstrate the suitability of the proposed algorithm in practice. The improvement is noticeable. A test on varying noise levels has also been carried on to show the adaptiveness of this algorithm.

3.1.1. Evaluation Metrics

The typical evaluation metrics of spike detection performance are detection accuracy (Acc), sensitivity (Sens) and false detection rate (FDR), which are calculated according to the equations below:

$$Sens = \frac{TP}{TP + FN} \tag{15}$$

$$FDR = \frac{FP}{TP + FP} \tag{16}$$

$$Acc = \frac{TP}{TP + FP + FN} \tag{17}$$

where TP stands for true positive, successfully detected spikes. FP stands for false positive, the locations where the signal is wrongly detected as spikes. FN stands for false negative, the undetected spikes.

A high sensitivity means most of the spikes are detected while low FDR means only a few of the detected spikes are incorrect. The accuracy provides an overall measure that represents the trade-off between sensitivity and FDR. For applications that further process the data after spike detection, e.g. spike sorting, a high sensitivity is desirable (at the expense of FDR) because, any incorrectly detected spikes can be removed after subsequent classification. For a low sensitivity however, undetected spikes have no way to be recovered.

3.1.2. Testing with synthetic data

Four types of signal named "Easy1", "Easy2", "Difficult1", "Difficult2" with noise levels 0.05, 0.1, 0.15, 0.2 (16 test signals in total) have been used for performance testing. The results of the floating and fixed-point implementation are given in Table. 1 (Only Easy1 and Difficult1 are included for conciseness). We have also compared our results with [42, 54, 43, 18]. Spike detection performances are similar (Acc ranging from 90% to 99%). We provide results for one of these datasets in Table. 1 (other datasets excluded for conciseness). We however calculate the average scores for each of the different SNR level across all four dataset, with results shown in Fig.9. This clearly shows how the proposed algorithm is robust to different noise levels, with only a minor degradation in performance with increasing noise levels (for this synthetic data). In [42], the scores could overfit to some noise levels and behave worse in other cases (especially in 0.2 noise level). The performance variance is much smaller in our implementation. From the table, one can notice that the average performance is similar between the float-point implementation. and the detection method in [42]. For the fixed-point implementation, the sensitivity only degrades for 1% and the FDR is increased for 5%. Such minor scarification gives us the power reduction of 2/3 as it will be demonstrated in later section.

3.1.3. Mean Subtraction vs. High-Pass IIR Filters

It has been shown that in [41, 36, 5], causal IIR filters will cause phase distortion, i.e., the phase response of the IIR filter is nonlinear in frequency, which can change the shape of the spikes and reduce the detection accuracy.

However, with the mean subtraction, the phase response of the corresponding filter is approximately linear. A comparison between the phase response of the proposed mean subtraction filter and a 2-pole causal Butterworth filter used

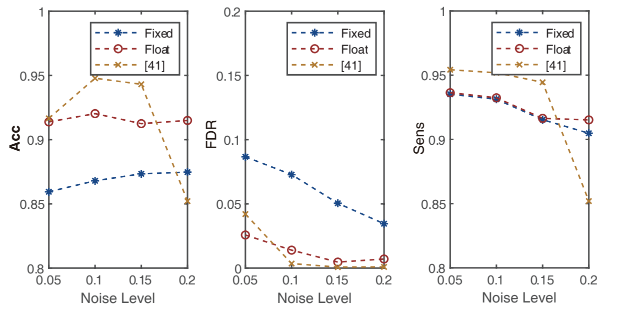

Figure 9: A comparison of different algorithms on four noise levels

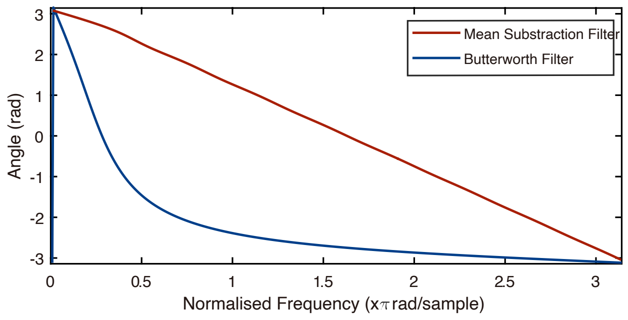

Figure 10: Phase response of the proposed mean subtraction filter and a second order Butterworth filter, where the phase response of the proposed filter has significantly better linearity than that of the Butterworth filter.

in [36] is shown in Fig. 10. From the filtered results in Fig. 11, it can be observed that by using the proposed filter, the spike shape is preserved in addition to the peak value, which is highly beneficial to spike detection. A comparison of the detection performance is shown in Table 2.

The spike detection performance using each filter type is given in Table 2. Here it can be observed that the proposed LFP removal technique can outperform an IIR filter implementation, in particular the FDR, which is improved by some 6 %. These results include the averaged performance across all 65 channels, and also with the 10 ‘worst’ channels excluded. The 10 channels that are excluded contain less detectable single unit activities and as such contain mainly background (i.e. multi-unit) activity, therefore having a significantly reduced SNR (below 11 dB) and detection accuracy (below 50 %).

3.1.4. SNR enhancement

The proposed SNR enhancement function (ASO) is compared to a commonly used energy operator (NEO) to assess its suitability as a pre-processing step for spike detection. A sample spike recording is shown in Fig. 12 including both the ASO and NEO processed signals. It can be observed that the ASO function produces higher peak values in comparison to the NEO. The overall performance metrics are provided in Table 3

It can be observed that a higher sensitivity can be achieved using ASO, however this is at the cost of also a higher FDR.

Table 1
Performance of spike detection on synthetic dataset

	Dataset	Easy1				Difficult1				Average
	Noise Level	0.05	0.1	0.15	0.2	0.05	0.1	0.15	0.2	-
Fixed-point Implementation	Acc	0.87	0.86	0.82	0.87	0.81	0.86	0.92	0.92	0.87
	Sens	0.93	0.94	0.89	0.91	0.93	0.92	0.93	0.95	0.92
	FDR	0.07	0.09	0.09	0.05	0.14	0.08	0.02	0.03	0.06
Float-point Implementation	Acc	0.90	0.93	0.89	0.87	0.92	0.91	0.93	0.94	0.92
	Sens	0.93	0.93	0.89	0.86	0.93	0.93	0.93	0.95	0.93
	FDR	0.03	0.00	0.00	0.01	0.01	0.03	0.00	0.01	0.01
Offline spike detection (reported in [42] for comparison)	Acc	0.94	0.95	0.95	0.90	0.82	0.92	0.91	0.77	0.91
	Sens	0.95	0.96	0.95	0.90	0.95	0.93	0.92	0.77	0.93
	FDR	0.02	0.00	0.00	0.00	0.15	0.01	0.00	0.00	0.01

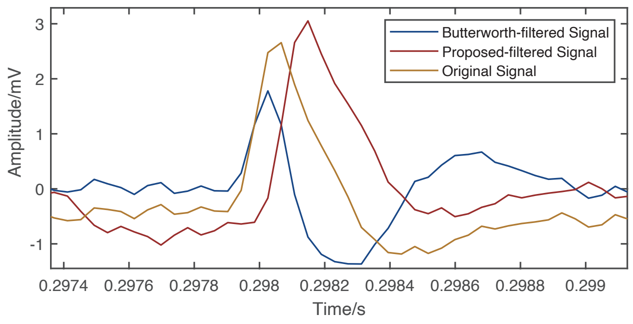

Figure 11: Effect of two different filter types on a spike waveform. This illustrates that the proposed filter does not attenuate the spike peak or distort the spike shape.

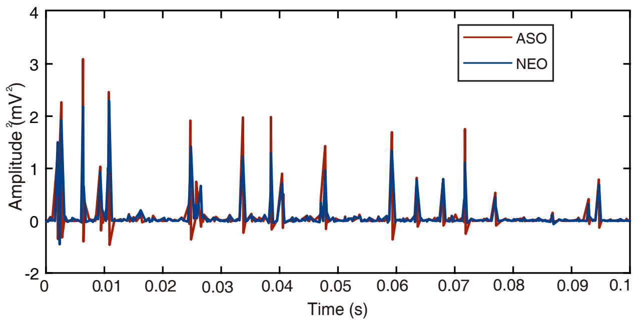

Figure 12: Comparison between the ASO and NEO pre-processor functions for SNR enhancement.

Table 2
Performance comparison between mean subtraction and 2nd-order Butterworth IIR filters for LFP removal

		All (65) Channels	Excluding 10 channels with very low SNR
Butterworth Filter	Acc	0.56	0.66
	Sens	0.81	0.9
	FDR	0.34	0.28
Proposed Filter	Acc	0.69	0.74
	Sens	0.82	0.9
	FDR	0.18	0.16

Table 3
Performance comparison of the using NEO and ASO for SNR enhancement

		All (65) channels	Excluding 10 channels with very low SNR
NEO	Acc	0.68	0.73
	Sens	0.75	0.83
	FDR	0.16	0.14
ASO	Acc	0.69	0.74
	Sens	0.82	0.9
	FDR	0.18	0.16

The two operators are then tested across different noise levels, with results shown in Fig. 13. This shows the spike detection performance for added Gaussian noise and background activity for SNRs between 5 dB and 20 dB. From those plots, it can be observed that the ASO operation consistently outperforms the NEO as noise is added.

3.1.5. Noise estimation

As described previously a basic ‘noise estimation’ is used to define the adaptive threshold level. This is based on a running average, excluding any samples referring to detecting spikes and background activity. This is achieved by applying two specific techniques: spike exclusion (SE) and sub-threshold exclusion (STE).

This process is illustrated in Fig. 14 demonstrating the impact of applying these estimation enhancement techniques. The upper plot here shows the SNR enhanced signal (after ASO pre-processing) and samples that are excluded from the noise estimation by SE and STE. The lower plot shows the noise estimation based alone on ASO pre-processed signal, and with SE and both SE/STE techniques. The detected spike trains are shown below the plot alongside the ground truth data. This qualitatively demonstrates the effectiveness of the noise estimation algorithm as implemented. It is clearly observed that without use of any of the exclusion techniques (SE/STE) the spike detection performance is poor.

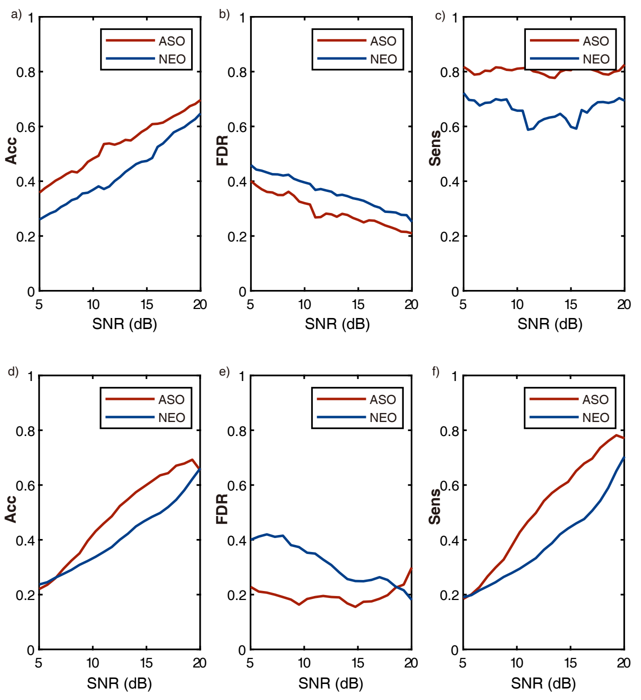

Figure 13: Spike detection performance (Sens, Acc and FDR) of the proposed algorithm using either ASO or NEO functions for input signals with added noise (Gaussian or background activity). Shown are: (top): Performance using ASO and NEO for SNR enhancement for background noise added at a SNR in between 5 dB and 20 dB. a) Acc; b) FDR; c) Sens. (Bottom): Performance using ASO and NEO functions for SNR enhancement for Gaussian noise added at a SNR in between 5 dB and 20 dB d) Acc; e) FDR; f) Sens.

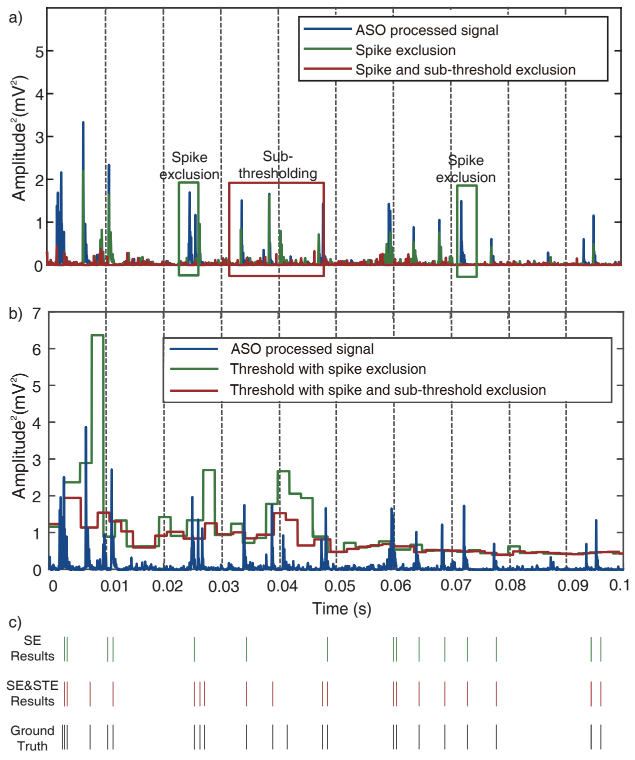

Figure 14: Noise estimation process using the SNR enhanced signal. Shown are: (a) SNR enhanced signal (using ASO function) with excluded samples annotated (i.e. for noise estimation); (b) noise level estimation based on averaging samples corresponding to noise (using different techniques); (c) spike train of detected spikes (using different techniques) including ground truth.

Table 4
Spike detection performance using spike exclusion techniques (SE and STE)

		All (65) Channels	Excluding 10 channels with very low SNR
SE	Acc	0.65	0.7
	Sens	0.78	0.87
	FDR	0.2	0.18
SE&ST	Acc	0.69	0.74
	Sens	0.82	0.9
	FDR	0.18	0.16

The overall performance evaluation is provided in Table 4. This again takes the average results across all 65 channels, and across 55 channels – excluding the noisy data (i.e. 10 channels specifically that not observe any single unit activity). It can be observed that all the performance metrics are improved by applying both SE and STE techniques: 4 % for Acc, 2 % for FDR, and 4 % for Sens on average. This was subsequently tested with added background noise (5 dB SNR), revealing that the sensitivity can remain at around 80 % if both SE and STE are enabled, otherwise falling to 60 % (using SE alone). Both SE and STE are therefore adopted.

3.1.6. Adaptivity

A key aim of this work has been to implement an adaptive threshold. This has mainly been for purposes of avoiding the need for calibration of each individual channel, but can also adapt to changing signal dynamics (e.g. a changing SNR due to fading or new units appearing over time).

The adaptivity to channel specific dynamics has been evaluated by assessing the average performance across the 65 channels (without any calibration or training). Testing for transient changes in SNR is however highly dependent on the experimental method (e.g. electrode type, implantation technique, etc). We have therefore constructed a synthetic dataset that varies the noise level (SNR) to visually observe the adaptation (i.e. qualitatively).

Specifically, we have constructed two test signals that vary the noise level across a 4 s segment. The first test signal adds an increasing Gaussian noise level (SNR decreases from 5dB to -5 dB) whereas the second test signal adds an increasing background activity level (SNR decreases from 20 dB to 5 dB). The results are shown in Figs. 15 and 16.

These tests reveal that adaptive threshold provides some robustness to signals of changing dynamics (e.g. SNR). For the test with added Gaussian noise, the threshold can be seen to gradually increase, to balance a reducing sensitivity with increasing FDR. For the worst case tested (-5 dB SNR), the

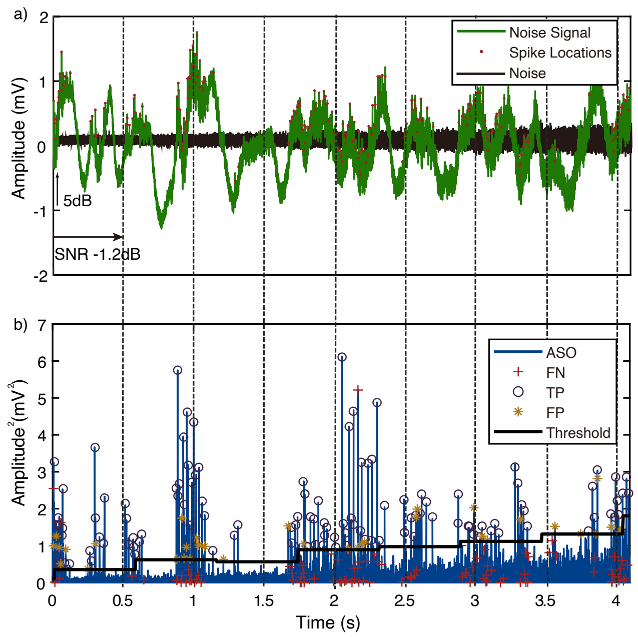

Figure 15: Spike detection on signal with varying level of added Gaussian noise (SNR decreasing from 5 dB to -5 dB). Shown are: a) The original recording with MUA noise and spike locations; b) ASO emphasised signal, the threshold and detection results.

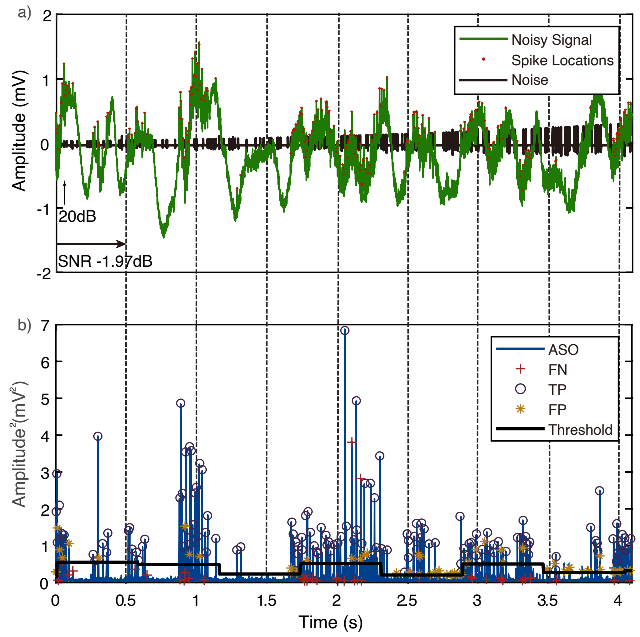

Figure 16: Spike detection on signal with varying level of added background activity (SNR decreasing from 20 dB to 5 dB). Shown are: a) The original recording with added increasing noise level; b) ASO emphasised signal, the threshold and detection results.

sensitivity falls to around 65 % with an FDR of below 15 %. For the test with added background activity, the threshold cannot be observed to change much. Consequently, although the sensitivity remains high at around 88 %, the FDR increases to about 25 %. This is because the background activity starts being detected as spikes.

These results also demonstrate that the empirically determined design parameters do not overfit at certain noise levels, and can generalise well to different noise environments, provided the noise remains at realistic levels.

Table 5
Comparison in spike detection performance between the floating and fixed point implementations

		All (65) Channels	Excluding 10 channels with very low SNR
Fixed Point	Acc	0.67	0.7
	Sens	0.8	0.89
	FDR	0.2	0.19
Float Point	Acc	0.69	0.74
	Sens	0.82	0.9
	FDR	0.18	0.16

3.1.7. Fixed-point Representation

The algorithm has been translated to a fixed-point representation to improve efficiency of hardware implementation (complexity, power). There is however a trade-off with precision in the underlying state variables. This sub-section therefore evaluates the impact of fixed-point representation on spike detection performance, with results provided in Table 5.

This comparison reveals that the degradation in performance due to the fixed-point implementation is relatively minor, with only a 1 % to 4% reduction in each metric. These results also show that the ASO fixed point implementation as described in Eq. 8 can still effectively enhance the SNR (i.e. of spikes over noise) as was originally intended. These results include all features that have been previously described (e.g. using 16-bit integers for all variables, reducing multiplication/division operations to logical bit shifts, etc).

3.2. Hardware Evaluation

To demonstrate the suitability of the proposed algorithm for hardware implementation, we have used an embedded target (microcontroller platform) to measure power consumption, run time and resource utilisation (memory requirements).

3.2.1. Power Consumption and Run-time

These are two key parameters in assessing the efficiency of hardware implementation in an implantable application, where both hardware resource and energy budget and highly constrained. The run-time provides a direct measure of computational complexity, which can be used to estimate the total capacity of a given hardware target, e.g. how many channels can be processed on a single microcontroller.

To improve precision in both energy and timing measurements (taken using an oscilloscope), we repeat each function 500 times, such that an average value for a single operation can be determined. All tests are initially done on the K64F platform (ARM Cortex M4 microcontroller) with the 'final' optimised algorithm then compared to an equivalent implementation on the KL05Z platform (ARM Cortex M0).

- NEO vs. ASO: This is first tested for a floating point implementation - the average power consumption for

Table 6
Average power, energy per operation and run-time of each function for floating and fixed point implementations

Operation		Floating point, K64F			Fixed point, K64F			Fixed point, KL05Z		
		Power (mW)	Energy (pJ)	Time (ns)	Power (mW)	Energy (pJ)	Time (ns)	Power (mW)	Energy (pJ)	Time (ns)
Subtract mean		3.20	509	160	1.17	187	160	0.15	87	578
ASO		0.14	8	60	0.14	14	100	0.15	9	60
Threshold	Wait for update	0	0	54	0	0	43	0	0	124
	Accumulate mean	0.94	134	143	0.14	19	133	0.04	12	343
	Update	1.62	281	174	0.94	147	157	0.21	113	546
Recalculate mean		-	-	-	1.20	759	612	0.08	85	1000
Average/total amount per sample		1.90	519	275	0.75	204	306	0.13	97	769

these functions are 0.87 mW and 0.14 mW respectively (operating at 7 kHz). The run-time for each operation is 60 ns. This corresponds to an energy per SNR enhancement operation of 52.2 pJ and 8.4 pJ. As expected the ASO implementation consumes significantly less power.

- Floating point vs. fixed point: The algorithm is then tested using both a floating and fixed-point implementation. The average power, energy per operation and run-times for each implementation are given in Table 6. One key observation is that the power requirements for ASO operation does not reduce for the fixed point implementation, in fact the run-time increases. This is due to the availability of a hardware multiplication in the ARM Cortex-M4. The average power consumption of the floating-point implementation is 1.9 mW (519pJ per sample), compared to the fixed-point implementation consuming only 0.75 mW (204pJ per sample). The run-time is slightly increased in going from floating point to fixed point implementation from 275 ns to 304 ns (per sample). This would allow for over 450 channels to be implementable using a single microcontroller with 7 kHz sampling rate.
- K64F (ARM Cortex M4) vs. KL05Z (ARM Cortex M0+): The same algorithm is then implemented in KL05Z. The average power, energy per operation and run-times for each operations is also given in Table 6. The power is highly reduced with KL05Z since the clock frequency is decreased from 120 MHz to 48 MHz. The average power is measured to be 0.13 mW, which is 17% that of the K64F fixed point implementation and 7% the floating point implementation on K64F. The run-time however increases to 768 ns. This however still can support nearly 200 channels sampled at 7 kHz.

3.2.2. Memory Requirements
In total there are 27 variables declared that include the 16-sample buffer (for mean subtraction). The zero initialised data is therefore 27×2 bytes $= 54$ bytes and 27×4 bytes $= 108$ bytes respectively for KL05Z and K64F, with the program memory requiring approximately 3 kb flash. It should be noted that in our implementation the threshold that is calculated does not require a buffer, thus saving the need for a further 63 variables.

3.2.3. Real-time Implementation
A transient response demonstrating the operation of the hardware implementation is shown in the oscilloscope screenshot in Fig. 17. The system here is tested with a broadband input signal (including both LFP and extracellular action potential recording) with an increasing noise level (added Gaussian noise with SNR decreasing from 5 dB to -5 dB), then with a step change (to no added noise). This demonstrates the effectiveness and ability of the algorithm to adapt to signals with changing SNR.

3.3. Comparison with the State-of-the-Art
Although there exists a significant amount of previous work on spike detection [29, 49, 38], the vast majority focuses on computational methods for offline analysis. Although there are several examples in the literature of hardware implementations [55, 25, 52, 28, 16, 51, 30, 13], it is challenging to assemble a fair and comprehensive comparison. This is in part due to the diverse hardware methods available (e.g. ASIC implementation, embedded processor, reconfigurable logic, computational emulation). In this section we therefore select a narrow representative sample to include some qualitative comparison.

3.3.1. Computational Methods
Five specific algorithms from the literature have been selected for comparison with the presented algorithm, provided in Table 7. These methods achieve a slightly higher sensitivity to the method described herein (compared to fixed point implementation), but with a significantly higher computa-

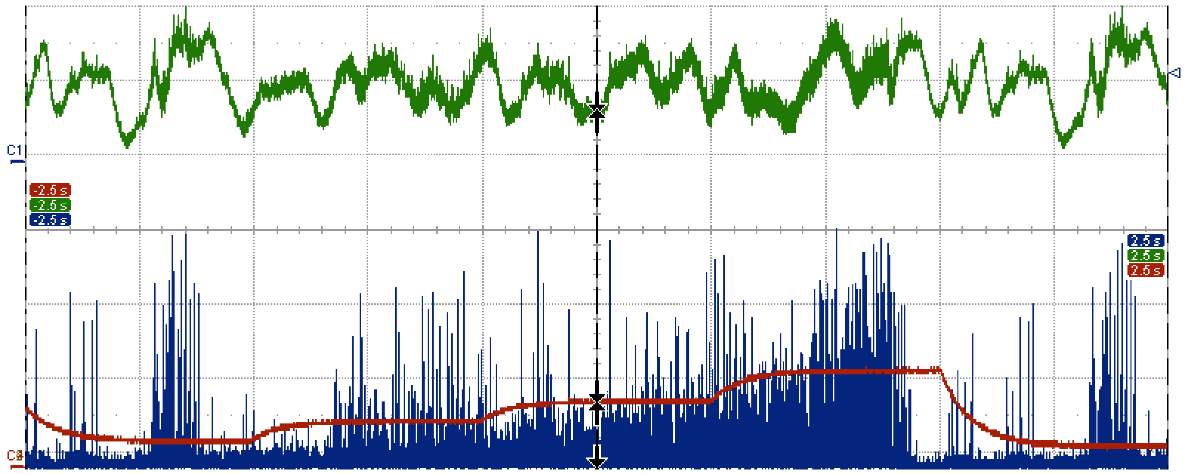

Figure 17: Real-time implementation of the adaptive spike detection algorithm on the KL05Z microcontroller platform. Shown are: the original input signal (green trace), SNR enhanced signal (i.e. pre-processed using ASO function) (blue trace) and adaptive threshold (red trace).

Table 7
Comparison with computational methods for spike detection

	Algorithm	data	Sens	Low complexity
Our work	ASO	Real data	89%	✓
[27]	TM	Real data w/o LPF	90.50%	✗
[28]	NEO	Synthetic	>90%	✓
[34]	SNEO	Synthetic	>90%	✓
[45]	SNEO	Synthetic	90%	✓
[17]	Sigma&Delta	Synthetic	99%	✓
[55]	PCA	Synthetic	99%	✗

Table 8
Comparison with hardware methods for spike detection

	Power /mW	Hardware	Pre-peocessing	Adaptive Thr
Our work	0.12	KL05Z	✓	✓
[3]	0.255	KL25Z	✗	✗
[16]	>41.9	MSP430F	✓	✗
[47]	16.5	MSP430	✓	✓
[25]	0.11	CMOS	✓	✓
[53]	$5e^{-5}$	CMOS	✓	✓

tional complexity. For example, not to mention the complexity of PCA decomposition in [55], which required offline training and real-time projection, the complexity of template matching [27] is at least O(MN), where M stands for the template length, and N is the signal length. For NEO [28] and SNEO [45, 34], this is reported to have a complexity of O(N), which comparable to the proposed method, however, they are using synthetic data and the LFP are not considered in some cases.

3.3.2. *Hardware methods*

The hardware implementation is compared with four different works that target embedded processors [3, 16] and an integrated circuit implementation [25, 53], shown in Table 8,

4. Conclusion

This work has presented a novel spike detection algorithm and hardware realisation intended for autonomous, calibration-free high channel count systems. Key features of the algorithm include:

- It includes a mean subtraction filter that can minimise the phase distortion whilst removing the LFP.
- A novel pre-processor to enhance SNR with the lowest reported computational complexity among all preprocesso
- A novel and robust thresholding schema which can reduce the effect of the spikes and multiunit activities on the stability of the threshold.

The hardware implementation of the algorithm achieved the following:

- Power consumption is amongst the lowest reported (130 μW average) for a microcontroller implementation including pre-filtering, SNR enhancement, adaptive threshold, and spike detection.
- From a hardware portability view, to prolong the battery life, the power consumption is expected to be scale down by 1-3 orders of magnitude (depending on technology) if translating to FPGA or ASIC implementation.
- Can be implemented using only fixed point arithmetic with no requirement for any multiplication operations.
- Requires a program memory of 3 kB and under 0.1 kB RAM.

- Low computational complexity allows for over 100 channels to be implemented on a single ARM Cortex-M0+ device.

With these outcomes, the proposed adaptive neural spike detection algorithm is suitable for multichannel implants of BMI applications, and the bit rate is excepted to be reduced for more than 20 times.

Acknowledgement

The authors would like to thank Rodrigo Quian Quiroga (Leicester) and Nick Steinmetz (University College London) for providing us the test datasets used herein.

Conflict of interest

We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

References

[1] Angotzi, G.N., Boi, F., Lecomte, A., Miele, E., Malerba, M., Zucca, S., Casile, A., Berdondini, L., 2019. Sinaps: An implantable active pixel sensor CMOS-probe for simultaneous large-scale neural recordings. Biosensors and Bioelectronics 126, 355–364.

[2] Bankman, I.N., Johnson, K.O., Schneider, W., 1993. Optimal detection, classification, and superposition resolution in neural waveform recordings. IEEE Transactions on Biomedical Engineering 40, 836–841.

[3] Barsakcioglu, D.Y., Constandinou, T.G., 2016. A 32-channel MCU-based feature extraction and classification for scalable on-node spike sorting, in: 2016 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE. pp. 1310–1313.

[4] Barsakcioglu, D.Y., Eftekhar, A., Constandinou, T.G., 2013. Design optimisation of front-end neural interfaces for spike sorting systems, in: 2013 IEEE International Symposium on Circuits and Systems (ISCAS2013), IEEE. pp. 2501–2504.

[5] Barsakcioglu, D.Y., Liu, Y., Bhunjun, P., Navajas, J., Eftekhar, A., Jackson, A., Quiroga, R.Q., Constandinou, T.G., 2014. An analogue front-end model for developing neural spike sorting systems. IEEE Transactions on Biomedical Circuits and Systems 8, 216–227.

[6] Casson, A.J., Luna, E., Rodriguez-Villegas, E., 2009. Performance metrics for the accurate characterisation of interictal spike detection algorithms. Journal of Neuroscience Methods 177, 479–487.

[7] Chan, H.L., Lin, M.A., Wu, T., Lee, S.T., Tsai, Y.T., Chao, P.K., 2008. Detection of neuronal spikes using an adaptive threshold based on the max–min spread sorting method. Journal of Neuroscience Methods 172, 112–121.

[8] Chaure, F.J., Rey, H.G., Quian Quiroga, R., 2018. A novel and fully automatic spike-sorting implementation with variable number of features. Journal of Neurophysiology 120, 1859–1871.

[9] Choi, J.H., Jung, H.K., Kim, T., 2006. A new action potential detector using the MTEO and its effects on spike sorting systems at low signal-to-noise ratios. IEEE Transactions on Biomedical Engineering 53, 738–746.

[10] De Dorigo, D., Moranz, C., Graf, H., Marx, M., Wendler, D., Shui, B., Herbawi, A.S., Kuhl, M., Ruther, P., Paul, O., et al., 2018. Fully immersible subcortical neural probes with modular architecture and a delta-sigma ADC integrated under each electrode for parallel readout of 144 recording sites. IEEE Journal of Solid-State Circuits 53, 3111–3125.

[11] Delgado-Restituto, M., Rodríguez-Pérez, A., Darie, A., Soto-Sánchez, C., Fernández-Jover, E., Rodríguez-Vázquez, Á., 2017. System-level design of a 64-channel low power neural spike recording sensor. IEEE transactions on biomedical circuits and systems 11, 420–433.

[12] Dunn, R.B., Quatieri, T.F., Kaiser, J.F., 1993. Detection of transient signals using the energy operator, in: 1993 IEEE International Conference on Acoustics, Speech, and Signal Processing, IEEE. pp. 145–148.

[13] Dwivedi, S., Gogoi, A.K., 2018. A novel adaptive real-time detection algorithm for an area-efficient CMOS spike detector circuit. AEU-International Journal of Electronics and Communications 88, 87–97.

[14] Eftekhar, A., Paraskevopoulou, S.E., Constandinou, T.G., 2010. Towards a next generation neural interface: Optimizing power, bandwidth and data quality, in: 2010 Biomedical Circuits and Systems Conference (BioCAS), IEEE. pp. 122–125.

[15] Even-Chen, N., Muratore, D.G., Stavisky, S.D., Hochberg, L.R., Henderson, J.M., Murmann, B., Shenoy, K.V., 2020. Power-saving design opportunities for wireless intracortical brain–computer interfaces. Nature Biomedical Engineering .

[16] Gagnon-Turcotte, G., Camaro, C.O.D., Gosselin, B., 2015. Comparison of low-power biopotential processors for on-the-fly spike detection, in: 2015 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE. pp. 802–805.

[17] Gagnon-Turcotte, G., Sawan, M., Gosselin, B., 2015. Low-power adaptive spike detector based on a sigma-delta control loop, in: 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 2167–2170. doi:10.1109/EMBC.2015.7318819.

[18] Gibson, S., Judy, J.W., Markovic, D., 2008. Comparison of spike-sorting algorithms for future hardware implementation, in: 2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 5015–5020.

[19] Gibson, S., Judy, J.W., Marković, D., 2011. Spike sorting: The first step in decoding the brain: The first step in decoding the brain. IEEE Signal Processing Magazine 29, 124–143.

[20] Harrison, R.R., 2008. The design of integrated circuits to observe brain activity. Proceedings of the IEEE 96, 1203–1216.

[21] Hulata, E., Segev, R., Ben-Jacob, E., 2002. A method for spike sorting and detection based on wavelet packets and shannon's mutual information. Journal of Neuroscience Methods 117, 1–12.

[22] Jun, J.J., Steinmetz, N.A., Siegle, J.H., Denman, D.J., Bauza, M., Barbarits, B., Lee, A.K., Anastassiou, C.A., Andrei, A., Aydın, Ç., et al., 2017. Fully integrated silicon probes for high-density recording of neural activity. Nature 551, 232.

[23] Kaiser, J.F., 1990. On a simple algorithm to calculate the 'energy' of a signal, in: International Conference on Acoustics, Speech, and Signal processing, IEEE. pp. 381–384.

[24] Karkare, V., Gibson, S., Marković, D., 2013. A 75-μw, 16-channel neural spike-sorting processor with unsupervised clustering. IEEE Journal of Solid-State Circuits 48, 2230–2238.

[25] Kim, D., Stanacevic, M., Kamoua, R., Mainen, Z., 2008. A low-power low-data-rate neural recording system with adaptive spike detection, in: 2008 51st Midwest Symposium on Circuits and Systems, IEEE. pp. 822–825.

[26] Kim, K.H., Kim, S.J., 2000. Neural spike sorting under nearly 0 dB signal-to-noise ratio using nonlinear energy operator and artificial neural-network classifier. IEEE Transactions on Biomedical Engineering 47, 1406–1411.

[27] Kim, S., McNames, J., 2007. Automatic spike detection based on adaptive template matching for extracellular neural recordings. Journal of Neuroscience Methods 165, 165–174.

[28] Koutsos, E., Paraskevopoulou, S.E., Constandinou, T.G., 2013. A 1.5 μw NEO-based spike detector with adaptive-threshold for calibration-free multichannel neural interfaces, in: 2013 IEEE International Symposium on Circuits and Systems (ISCAS2013), IEEE. pp. 1922–1925.

[29] Lewicki, M.S., 1998. A review of methods for spike sorting: the detection and classification of neural action potentials. Network: Computation in Neural Systems 9, R53–R78.

[30] Liu, Y., Luan, S., Williams, I., Rapeaux, A., Constandinou, T.G., 2017. A 64-channel versatile neural recording SoC with activity-dependent data throughput. IEEE Transactions on Biomedical Circuits and Systems 11, 1344–1355.

[31] Liu, Y., Urso, A., da Ponte, R.M., Costa, T., Valente, V., Giagka, V., Serdijn, W.A., Constandinou, T.G., Denison, T., 2020. Bidirectional bioelectronic interfaces: System design and circuit implications. IEEE Solid-State Circuits Magazine 12, 30–46.

[32] Luan, S., Williams, I., Maslik, M., Liu, Y., De Carvalho, F., Jackson, A., Quiroga, R.Q., Constandinou, T.G., 2018. Compact standalone platform for neural recording with real-time spike sorting and data logging. Journal of neural engineering 15, 046014.

[33] Moses, David A., L.M.K.M.J.G.C.E.F., 2019. Real-time decoding of question-and-answer speech dialogue using human cortical activity. Nature Communications 10, 3096.

[34] Mukhopadhyay, S., Ray, G., 1998. A new interpretation of nonlinear energy operator and its efficacy in spike detection. IEEE Transactions on Biomedical Engineering 45, 180–187.

[35] Musk, E., et al., 2019. An integrated brain-machine interface platform with thousands of channels. Journal of Medical Internet Research 21, e16194.

[36] Navajas, J., Barsakcioglu, D.Y., Eftekhar, A., Jackson, A., Constandinou, T.G., Quiroga, R.Q., 2014. Minimum requirements for accurate and efficient real-time on-chip spike sorting. Journal of Neuroscience Methods 230, 51–64.

[37] Obaid, A.M., Hanna, M.E.S., Wu, Y.W., Kollo, M., Racz, R.R., Angle, M.R., Muller, J., Brackbill, N., Wray, W., Franke, F., et al., 2019. Massively parallel microwire arrays integrated with CMOS chips for neural recording. bioRxiv , 573295.

[38] Obeid, I., Wolf, P.D., 2004. Evaluation of spike-detection algorithms for a brain-machine interface application. IEEE Transactions on Biomedical Engineering 51, 905–911.

[39] Paraskevopoulou, S.E., Barsakcioglu, D.Y., Saberi, M.R., Eftekhar, A., Constandinou, T.G., 2013. Feature extraction using first and second derivative extrema (fsde) for real-time and hardware-efficient spike sorting. Journal of Neuroscience Methods 215, 29–37.

[40] Putzeys, J., Raducanu, B.C., Carton, A., De Ceulaer, J., Karsh, B., Siegle, J.H., Van Helleputte, N., Harris, T.D., Dutta, B., Musa, S., et al., 2019. Neuropixels data-acquisition system: A scalable platform for parallel recording of 10,000+ electrophysiological signals. IEEE Transactions on Biomedical Circuits and Systems .

[41] Quiroga, R.Q., 2009. What is the real shape of extracellular spikes? Journal of Neuroscience Methods 177, 194–198.

[42] Quiroga, R.Q., Nadasdy, Z., Ben-Shaul, Y., 2004. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. Neural computation 16, 1661–1687.

[43] Rodriguez-Perez, A., Ruiz-Amaya, J., Delgado-Restituto, M., Rodriguez-Vazquez, , 2012. A low-power programmable neural spike detection channel with embedded calibration and data compression. IEEE Transactions on Biomedical Circuits and Systems 6, 87–100.

[44] Rosoklija, G.B., Petrushevski, V.M., Stankov, A., Dika, A., Jakovski, Z., Pavlovski, G., Davcheva, N., Lipkin, R., Schnieder, T., Scobie, K., et al., 2014. Reliable and durable Golgi staining of brain tissue from human autopsies and experimental animals. Journal of Neuroscience Methods 230, 20–29.

[45] Semmaoui, H., Drolet, J., Lakhssassi, A., Sawan, M., 2012. Setting adaptive spike detection threshold for smoothed teo based on robust statistics theory. IEEE Transactions on Biomedical Engineering 59, 474–482.

[46] Steinmetz, N., Carandini, M., Harris, K.D., 2019. "Single Phase3" and "Dual Phase3" Neuropixels Datasets .

[47] Turcotte, G.G., Camaro, C..D., Kisomi, A.A., Ameli, R., Gosselin, B., 2015. A wireless multichannel optogenetic headstage with on-the-fly spike detection, in: 2015 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1758–1761. doi:10.1109/ISCAS.2015.7168994.

[48] Watkins, P.T., Santhanam, G., Shenoy, K.V., Harrison, R.R., 2004. Validation of adaptive threshold spike detector for neural recording, in: The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, IEEE. pp. 4079–4082.

[49] Wilson, S.B., Emerson, R., 2002. Spike detection: a review and comparison of algorithms. Clinical Neurophysiology 113, 1873–1881.

[50] Yang, X., Shamma, S.A., 1988. A totally automated system for the detection and classification of neural spikes. IEEE Transactions on Biomedical Engineering 35, 806–816.

[51] Yang, Y., Boling, C.S., Kamboh, A.M., Mason, A.J., 2015. Adaptive threshold neural spike detector using stationary wavelet transform in CMOS. IEEE Transactions on Neural Systems and Rehabilitation Engineering 23, 946–955.

[52] Yang, Y., Kamboh, A., Andrew, J.M., 2010. Adaptive threshold spike detection using stationary wavelet transform for neural recording implants, in: 2010 Biomedical Circuits and Systems Conference (BioCAS), IEEE. pp. 9–12.

[53] Yang, Y., Mason, A.J., 2017. Hardware efficient automatic thresholding for neo-based neural spike detection. IEEE Transactions on Biomedical Engineering 64, 826–833.

[54] Yao, E., Chen, Y., Basu, A., 2016. A 0.7 v, 40 nw compact, current-mode neural spike detector in 65 nm cmos. IEEE Transactions on Biomedical Circuits and Systems 10, 309–318.

[55] Zviagintsev, A., Perelman, Y., Ginosar, R., 2006. Algorithms and architectures for low power spike detection and alignment. Journal of Neural Engineering 3, 35.