An Efficient Hardware Architecture for Template Matching-based Spike Sorting

Daniel Valencia and Amir Alimohammad
Department of Electrical and Computer Engineering
San Diego State University, San Diego, U.S.A.

***Abstract*—This article presents an efficient hardware architecture for the design and implementation of a spike sorting system using on-line template-matching. Over the past decade, various spike sorting algorithms have been proposed, however, due to their computational complexity, they may not be suitable for implantable devices that have stringent area and power consumption requirements. We first developed a software-based spike sorting system in both floating-point and fixed-point representations. Then we used our developed software-based spike sorting system for (i) studying various neural signal processing algorithms and assessing their feasibility for efficient hardware implementations and (ii) off-line processing of previously recorded neural data and extracting the threshold data and spike templates for configuring our spike sorting hardware architecture. The characteristics and implementation results of the designed spike sorting system on a Xilinx Artix-7 A200TFBG676-2 field-programmable gate array (FPGA) are presented. The ASIC implementation of the designed spike sorting system is estimated to occupy 0.3 mm^2. Post-layout synthesis and simulation shows that the ASIC implementation will dissipate 64 nW from a 0.25 V supply while operating at a 24 kHz frequency in a standard 45-nm CMOS technology. Compared to the previously published work, our ASIC implementation consumes 96.8% less power while maintaining a comparable sorting accuracy. Moreover, our design can run at a higher clock frequency and uses fewer hardware resources while achieving a 168% reduction in output data rate when comparing the raw data sampling rate and the sorted spike output rate, yet offers comparable spike sorting accuracy.**

I. Introduction

Neuro-scientists often require a single neuron's activity to study how neurons correlate with each other for specific stimulus. However, recording electrodes usually record signals from several nearby neurons, including some background noise. Due to the relative position of the neurons with respect to the recording electrode, propagation and velocity effects cause a distortion of spike shapes. Different spike shapes thus correspond to the activity of different neurons. Spike sorting is the process of separating the activities of individual neurons [1], i.e., determining which spikes correspond to which neurons. Conventionally, analog signals from neurons surrounding electrodes are digitized and transmitted to an external receiver for subsequent off-line software processing. Possible neural recording configurations include a physical connection from electrodes to a workstation for software processing, or wireless transmission of neural recordings. The data rate requirements for the wireless transmitter can become exceedingly large especially when recording from an array of microelectrodes or a multielectrode array (MEA) containing on the order of 100 recording sites, such as the Utah Array [2]. Depending on the resolution and sampling rate of the analog-to-digital converter (ADC), transmission of the raw neural recording produced by each recording site can be a few mega bits per second. For example, with sampling rates of up to 32 KSamples/s, ADC resolutions of 10 bits, and several recording sites (on the order of tens to hundreds), the raw data rates can be larger than 1 Mbps [3]. Wireless transmission of large amounts of data when the number of electrodes approaches a few hundred imposes serious limitations, such as the potential threat of heat-related tissue damage, as power consumption of the transmitter becomes too large. Since the majority of recorded information may be redundant, power consumption can be reduced significantly by on-chip neural signal processing. Digital signal processing can thus be used to process the recorded neural signals, which significantly relaxes the requirements on the wireless transmission through the skull. By performing neural signal processing at the recording site efficiently and transmitting only the spike sorting results, data rates and power consumption could be reduced significantly.

As shown in Fig. 1(a), the conventional process of an off-line spike sorting consists of four different modules: (i) spike detection, (ii) spike alignment, (iii) feature extraction, and (iv) clustering [4]. First, spikes are detected from an incoming bandpass ($300-3000$ Hz) filtered neural signal. The detected spikes are then aligned according to a certain attribute, such as maximum amplitude or maximum slope. In the feature extraction step, detected spikes are transformed to certain features such that similar spikes will have the same features, emphasizing the difference between spikes from different neurons and background noise. Finally, in the clustering stage, spikes are sorted to form groups or clusters of distinct spikes.

Each of these sub-modules of a spike sorting system can be realized using various algorithms. While off-line processing can utilize computationally-intensive but relatively accurate signal processing algorithms, neural prosthetic applications dictate that the spike sorting must be done in real-time so that the brain commands can be performed with negligible time delay and minimal data transmission rate. Therefore, it is crucial to choose the signal processing algorithms with the optimal balance between accuracy and computational complexity for real-time processing while staying within the strict area and power-density constraints. There have been several published work in both field-programmable gate array (FPGA) [5] [6] [7] and ASIC [8] [9] [10] [11] [12] that seek to reduce the output sorting data-rate, minimize the sorting latency associated with spike sorting, as well as achieving a reduction in computational resource utilization. The design in [5] presents the FPGA implementation of a spike sorting co-processor using a probabilistic neural network (PNN) algorithm. The work in [6] presents a real-time spike sorting system that uses Hebbian

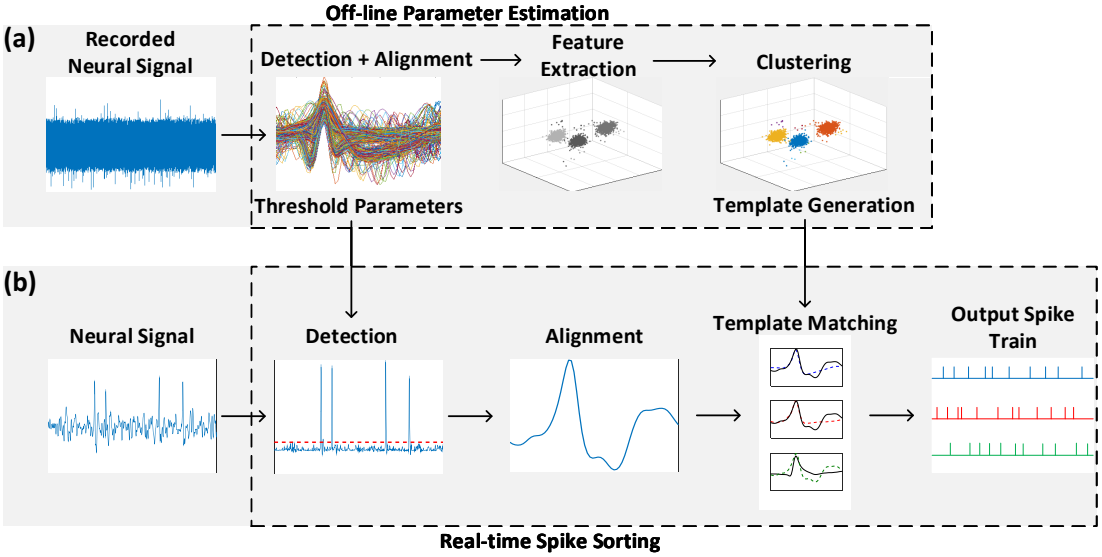

Fig. 1: (a) The conventional process of off-line spike sorting; (b) Using extracted features from the conventional spike sorting process, our proposed approach involves detecting spikes from neural signals, alignment to a certain metric and finally, comparing the aligned spikes to spike templates to generate output spike trains.

learning to implement the principal component analysis (PCA) for projecting input spikes to certain features. These two FPGA realizations seek to reduce the processing latency while reducing the number of utilized hardware resources. The work in [7] targets high channel count bidirectional brain-computer interfaces and supports on-line training to estimate spike templates for sorting. The work in [8] performs spike detection based on the non-linear energy operator (NEO) algorithm, aligns to maximum-derivative, implements feature extraction using discrete derivatives, and consists of four 16-channel modules. The work in [9] implements absolute value spike detection, clustering via OSort and consists of a single 16-channel module. Similar to [9], the work in [10] implements the OSort clustering algorithm. Spike detection is performed using a voltage threshold and spikes are aligned to maximum absolute amplitude (positive or negative). The work presented in [11] performs single-channel spike sorting utilizing NEO detection, aligns to peak amplitude, and performs feature extraction via adaptive discrete derivatives. Thus, it has a built in learning process, similar to other feature extraction and OSort implementations. The work in [12] performs multi-channel spike sorting via feature extraction as well as decision trees in place of memory units to sort spikes into separate classes.

In this article, we propose to significantly reduce the computational complexity of single-channel real-time spike sorting by using on-line template matching instead of performing feature extraction and clustering. The proposed architecture is scalable and can be readily used for spike sorting of multi-channel systems. Template matching [13] is the process of classifying newly detected action potentials (APs) by comparing their similarity to existing spike templates. This requires apriori knowledge of the spike waveforms, and hence, the proposed system is configured with spike threshold and spike template data generated by our software system using recorded neural data, as shown in Fig. 1(b). We first developed our proposed spike sorting system in both floating-point and fixed-point representations in MATLAB using our custom library of numerical operations in MEX/C. Using our developed offline spike sorting system, various neural signal processing algorithms are studied and their feasibility for efficient hardware implementation are investigated. We also used our software-based spike sorting system to extract the threshold data and spike templates from recorded neural signals. These parameters are used to configure our real-time spike sorting system. Fig. 2 shows one possible system-level configuration which interfaces our template matching-based spike sorting system with the recording electrode, the analog-to-digital converter and filtering unit, and the wireless transceiver (Tx/Rx) to interface with the workstation. The system would operate in two modes: (i) pass through mode to process recorded signals for offline parameter estimation; (ii) online spike sorting mode that only transmits the outputs of spike sorting, reducing the raw data rate. The process of off-line parameter estimation, selection of spike sorting algorithms for efficient hardware implementation, and our developed spike sorting software system are discussed in Section II. Sections III, IV, and V present the designed reconfigurable hardware architectures for spike detection, alignment, and template matching, respectively. Simulation results and hardware implementation results of the proposed spike sorting system on a FPGA are presented in Section VI. Section VII presents our ASIC implementation results in a standard CMOS technology and compare them with those of the previously-published work. Finally, Section VIII makes some concluding remarks.

II. Off-line Parameter Estimation

The requirements on the characteristics of implantable microelectronic devices are extremely stringent, especially with

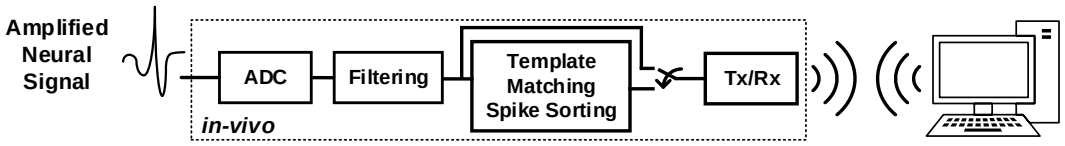

Fig. 2: System-level configuration of our proposed design.

respect to size and power consumption [14]. To meet such severe requirements, our proposed spike sorting system operates in two steps: (i) off-line estimation of template and threshold data using software and (ii) on-line detection, alignment, and template matching using an implantable hardware. A relatively long (about one minute) recording of neural data for a particular or a set of electrodes is processed by our developed spike sorting software for the estimation of the spike detection thresholds, the creation of spike templates used for template matching, and the template match threshold. The conventional spike sorting process includes detection, alignment, feature extraction, and clustering, which is computationally-daunting for on-line implementation. Since the estimated parameters are calculated once, it is feasible that the silicon area is not reserved for a process with such low utilization. By estimating the parameters offline, area and power dissipation can be significantly reduced. These parameters are then uploaded to the reconfigurable spike sorting hardware architecture for real-time processing.

A. Finding Spike Detection Threshold

The purpose of spike detection is to identify a neural spike, also known as an action potential, from ambient noise or from the idle period of a particular neuron or a group of neurons. The first stage of spike detection is known as pre-emphasis, in which an operation is performed on the input signal to prepare it for the second stage of detection, thresholding. There are many pre-emphasizing methods used for spike detection, such as absolute value [15], NEO [16], and discrete wavelet transform (DWT) [17]. Thresholding consists of comparing the pre-emphasized signal with a threshold value. If the pre-emphasized signal crosses the threshold value, this indicates that a spike is present. The thresholding approach based on absolute value is not adaptive while NEO and DWT-based detection algorithms utilize adaptive thresholding using the probability of false alarm P_{FA} and probability of detected spikes P_D. The energy operator can be defined as:

$$\psi\big(x(t)\big) = \left(\frac{dx(t)}{dt}\right)^2 - x(t)\left(\frac{d^2x(t)}{dt^2}\right), \tag{1}$$

where $x(t)$ is the instantaneous value of the input at time t. It can be shown that the output of NEO is proportional to the product of the amplitude and frequency of the input signal. For a discrete-time sequence $x[n]$, NEO is given as:

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

where $x[n]$ is a sample of the waveform at time n. The result is large only when the signal is both large in power (i.e., $x^2[n]$) and in frequency (i.e., $x[n]$ is large while both $x[n+1]$ and $x[n-1]$ are small). Since a spike, by definition, is characterized by localized high frequencies and an increase in instantaneous energy, this method is preferred over methods that only consider an increase in signal energy or amplitude. Another advantage of NEO is that it is relatively simple to implement, whether in the digital or analog domain, while DWT is significantly more computationally-intensive. The threshold Thr can be automatically set to a scaled version of the mean of the recorded signal as:

$$Thr = C\frac{1}{N}\sum_{n=1}^{N}\psi\big(x[n]\big), \tag{3}$$

where N is the number of samples in the signal and the scale C can be chosen initially by experiment as a constant (e.g., 8). The value of C will then be increased if $P_{FA} < 0.3$ and $P_D > 0.7$. Using the NEO algorithm, at least 98% of simulated neural spikes, developed as part of Wave_Clus software [18], which is a well-known and widely available unsupervised spike sorting program developed at the University of Leicester, can be detected.

B. Creation of Spike Templates

The earliest methods of spike sorting were performed by sorting based on spike amplitudes [19]. However, this approach suffers when the sensed neurons exhibit similar amplitude spikes. An alternative approach is to use window discriminators [20] in which one or more time amplitude windows are defined and the waveforms crossing them are assigned to a particular neuron. Even though implementing window discriminators is relatively straightforward and can be done in real-time, this approach is not practical when a relatively large number of electrodes are used. Moreover, windows may need to be readjusted during an experiment due to the non-stationary nature of the recordings and the consequent changes in spike shapes.

A different approach involves capturing features from the spike shapes that will be used for clustering the waveforms. For example, the peak amplitude and width of the spikes can be passed as inputs to a clustering algorithm. However, it has been shown in [4] that these features exhibit relatively poor accuracy when used for differentiating spike shapes. In general, the more discriminative features are used, the better the ability to distinguish spike shapes. If M samples are stored for each waveform, the spike shapes can be represented as points in an M-dimensional space. The complexity of clustering in a relatively large dimensional space demands dimensionality reduction, which only maintains the features that help the classification, given that eliminating inputs dominated by noise can improve clustering outcomes. The key challenge

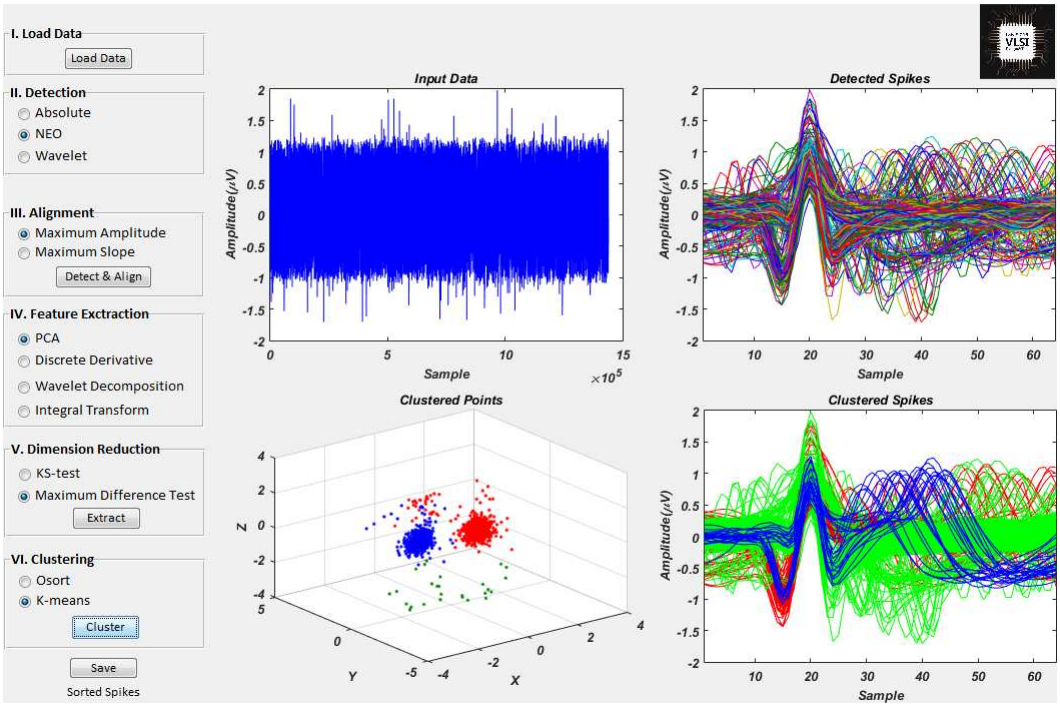

Fig. 3: Graphical user interface of the developed software-based spike sorting system.

is, however, selecting the minimal set of features that yields the best discrimination.

One of the most commonly used feature extraction and dimensionality reduction methods is PCA [21]. PCA gives an ordered set of orthogonal vectors that captures the directions of largest variations in the data and a waveform is represented as a linear combination of its principal components. Using only the first K components can account for most of the energy in the data and their scores can be used as an input to the clustering algorithm. The dimensionality reduction is thus achieved by going from an M-dimensional space to a K-dimensional space, with $K \ll M$ (typically, K equals to 2 or 3). During the off-line parameter estimation, PCA is performed on all detected spikes to create a subspace of maximum variance, which can be used to extract common features between spikes. The DWT has also found an application in feature extraction, as discussed in [18]. The output coefficients of the DWT, which describe both the time and frequency domains, can thus be used as features for clustering. Additionally, discrete derivatives, which computes the slope at each sample point of the spike, can be considered as a simplified DWT and has been used for feature extraction [22]. The integral transform (IT) [23] has also been used for feature extraction, where the discriminating features are the areas under the positive and negative portions of the AP. The appeal of the IT is its inherent reduction to two features.

Various clustering algorithms have been used for spike sorting. In manual cluster cutting [4], extracted features were plotted in a scatter plot, and cluster boundaries were defined manually. k-means clustering [24] is a commonly used clustering algorithm in the field of spike sorting, where an operator first defines k number of clusters, the k cluster centroids are randomly defined, and the new data points are assigned to the closest cluster by means of a distance metric. Once a cluster is assigned, the cluster centroid is recomputed as the mean of that cluster. The main disadvantage of these two clustering algorithms are that they require user supervision. One of the most efficient unsupervised clustering algorithm is OSort [25]. In principal, OSort uses the detected spikes to create spike clusters "on-the-fly"; the first spike is assigned to its own cluster, and newly detected spikes are compared to each cluster and are assigned to the nearest cluster according to the Euclidean distance metric.

The graphical user interface (GUI) of our developed software-based spike sorting system is shown in Fig. 3. This system is used for algorithm exploration, offline processing of recorded neural data, and extracting the template spikes, along with the spike detection and clustering threshold values. The software system performs all steps of the off-line spike sorting signal processing chain shown in Fig. 1 using a library of alternative algorithms. For spike detection, the absolute value, NEO, and DWT are supported. The user can select between the maximum amplitude and maximum slope for spike alignment, as well as various feature extraction algorithms, such as PCA, discrete derivatives, wavelet decomposition, and integral transform. For clustering, the system supports the Osort algorithm, and k-means. After clustering, the software displays all spike waveforms corresponding to various spike clusters. In Fig. 3, it is shown that a dataset is processed using NEO-based spike detection, spikes are aligned to maximum amplitude, PCA is used for feature extraction, with the maximum difference test for dimensionality reduction, and k-means clustering. Note that the two large clusters, shown in blue and red, are formed, while a smaller set of spikes have some common

features, shown in green, in the Clustered Points feature projection space. The spikes are then clustered and color coded accordingly in the Clustered Spikes view. The smaller set of spikes have some common features along the Y axis in the feature projection space, which are shown to be the maximum amplitudes of the spikes shown in green in the Clustered Spikes view. Note, however, that they have more variation compared to the red and blue waveforms.

We used the OSort algorithm to estimate spike templates without the need for apriori signal information. OSort will often converge to the N most-often occurring spike waveforms, which will form clusters. The number of generated clusters depends on the number of sortable neurons relative to the position of the recording electrode. Using simulated datasets with known ground truth, three spike waveforms can be used to represent the large majority of the types of waveforms that are detected. Therefore, these most commonly occurring waveforms are selected as spike templates and are used for template-matching in our spike sorting system. This is also convenient for estimating the maximum distance threshold, which is used to determine whether a new spike is that of the same class or type as that particular waveform. Note that while three templates are utilized for the selection of templates in this proposed design, the number of templates for practical applications can vary. For example, templates can be used to model overlapping spikes of known waveforms, which can help alleviate the overlapped spike problem. Note that the number of templates that are given dictates the number of sortable neurons, and as stated previously, the number of neurons that require sorting can vary between applications. Thus, our selection for the three most commonly occurring waveforms is appropriate for the ground truth dataset, which is mainly used to quantify the relative sorting performance of our proposed design. Since the estimation of spike templates is performed offline using our developed software system running on a workstation, alternative spike clustering algorithms can be utilized. For example, if PCA has the best spike waveform discrimination on a given dataset, then PCA can be used to cluster the spikes and create cluster average waveforms to use as static spike templates for the hardware.

III. Spike Detection Architecture

Fig. 4 shows the architecture of the designed and implemented NEO-based spike detection unit. The input signal is shifted into the *NEO Shift Reg*, which is used to delay the input signal accordingly to compute the energy of the signal at any given sample $x[n]$. The energy output $\psi[n]$ is then compared to the energy threshold. If the energy output matches or exceeds the given threshold value, the comparator will assert a high output value, which indicates to the alignment unit that a spike is present in its respective buffer. The given threshold is programmed by setting the threshold to the desired value while asserting the *PROG* input high. The NEO scalar, given as C in (3), is estimated via the P_{FA} and P_D, which are computed during the offline parameter estimation phase. This allows the end-user to readily fine-tune the threshold value used for spike detection.

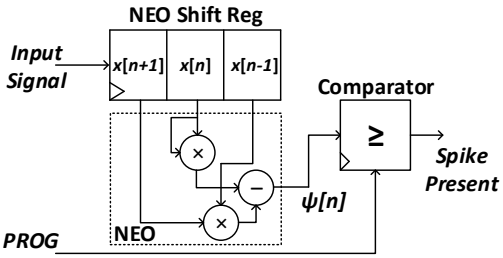

Fig. 4: Architecture of the NEO-based spike detection unit.

IV. Spike Alignment Architecture

For efficient template matching, spikes must be aligned. Aligning to the maximum amplitude is commonly used, as it is both computationally-simple and it allows the template matching module to easily differentiate spikes from one another. Because the stored templates are also aligned to maximum amplitude, this requires the detected spikes to be aligned to the same metric to increase the consistency of the template matching unit. As shown in Fig. 5, the spike alignment unit consists of two main buffers: the *Master Buffer* shift register, which receives new values of the input signal every clock cycle, and the *MBA* shift register. The *MBA* shift register is a parallel-in parallel-out register that copies the values stored in the *Master Buffer* when the *Spike Present* input signal is asserted by the NEO-based spike detection unit. The *Spike Present* signal is delayed using an N-bit shift register to allow the detected spike to buffer more input data for more accurate alignment. The *Control Unit* will assert the *LOAD* signal high to copy the contents of *Master Buffer* to *MBA*. The *Control Unit* then begins reading values from the *MBA* by controlling the select line *IDX* accordingly. The first value read from *MBA* is stored in the max value register *MR* by asserting the register enable signal *MVU* high. During all subsequent reads, the value read from the *MBA* will be compared to the value currently stored in *MR*, with *MR* being updated accordingly. The *Control Unit* also stores the index of the maximum value. The alignment point, which can be defined by the user as a top-level parameter, is used such that the maximum amplitude of each spike lies at the chosen alignment point index. This helps to prevent erroneous comparisons in the template matching module that would occur from temporal shifts in the waveforms, which can induce a significant distance metric error.

Disregarding the *Spike Present* buffering latency, that is, the time of actual spike occurrence delayed by the buffers, the alignment unit takes 17 clock cycles to find the maximum value within the master buffer. The latency is directly related to the number of samples in a spike waveform, the chosen alignment point, and the size M of the master buffer and the *MBA*. The alignment unit begins reading values from the alignment point index to *max address*, which is computed as $M - (NSS - AP)$, where M denotes the size of the master buffer, NSS denotes the number of samples in a spike waveform, and AP denotes the chosen alignment point. For 64-sample spike waveforms, buffer size $M = 80$, and an alignment point of 23, the maximum value search spans 17 values starting from the alignment point. Assuming a 24 kHz

sampling rate, the search spans 0.7 ms of signal time. The max index is used to then part select the parallel output of *MBA*, shown by the [ ] block, such that the maximum value is placed at the alignment point. The *Control Unit* also asserts an output flag to notify the template matching module that an aligned spike is ready.

One challenge during spike detection and alignment is due to the overlapping of spikes, i.e., when more than one spike resides in the detection window. While the refractory period of a neuron will prevent same-neuron overlapping, neurons can fire at around the same time. Thus, the input to the system is the summation of signals at the electrode. In the worst case, our system will incorrectly match the overlapped spikes to one of the given templates and in the best case, our system will not classify the overlapped spikes.

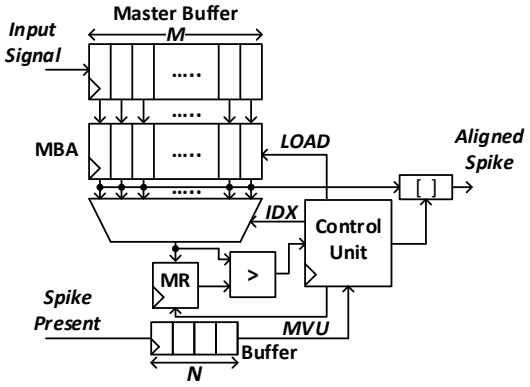

Fig. 5: Architecture of the spike alignment unit.

V. Template Matching Architecture

During template matching, the Euclidean distance metric is used for matching a new spike waveform to a template. The Euclidean distance is given as:

$$\sqrt{(x_0 - y_0)^2 + (x_1 - y_1)^2 + ... + (x_{N-1} - y_{N-1})^2} < T_M, \quad (4)$$

where x_k denotes the k-th sample of the detected spike waveform, y_k denotes the k-th sample of the template spike waveform, N denotes the number of samples used to represent a spike, and T_M denotes the match threshold. The match threshold defines the maximum distance the detected spike can be away from the template spike in order to classify it as part of that particular spike cluster. Since the off-line parameter estimation stage has access to significantly more storage, T_M is typically given as the standard deviation of the filtered neural recording.

Once the detected spike has been aligned, the last step is to classify the incoming spike from the alignment unit via template matching, which will indicate whether the spike is similar enough to one of the available templates. Utilizing the simulated neural data provided by WaveClus, we have opted to use three spike templates, each 64 samples long. The system can support an arbitrary number of spike templates at the cost of a larger silicon area due to the additional template memory requirement, as well as additional hardware to perform the comparison with the additional template. To classify the spikes, the Euclidean distance between an aligned spike and the three template spikes are calculated simultaneously based on Equation (4). To avoid implementing a square root unit, both sides of Equation (4) are squared as:

$$(x_0 - y_0)^2 + (x_1 - y_1)^2 + ... + (x_{63} - y_{63})^2 < T_M^2. \quad (5)$$

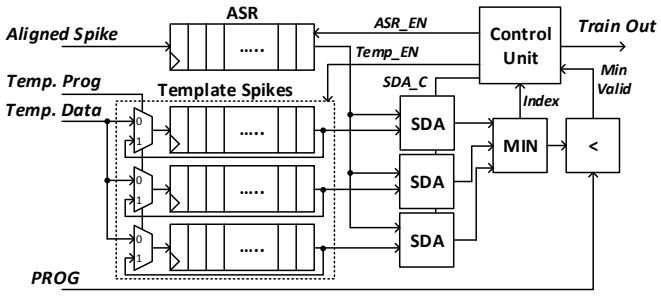

Fig. 6: Architecture of the template matching unit.

Fig. 6 shows the architecture of the designed template matching unit. The aligned spike is passed to the *ASR* shift register, which is configured for parallel-input and serial-output. The template spikes are stored in serial-in, serial-out shift registers during an initial programming sequence. The programming sequence involves shifting the spike template values on the *Template Data* port and setting the 3-bit *Template Prog* input to its proper value to program the respective template. The values stored in the templates and the *ASR* are shifted into the *SDA* units. The *SDA* units are used to compute and accumulate the squared difference between the spike waveform stored in the *ASR* and the templates. For spikes represented using 64 samples, it will take 64 clock cycles to compute the sum of squared differences between the aligned spike and each stored template. The *MIN* unit finds and passes the minimum value to the comparator, as well as the minimum value index to the *Control Unit*. If the minimum value is less than or equal to the stored maximum distance threshold, the comparator asserts the *Min Valid* signal high. The *Control Unit* will then set assert one of the bits of the 3-bit train output high to generate the spike train output. Thus, the large reduction of raw data to sorted spikes is due to only transmitting sorted spikes which are discriminated against only a small set of commonly occurring waveforms (templates).

VI. Simulation and FPGA Implementation Results

The top-level diagram of the proposed spike sorting system is shown in Fig. 7. The input control signal *PROG* is used to configure the system using the spike threshold, clustering threshold, and template spikes. The input from the recording electrode is given as input to both the detection and alignment units, while the input to the template matching unit is an aligned spike of size $WL \times NSS$, where WL denotes the input signal wordlength and NSS denotes the number of samples in a spike waveform. The designed spike sorting system implemented on a Xilinx Artix-7 XC7A200TFBG676-2 FPGA utilizes 4880 (1%) slice registers, 6628 (1%) lookup-tables (LUTs), and 5 (0.67%) DSP48E1 dedicated multipli-

ers. Because we utilize shift registers for storing template waveforms, no dedicated block RAM (BRAM) resources were utilized. While the spike detection module runs at the sampling frequency, the maximum operating frequency of the proposed spike sorting system is 102 MHz. After detecting a spike, our spike sorting system classifies the spike with a latency of 68 clock cycles. Fig. 8(a) shows the relative reconfigurable resource utilization of the FPGA when the *Template Match* unit is configured to support a varying number of template memories. Fig. 8(b) shows the maximum operating frequency of the FPGA when supporting various numbers of templates. While the resource utilization increases at a relatively low rate, it can be seen that supporting more than 4 clusters affects the maximum operating frequency. The optimal number of templates may vary between test subjects and/or neural recordings.

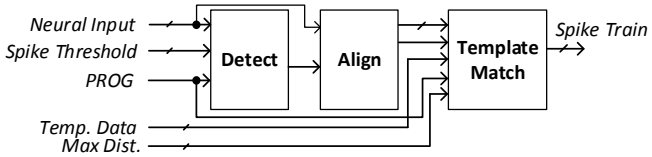

Fig. 7: Block diagram of the designed spike sorting system.

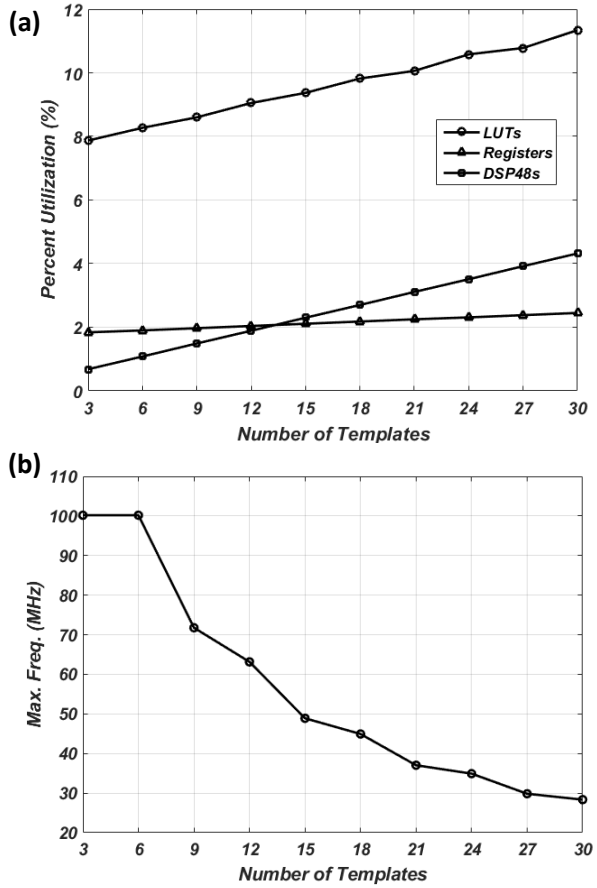

Fig. 8: (a) Relative utilization of FPGA resources and (b) maximum operating frequency for varying number of supported templates.

We have tested our spike sorting hardware against the WaveClus software for the two provided datasets: C_Easy1_Noise01 and C_Difficult1_Noise005 [18], which have signal-to-noise ratios (SNRs) of 0.1 and 0.05, respectively. Threshold parameters and spike templates were estimated using our software-based spike sorting system and then simulated using the Xilinx Vivado simulator.

To quantify the performance of our designed spike sorting system, we have employed the commonly-used F-score metric [26], which expresses the mean precision and sensitivity as:

$$F = \frac{2T_P}{2T_P + F_P + F_N},$$

where T_P, F_P, and F_N denote the number of true positives, the number of false positives, and the number of false negatives, respectively. A true positive is defined as a spike that has been classified correctly by our system. The classification is verified against the dataset by comparing the spike waveform as well as the time that the spike occurs. A false positive is defined as a spike that is classified, but does not exist in the dataset itself. A false negative is defined as a spike that exists in the dataset, but is not classified by our system. In testing with the two datasets, our design does not detect false positive spikes. Note that while the NEO-based spike detector may detect and pass false positive spikes to the alignment unit and template matching unit, the Euclidean distance between the false positive spikes and the verified templates will be greater than the maximum distance threshold in our testing, and will not produce a spike train output. In practical applications, however, it is possible for the template matching module to classify false positive spikes. For the Easy1_Noise01 and Difficult1_Noise005 datasets, our design achieves F-scores of 0.933 and 0.939, respectively. For the Easy1_Noise01 and Difficult1_Noise005 datasets, our design did not classify 438 and 384 spikes, respectively, due to the template matching unit being busy when detected by the detection unit. While it can be ensured that all detected spikes are maintained for classification by implementing a queue for storing detected spikes, however, the queue requires a significant amount of hardware resources, which considerably increases the silicon area and power consumption of the overall circuit. Additionally, some spikes may also be missed because of overlapped spikes that are not classified, as the overlapping spike templates have not been modeled. This is the tradeoff between using template-matching for spike sorting compared to the computationally-intensive, conventional off-line spike sorting processes. By limiting the number of template spikes, the number of candidate spikes that will appear at the spike train output may result in reduced accuracy. However, this reduces the total number of computations required to classify spikes. However, our simulation and implementation results verify that efficient real-time spike sorting is feasible for hardware implementation of implantable devices with comparable performance to offline spike sorting that uses more accurate, but significantly more computationally-intensive neural signal processing algorithms. Table I gives the characteristics and implementation results for our proposed design and the relevant published work for various wordlengths and different FPGA devices. The design

TABLE I: The characteristics and implementation results of various spike sorting realizations on different FPGAs.

Work	Algorithm	Device	$WL.WF$	Regs. (%)	LUTs. (%)	BRAMs (%)	DSP48s. (%)	Freq. (MHz)	Sorting Latency	Sorting Accuracy
[5]*	PNN	Virtex-6	16.7	3936 (1%)	13776 (7%)	7 (1%)	54 (4%)	100	6.7 μs	-
Ours	Template Matching	Virtex-6	16.11	4880 (1%)	6635 (3%)	0 (0%)	5 (0.6%)	122	0.55 μs	-
[5]*	PNN	Virtex-6	32.23	3936 (1%)	17712 (9%)	7 (1%)	92 (13%)	100	6.7 μs	-
Ours	Template Matching	Virtex-6	32.20	9616 (2%)	12729 (6%)	0 (0%)	11 (0.8%)	95	0.71 μs	-
[6]**	Hebbian	Spartan-6	10	∼6216 (3%)	∼4662 (5%)	41 (15%)	-	-	0.96 μs	95%
Ours**	Template Matching	Spartan-6	10.8	804 (0.43%)	1085 (1%)	0 (0%)	3 (1%)	78	0.87 μs	90%
[6]**	Hebbian	Spartan-6	16	∼8904 (5%)	∼6678 (7%)	65 (24%)	-	-	0.96 μs	95%
Ours**	Template Matching	Spartan-6	16.11	1224 (0.66%)	1572 (1.7%)	0 (0%)	3 (1%)	78	0.87 μs	90%
Ours	OSort	Virtex-6	16.11	8444 (2 %)	16472 (9 %)	29 (4 %)	130 (9 %)	123	0.25 μs	87%

*The authors in [5] utilize a floating-point number representation with 1 sign bit, 8 exponent bits, and WF fraction bits.
**The authors in [6] report the utilization for the Hebbian Eigenfilter module only.

in [5] presents the FPGA implementation of a spike sorting co-processor using a probabilistic neural network (PNN) algorithm. They utilize a custom-defined floating-point number representation with 1 sign bit, 8 exponent bits, and 7, or 23, fraction bits for their 16 and 32 bit designs, respectively. For a fair comparison, we have synthesized our design on the same FPGA with the same wordlengths. The design in [5] uses fewer registers than our design since we use a larger 80-sample detection window. However, our design uses fewer LUTs and DSP48E1 units. Moreover, our design does not use BRAM resources. While wider bus widths interconnecting the units shown in Fig. 7 contribute to the size of the system, spikes are transferred within the system in this fashion to minimize the latency between detection and classification. Moreover, the classification time, which is the computation time after spike detection given in Table I, indicates the advantage of spike sorting based on template matching over the ones based on the probabilistic neural network algorithm. The work in [5] reports a spike sorting performance of 92% accuracy. However, we cannot directly compare the accuracy of their work and ours because they use a dataset that is not openly available.

The work in [6] presents a real-time spike sorting system that uses Hebbian learning to implement the principal component analysis for projecting input spikes to certain features. They present results for 10-bit and 16-bit wordlengths, as well as modeling their designs using MATLAB's fixed-point toolbox, but do not report if any bits are allocated for the fraction. Their target device was a Xilinx Spartan-6 FPGA and their resource utilization was reported based on the reconfigurable slices. For a fair comparison, we also implemented our designs using the same two wordlengths, on the same target device. Note that the design in [6] only reports the implementation results of the Hebbian Eigenfilter hardware. Therefore, for a fair comparison, we have chosen to compare their Eigenfilter hardware to our implemented template matching hardware, as both designs identify and classify the input spikes accordingly. We have estimated their register and LUT usage using the conversion information in [27]. It is shown that template matching uses considerably fewer reconfigurable resources for classification of input spikes. Unfortunately, they have not reported their operating frequency, but their reported projection time is longer than our sorting latency. The authors in [6] reported 95% spike sorting performance when sorting spikes from three neurons. When the number of neurons increases from three to four, they reported 85% spike sorting performance accuracy. While the overlapping spikes contribute to performance reduction, we anticipate that the accuracy of our spike sorting system based on template matching would also experience a slight decrease in spike sorting performance, however, due to the larger number of APs to sort, the system would require more template spike memories.

While an increase in operating frequency naturally incurs an increase in power consumption, note that the results in Table I are for FPGA implementations. Thus, the maximum operating frequency metric is used when the spike sorting system is implemented on a FPGA for off-line acceleration of spike sorting. Because it is an off-line system, the classification steps of the spike sorting designs can be run at their maximum operating frequency to reduce the sorting latency as given in Table I.

We have also implemented the OSort algorithm on a Virtex-6 FPGA for comparison with the proposed template-matching approach. The OSort clustering unit is implemented using several BRAMs to store the growing clusters, which implements on-chip learning. Realizing the OSort algorithm utilizes significantly more resources than the proposed template matching approach. Our implemented OSort unit utilizes roughly twice as many registers, three times as many LUTs, and 26 times more dedicated DSP48. Because more hardware resources are used, note that the sorting latency is decreased compared to the template matching approach. However, for an efficient realization for in-vivo spike sorting, we opt to use the template matching approach, which utilizes significantly fewer dedicated multiplication units and BRAMs.

Multichannel-based designs are also of interest with the advent of high-density MEAs. Note that because of the hybrid offline-online approach, our in-vivo spike sorting ASIC works on a per-electrode basis. That is, there is no strategy that has been implemented which accounts for the spatial correlation between groups of neurons. The work in [7] has implemented a multi-channel sorting design on a Xilinx Kintex-7 FPGA, but they fail to list their relative resource utilization for

TABLE II: The ASIC implementation results of different spike sorting systems.

Work	This Work	[8]	[9]	[10]	[11]	[12]
Algorithm	Template Matching	Feature Extraction	OSort	OSort	Feature Extraction	Feature Extraction
Technology	45-nm	90-nm	65-nm	45-nm	45-nm	130-nm
Core voltage	0.25 V	0.55 V	0.27 V	-	1.1 V	1.2 V
Operating frequency	24 kHz	4 MHz	480 kHz	56 kHz	960 kHz	160 kHz
Area per channel	0.30 mm^2	0.06 mm^2	0.07 mm^2	0.07 mm^2	2.7 mm^2	0.023 mm^2
Power per channel	64 nW	2 μW	4.68 μW	10.3 μW	20 μW	750 nW
Average accuracy	90%	77%	75%	93%	84.5%	-
Data rate reduction	3200×	11×	240×	278×	240×	-

comparative analysis. We can anticipate that because their design implements on-line training functionality to generate templates for matching, it likely utilizes significantly more reconfigurable resources than our proposed design. They do list the required memory per channel in kilobits for a 16-bit data resolution as 6 kilobit/channel. In terms of memory utilization, our proposed design stores three 64-sample template waveforms with 16-bit data resolution, which could be interpreted as 3.072 kbit/channel. The work in [28] uses the MEA, however, since they have not utilized the same dataset, we cannot directly compare our spike sorting performance with their's. However, we can estimate that scaling our design to sort the APs of 128 neurons using an array of electrodes (e.g. Utah Array) would require at least 43 instances of our design, assuming 3 templates per electrode. On a high-end Xilinx Virtex-6 device, we can potentially fit about 68 instances, to detect APs of up to 204 neurons, with 98% LUT utilization.

In practical applications, the following steps can be taken to use the proposed template matching-based spike sorting system: (1) A neural recording is taken from the subject and processed offline on a workstation. Because the spike waveforms vary from person to person, it is vital to estimate initial detection threshold parameters, template waveforms, and maximum distance thresholds. Additionally, as mentioned earlier, slight electrode movements can induce changes in waveform appearance, which would require reprogramming of the spike sorting system; (2) Programming the proposed system with the estimated parameters from Step (1). The programming phase would involve first asserting the *RST* signal for the device high for at least M clock cycles, where M denotes the size of the master buffer in the alignment unit. The *RST* signal must be held for this long to allow for the input zero to propagate through the shift registers, thus resetting them. After, the *RST* signal is de-asserted, the *PROG* input signal and the *Template Prog* inputs are asserted accordingly to program the spike template shift registers; (3) Finally, the output of the recording electrode is fed into the spike sorting system and the device will process and sort the spikes in real-time. This allows researchers to study the behavior of neurons of interest in real-time rather than studying the neural recordings on a workstation post-experiment. Also, researchers can fine-tune the estimated threshold parameters should the system detect too few or too many spikes than expected for the given experiment. Moreover, the template matching unit is the key component to significantly reducing the output spike data-rate. The template matching unit will only assert output signals when a newly detected spike matches one of the given templates. Therefore, it eliminates the transmission of non-spike information, and additionally, reduces the amount of data needed to represent the spike information down to L, where L represents the number of supported template spikes in the system.

VII. ASIC Implementation Results

Our designed spike sorting system has also been implemented in ASIC using the FreePDK45 process design kit for a 45-nm CMOS process [29], with a 0.25 V supply voltage. Synthesis was performed using Synopsys design compiler and place-and-route was done using Cadence SoC Encounter. The design was synthesized for a 24 kHz operating frequency, however, timing analysis performed using Synopsys Primetime concludes that the ASIC can operate at a maximum operating frequency of 186 MHz. The ASIC chip layout shown in Fig. 9 is estimated to occupy 0.3 mm^2 and dissipate 64 nW of power while operated at 24 kHz. After the chip layout was completed, a final Verilog netlist was generated, and the design was compiled and simulated using Synopsys Verilog Compiler Simulator (VCS) and Discovery Visual Environment (DVE). The HDL testbench simulated in DVE, which consists of performing template waveform programming and normal template matching operation on a simulated dataset provided by Wave_Clus, dumps the switching activity of the post-layout internal nets of the design into a variable change dump (VCD) file. The VCD file was then read in Cadence SoC Encounter to accurately estimate the power consumption of the design. The hardware architectures for FPGA and ASIC implementations are identical, with the main differences being that the FPGA synthesis tool utilizes shift register lookup tables (to implement the various shift registers in the design), compared to using positive-edge triggered standard cells for the ASIC.

To investigate the feasibility of our spike sorting system as a brain implantable device, we have calculated important parameters, such as transmission power and power density of our ASIC. During spike sorting operation, the sampling bitrate r_s is 24 Kbps and using 16 bits for representation of the neural signal results in an input bitrate of 384 Kbps. As the typical spiking rate of a neuron is 40 spikes per second [1], using 3 bits to represent the spike train output, the output bitrate r_o is 120 bps. Thus, our template matching spike sorting system effectively reduces the bitrate by a factor of 3200.

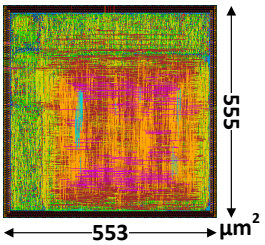

Fig. 9: Chip layout of the designed template matching spike sorting ASIC in a standard 45-nm CMOS process.

This reduction in output data rate significantly reduces the transmission power, thus reducing the amount of power needed for the wireless transmission of the output spike train. Note that this large amount of reduction is possible due to the fact that the template-matching approach discriminates to fewer spike waveforms compared to other conventional approaches while achieving comparable sorting accuracy. According to [30], the energy required to transmit one bit of data is 3 nJ, which results in a transmission power of about $0.36\ \mu$W with a r_o bitrate. Thus, the total power of our ASIC chip is $0.42\ \mu$W and the power density of our $0.3\ \mathrm{mm}^2$ chip comes to $1.27\ \mu\mathrm{W}/\mathrm{mm}^2$, which is well within the tissue-safe requirements [14].

Table II gives the ASIC implementation results of different spike sorting systems. As the ASIC chips in [8], [9], and [12] are multi-channel designs, we listed the area and power consumption results for a single channel for a fair comparison. As given in the table, our ASIC implementation consumes 96.8% less power per channel than other published works, yet still maintains a comparable sorting accuracy.

For the ground truth dataset used to quantify the performance of our proposed design, three templates are sufficient. However, some applications may require the use of larger numbers of templates to model the overlapping of specific neurons. Consider the following example of an electrode which detects the action potentials of 3 neurons A, B, and C. Seven template waveforms may be required to account for potential overlapped spikes $\big([A,\ B,\ C,\ AB,\ AC,\ BC,\ ABC]\big)$. However, this number of templates does not account for any relative time-shifts that the overlapped spikes can take. Thus, because of the combinatorial explosion that can result from attempting to model every possible overlap, preprocessing and analysis of neural signals is required to obtain the optimal number of templates. While different similarity measures between data and templates, such as Euclidean distance [31] [32] and cross-correlation [33] [34], have been employed for the selection of templates, a significantly more accurate approach for estimating the optimal templates was proposed by a form of linear filtering [35], which can be derived via Fisher's linear discriminant analysis [36]. In this approach, extracellular recordings were considered as having two different linearly added components, background activity (noise and action potentials from far away neurons) and spikes from close-by cells. Template matching was then performed using a finite impulse response (FIR) filter, which under the assumption of Gaussian noise is optimal in a Bayesian sense.

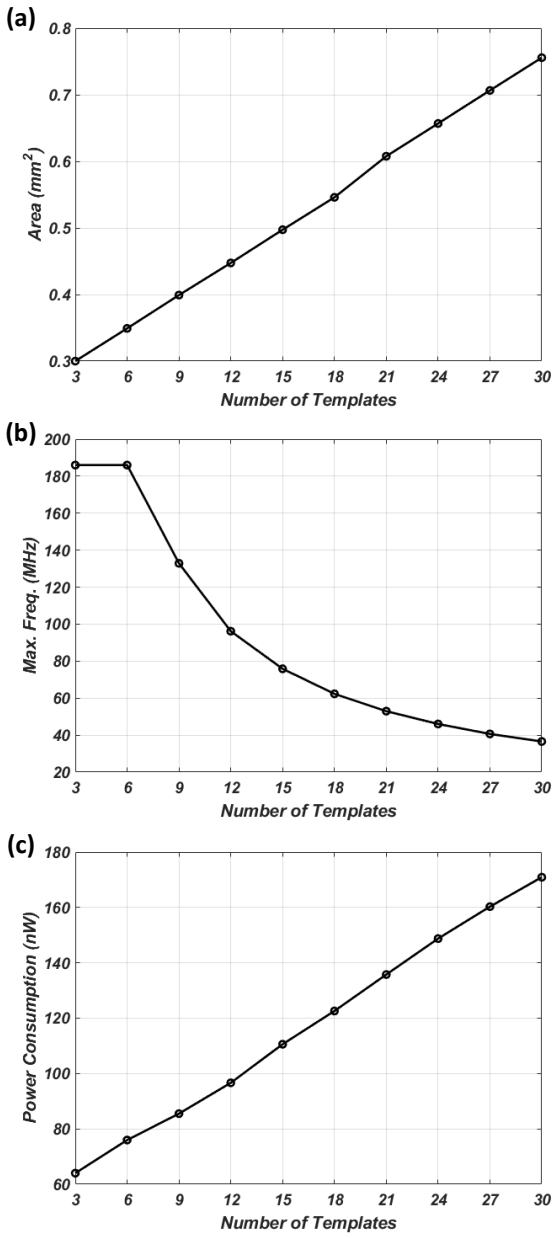

Fig. 10: (a) Area utilization, (b) maximum frequency, and (c) power consumption of the ASIC with storage for 3 to 30 template waveforms.

One advantage of the template matching approach is that template waveforms can be reconfigured, thus allowing the users to fine-tune the templates that are in use. For example, if the overlapping of two particular neurons is of importance, that particular overlapping can be modeled and stored as a template.

For efficient hardware realization, the optimal number of templates needs to be determined during the pre-processing of neural signals. While the details of this estimation are beyond the scope of our work, the limits of reconfigurable resources available on the target platform and power consumption of the implemented circuit place an important upper bound on the number of templates that can be modeled and stored. To explore the impact of the number of templates on the

implementation results of the template matching circuit, Fig. 10(a) shows the die area consumed by supporting 3 to 30 templates in the template matching portion of the ASIC. Fig. 10(b) shows the maximum operating frequency for supporting 3 to 30 templates. Naturally, more template memory results in an increase in silicon area, however, the maximum frequency varies between 187 MHz to 30 MHz. Fig. 10(c) shows the power consumption of the ASIC for supporting 3 to 30 templates. One can see that the power consumption grows roughly linearly with the increasing number of templates. As seen in Figs. 10(a) and 10(c), the most impactful aspects of increasing the number of templates are the silicon area and power consumption. While this does decrease the overall power density, it is simply due to the increase of silicon area. Additionally, an increase in the number of supported templates greatly diminishes the maximum operating frequency of the design, which is of importance for practical applications that execute different steps of spike sorting at different clock domains. For example, one may choose to increase the operating speed of the spike detection and alignment unit, such that the delay between alignment and classification is minimized. Note that due to the reconfigurable nature of FPGAs, the FPGA implementation can readily be scaled to support a large number of templates given the availability of computational resources and storage elements on the target device. For ASIC implementations, the number of templates is fixed.

VIII. Conclusion

This article presented an efficient hardware architecture for spike sorting using template matching. The computationally-daunting tasks of feature extraction and clustering used in the conventional spike sorting systems were performed off-line using our developed software-based spike sorting system on a workstation. By estimating template spikes from spike clusters, the hardware resource requirement and output data rates were significantly reduced. Comparison with the alternative spike sorting systems has confirmed that our design accurately sorts neural spikes using a smaller silicon area and significantly lower power consumption, while providing comparable performance. Our smaller and more power-efficient design makes it a suitable candidate for a multi-channel spike sorting system by instantiating multiple of the proposed spike sorting designs and time-multiplexing input/output ports.

Acknowledgment

This work was supported by the Center for Neurotechnology (CNT), a National Science Foundation Engineering Research Center (EEC-1028725).

References

[1] S. Gibson, "Neural spike sorting in hardware: From theory to practice," Ph.D. dissertation, University of California Los Angeles, 2012.

[2] R. R. Harrison *et al*., "A low-power integrated circuit for a wireless 100-electrode neural recording system," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 123–133, 2007.

[3] R. H. Olsson and K. D. Wise, "A three-dimensional neural recording microsystem with implantable data compression circuitry," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 12, pp. 2796–2804, 2005.

[4] M. S. Lewicki, "A review of methods of spike sorting: the detection and classification of neural action potentials," *Network: Comput. Neural Syst.*, vol. 9, no. 4, pp. R53 – R78, 1998.

[5] D. Wang, Y. Hao, X. Zhu, T. Zhao, Y. Wang, Y. Chen, W. Chen, X. Zheng, "FPGA implementation of hardware processing modules as coprocessors in brain-machine interfaces," in *International Conference of the IEEE Engineering in Medicine and Biology Society*, 2011, pp. 4613 – 4616.

[6] B. Yu, T. Mak, X. Li, F. Xia, A. Yakovlev, Y. Sun, C. Poon, "Real-time FPGA-based multichannel spike sorting using Hebbian Eigenfilters," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 4, pp. 502 – 515, 2011.

[7] J. Park, G. Kim, S. Jung, "A 128channel FPGA-based real-time spike-sorting bidirectional closed-loop neural interface system," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 12, pp. 2227–2238, 2017.

[8] V. Karkare, S. Gibson, D. Markovic, "A 130-μw, 64-channel neural spike-sorting DSP chip," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 5, pp. 1214–1222, 2011.

[9] V. Karkare, S. Gibson, D. Markovic, "A 75-μw, 16-channel neural spike-sorting processor with unsupervised clustering," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 9, pp. 2230–2238, 2013.

[10] Y. Liu, J. Sheng, and M. C. Herbordt, "A hardware design for in-brain neural spike sorting," in *IEEE High Performance Extreme Computing Conference*, 2016, pp. 1–6.

[11] M. Zamani, D. Jiang, and A. Demosthenous, "An adaptive neural spike processor with embedded active learning for improved unsupervised sorting accuracy," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 3, pp. 665–676, June 2018.

[12] Y. Yang, S. Boling, A. Mason, "A hardware-efficient scalable spike sorting neural signal processor module for implantable high-channel-count brain machine interfaces," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 11, no. 4, pp. 743–754, 2017.

[13] J. J. Capowski, "The spike program: A computer system for analysis of neurophysiological action potentials," in *Computer Technology in Neuroscience*, P. B. Brown, Ed. Washington: Hemisphere Publishing Corporation, 1976, ch. 17, pp. 237–251.

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

[15] I. Obeid, P. D. Wolf, "Evaluation of spike-detection algorithms for a brain-machine interface application," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 6, pp. 905–911, 2004.

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

[17] K. H. Kim, S. J. Kim, "A wavelet-based method for action potential detection from extracellular neural signal recording with low signal-to-noise ratio," *IEEE Transactions on Biomedical Engineering*, vol. 50, no. 8, pp. 999 – 1011, 2003.

[18] R. Quiroga, Z. Nadasdy, Y. Ben-Shaul, "Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering," *Neural Computation*, vol. 16, no. 8, pp. 1661 – 1687, 2004.

[19] W. Simon, "The real-time sorting of neuro-electric action potentials in multiple unit studies," *Electroencephalography and Clinical Neurophysiology*, vol. 18, no. 2, pp. 192–195, 1965.

[20] R. Q. Quiroga, "Spike sorting," *Scholarpedia*, vol. 2, no. 12, p. 3583, 2007.

[21] H. Hotelling, "Analysis of a complex of statistical variables into principal components." *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[22] Z. Nadasdy, R. Q. Quiroga, Y. Ben-Shaul, B. Pesaran, D. A. Wagenaar, R. A. Andersen, "Comparison of unsupervised algorithms for on-line and off-line spike sorting," in *Proceedings of the Annual Meeting Soc. for Neurosci.*, 2002.

[23] A. Zviagintsev, Y. Perelman, and R. Ginosar, "Low-power architectures for spike sorting," in *International IEEE EMBS Conference on Neural Engineering.*, 2005, pp. 162–165.

[24] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, 1967, pp. 281–297.

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

[26] C. J. V. Rijsbergen, *Information Retrieval*, 2nd ed. Newton, MA, USA: Butterworth-Heinemann, 1979.
[27] Xilinx, "Spartan-6 FPGA configurable logic block user guide," 2010.
[28] J. Dragas, D. Jckel, A. Hierlemann, and F. Franke, "Complexity optimization and high-throughput low-latency hardware implementation of a multi-electrode spike-sorting algorithm," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 23, no. 2, pp. 149–158, 2015.
[29] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh *et al.*, "Freepdk: An open-source variation-aware design kit," in *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*. IEEE, 2007, pp. 173–174.
[30] F. Chen, A. P. Chandrakasan, and V. M. Stojanovic, "Design and analysis of a hardware-efficient compressed sensing architecture for data compression in wireless sensors," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 3, pp. 744–756, 2012.
[31] Cambridge Electronic Design Limited, "Spike 2 version 7 manual," http://www.ced.co.uk/img/Spike7.pdf, 2012.
[32] Plexon Inc., "Plexon offline spike sorter manual, ch. 5.6.2," https://plexon.com/wp-content/uploads/2017/06/Offline-Sorter-v3-Manual.pdf, 2009.
[33] D. H. Friedman, *Detection of signals by template matching*. Johns Hopkins University Press, 1969.
[34] S. Kim and J. McNames, "Automatic spike detection based on adaptive template matching for extracellular neural recordings," *Journal of neuroscience methods*, vol. 165, no. 2, pp. 165–174, 2007.
[35] F. Franke, R. Q. Quiroga, A. Hierlemann, and K. Obermayer, "Bayes optimal template matching for spike sorting–combining fisher discriminant analysis with optimal filtering," *Journal of computational neuroscience*, vol. 38, no. 3, pp. 439–459, 2015.
[36] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

Daniel Valencia is a Research Assistant working in the VLSI Design and Test Laboratory in the Department of Electrical and Computer Engineering at the San Diego State University. His research interests include field-programmable gate arrays, embedded systems, and ultra-low power VLSI architectures for neural signal processing.

Amirhossein Alimohammad is an Associate Professor in the Electrical and Computer Engineering Department at the San Diego State University. He was the Co-Founder and Chief Technology Officer of Ukalta Engineering in Edmonton, Canada, from 2009-2011. He was a Postdoctoral Fellow at the University of Alberta between 2007-2009. He obtained a Ph.D. degree in Electrical and Computer Engineering from the University of Alberta in Canada and a M.Sc. degree from the University of Tehran in Iran. Before starting his Ph.D., he was a Hardware Engineer at Get2Chip GmbH, a Research Fellow in the Institute of Microelectronics at the University of Ulm and Atmel Wireless in Germany. His research interests include digital VLSI systems, reconfigurable architectures, wireless communication circuits, and signal processing algorithms.