A Multi-Channel Neural Signal Processing Engine using Synplify DSP Blockset

Vaibhav Karkare
Electrical Engineering Department
University of California, Los Angeles
vaibhav@ee.ucla.edu

Abstract—This report presents the design of a multi-channel neural signal processing engine using the Matlab/Simulink based design flow. Several algorithms used for spike sorting are analyzed in terms of their detection accuracy vs. hardware complexity. The “complexity – accuracy” optimal algorithms are implemented in Simulink using the Synplify DSP Blockset. Architectural transformation techniques like pipelining, interleaving, word length optimization etc. are applied to improve the design efficiency. This design can support around 1200 parallel channels at 0.4V consuming 2.5µW/channel when synthesized for a 90nm CMOS ASIC.

I. INTRODUCTION

Recording and processing of neural signals is extremely critical for the development of Neuroscience. The advancements in this field have a lot of potential applications in neuro-prosthetics, diagnosis and even entertainment. “Spike Sorting” is the process of mapping a recorded action potential to the source Neuron. This is essentially a detection & classification problem. However, unlike a traditional communication system, the set of signals to be classified is unknown at the receiver. The action potentials from a given neuron have an intrinsic variability, thereby complicating the situation. Moreover, the noise, dominated by surrounding neurons, is non–stationary and non–Gaussian. This implies that standard methods used in communication systems cannot be applied to this problem.

Traditionally, signals recorded from the brain are transferred over cables to an offline processor. These cables restrain the subject and also increase the risk of infection. The development of an on-chip spike sorting engine with wireless telemetry would eliminate these cables and allow for free movement of the subject. The hardware implementation is subject to aggressive low power and area constraints for it to be implantable in the brain. The upper limit for power dissipation has been quoted as 80mW/mm². [1]

The spike sorting process is classified into five steps – Detection, Alignment, Feature Extraction, Coefficient Selection and Clustering. [2] In this project we focus on Detection, Alignment and Feature Extraction. A lot of algorithms have been proposed in literature for each of these steps. However, there is no consensus on which algorithms are the best suited for hardware implementation. In this project, we aim at finding the “best-pick” among these algorithms. Further, we present an efficient hardware implementation of these algorithms using different architectural transformations. The design is done in a Matlab / Simulink based environment using the Synplify DSP blockset. This methodology is advantageous since the design needs to be entered only once. RTL generation, verification and test vector generation is done with a simple push button flow. Also, the graphical interface makes architectural transformations easier to visualize and implement.

Section I of this report presents our analysis of the existing algorithms for hardware implementation. Section II gives an overview of the hardware implementation and discusses some of its features. The third section presents the various architectural transformations applied to the design. Results and comparisons with the existing implementations are presented in the fourth section. The report is concluded and plans for future work are presented in Section V. The appendix lists some of the custom blocks used in this design. We think that if these blocks are added to the Synplify DSP Blockset, they could be useful for various other designs.

II. ALGORITHM SELECTION

The importance of selection of the right algorithms can hardly be over emphasized. Savings at the algorithm level far exceed any tricks that we play later at the circuit level. We analyze the existing algorithms in terms of their accuracy of detection vs. hardware complexity. Accuracy of these algorithms is estimated by using data generated from a Neural Signal Generator. Time complexity and count of atomic additions are popular methods for comparison of algorithms. These techniques however, might not always give accurate comparison of hardware complexity of different algorithms. In order to obtain more accurate comparisons for hardware complexity, we evaluate the number of addition operations, number of memory accesses and amount of memory storage required for an algorithm, at the bit level. Such a calculation would aptly punish accumulation more than regular addition. These calculations also bring up the excessive memory requirements in some algorithms. Number of additions are converted into equivalent power numbers assuming 2GOPS/mW at 0.4V in 90nm CMOS process.

This report is submitted towards partial fulfillment of requirements of the course EE219A at UCLA EE Department, Los Angeles.
A. Spike Detection

We consider “absolute value detection”, “non-linear energy operator (NEO)” and “Stationary Wavelet Transform Product (SWTP)” for spike detection. Figure 1 plots the detection accuracy vs. atomic complexity estimates for these three methods at various SNR levels. Bit level hardware complexity estimates are listed in Table 1. Absolute value detection is the cheapest, but is also very inaccurate. SWTP is less accurate and much more complex than NEO. We therefore, decide to use NEO for spike detection.

B. Alignment

“Maximum Derivative Alignment”, “Maximum absolute value alignment”, and “Alignment on threshold crossing” are the alignment methods that we consider. Hardware complexity estimates of these methods are listed in Table 2. The accuracy – complexity curves for the same are plotted for different SNRs in Figure 2. Unlike, the detection curves we do not find a knee point in the curve for alignment algorithms. Since we do not want to compensate on significant algorithm accuracy for saving hardware cost, we decide to use the maximum derivative method for alignment since it is the most accurate and is not much expensive as compared to maximum absolute value alignment.

C. Feature Extraction

“Principal Component Analysis (PCA)”, “Discrete Derivatives (DD)”, “Integral Transform (IT)” and “Discrete Wavelet Transform (DWT)” are the methods that we consider for Feature Extraction. Figure 3 shows the accuracy – atomic complexity plots for these methods. The detailed hardware complexity estimates for the same are listed in Table 3. The Discrete Derivatives method is found to have a reasonable accuracy and has a simple implementation. However, it calculates multiple levels of derivatives for a single data point. This causes a data expansion (300% for 3 level derivatives) and hence makes the following spike sorting steps extremely complex. IT performs poorly at low SNRs while PCA is too complex. Moreover, both the IT and PCA assume offline training and hence are not truly unsupervised. We therefore decide to use the DWT for feature extraction. The 4 - level Discrete Haar wavelet is the simplest wavelet and is also found to have good performance [3] as illustrated in Figure 3.

In summary, NEO for spike detection, Maximum Derivatives for Alignment, and DWT for Feature Extraction form the set of algorithms that we find to be best suited for hardware implementation.

II. HARDWARE IMPLEMENTATION

The algorithms chosen in Section I are implemented in Matlab / Simulink using Synplify DSP blockset. Figure 4 gives a block level overview of the system. The incoming data is routed to the threshold calculation block or to the detection circuit based on a training_n signal. (As a convention, we label all active low signals as signal_n ). During the training phase the threshold calculation block calculates the threshold for spike detection according to (1).

\[
Thr = \frac{C}{N} \sum_{n=0}^{N-1} \psi[x(n)]
\]  

(1)

where \( \psi[x(n)] \) is the non linear energy operator defined in (2).

\[
\psi[x(n)] = x^2(n) - x(n+1)x(n-1)
\]  

(2)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Absoloute Value</strong></td>
<td>1.3 µW 5x10^6 bits 0.25 Kbytes</td>
</tr>
<tr>
<td><strong>NEO</strong></td>
<td>64.7 µW 0 0</td>
</tr>
<tr>
<td><strong>Max. Absolute</strong></td>
<td>64.7 µW 0 0</td>
</tr>
<tr>
<td><strong>Max. Derivative</strong></td>
<td>64.7 µW 0 0</td>
</tr>
<tr>
<td><strong>IT</strong></td>
<td>0.16 µW 0 0</td>
</tr>
</tbody>
</table>

Table II. HARDWARE COMPLEXITY OF DETECTION ALGORITHMS
This threshold is fed to the Detection block. The detection circuit calculates $\psi[x(n)]$ for the incoming data stream and compares it to the threshold. A spike_detect signal is asserted upon detection. A set of 24 samples before threshold crossing and 48 samples after threshold crossing are saved in a RAM as a detected spike. This implies that detect signals in the first 23 cycles and detect signals within 47 cycles of a given detection have to be ignored. While a detected spike is fed into the RAM, it is also fed in parallel to a maximum derivative calculation block. This block calculates the derivatives at each point on the spike and determines the offset for the maximum derivative. If this offset lies outside the range (11, 35) then the spike is discarded. If the offset lies within this valid range then a read address is fed to the memory such that the maximum derivative occurs at the 11th position on the spike. This aligned spike is fed to the DWT block which calculates the detail and approximation coefficients of a 4-level Haar wavelet decomposition of the spike.

Along with the Simulink model, the system is also modeled as a Matlab code for verifying the Simulink design. Data from a neural signal simulator is fed to the code and to the Simulink model and the outputs are compared. Figure 5 shows a comparison of these outputs and demonstrates functional verification of the design. After verification, the design is synthesized for a Xilinx Virtex2 Pro FPGA. Test vectors are captured from the simulation. Synplicity tools also generate a “.do” file to facilitate RTL verification. This file is then executed in the ModelSim RTL simulator. The automated verification flow confirms the equivalence of the generated RTL to the Simulink model. We also check the same manually with the help of the waveform dump in the RTL simulator.

With a functionally verified design at hand, we use several architectural techniques to improve the efficiency of the baseline design.

### III. DESIGN OPTIMIZATION

In this section we give an overview of the design optimization techniques that are used.

#### A. Data Stream Interleaving

The bandwidth of neural signals is 12.5 Khz which implies that a sampling rate of 25Khz is sufficient for neural signal processing. Given that the speed of the technology is much higher, we can utilize the timing slack that we have to support interleaved data streams in an effort to reduce area. The design described in Section II has been interleaved to support multiple data streams. Interleaving the design for a different number of channels requires minimal changes in the Simulink model. This was demonstrated by changing the design from a two stream interleaved version to a four stream interleaved version.

#### B. Complexity Reduction

We find that the DWT is the timing bottleneck of the design and works much slower than the rest of the circuit. In an effort to improve the DWT performance we analyze the filter structure. A single stage of the 2 coefficient Haar wavelet filter has symmetric coefficients $H = [0.707 0.707]$ and $G = [-0.707 0.707]$. Looking at these numbers we immediately see that all the 4 coefficients can use a single multiplier. The wavelet filter bank is a multi-rate filter bank. The down sampling operation implies that half the computed samples are discarded. In order to avoid this power loss we decide to use a polyphase implementation of the filter. In this implementation the input data stream is split into odd and even streams, and combined later to give the output. This

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Computation Power</td>
</tr>
<tr>
<td>PCA</td>
<td>30.6 µW</td>
</tr>
<tr>
<td>DD</td>
<td>0.085 µW</td>
</tr>
<tr>
<td>DWT</td>
<td>2.4 µW</td>
</tr>
<tr>
<td>IT</td>
<td>0.16 µW</td>
</tr>
</tbody>
</table>
implementation implies that we only compute the samples that are retained after the down sampling in the original filter. Figure 6 demonstrates these optimizations on the Haar wavelet filter. However, we find that the DWT is still much slower than the rest of this circuit after all the optimizations.

C. Pipelining

The reason for the low speed of DWT is the long critical path that spans filters in all the four stages. To bring the circuit to a more “balanced” state in terms of timing we decide to pipeline the DWT critical path as shown in Figure 7. After pipelining the DWT critical path the operating frequency of the entire circuit improves by 3X. We also consider multi-rate folding on the DWT but find that it adds a lot of overhead which does not make sense given that we have a very simple filter.

D. Avoid Redundant Switching

We add / restructure logic wherever possible in order to reduce the switching activity of nodes within a circuit. As an example, figure 8(a) shows the direct mapped implementation of the threshold calculation circuit. In this implementation the multiplier switches each clock cycle even though the multiplication is needed only on the final accumulated value. In order to avoid the extra switching we AND the input to the multiplier with the training_n signal as shown in figure 8(b). This implies that the multiplier switches only when this signal is asserted. For a single stream no switching would take place in this circuit in the detection phase. However, in an interleaved circuit the accumulator would input a different value to the multiplier each clock cycle even in the detection phase. To avoid the same, we modify the circuit in figure 8(b) to the one shown in figure 8(c). In this circuit we AND the multiplier inputs with the positive edge detect signal on training_n. This implies that the multiplier would correctly pass the input to the detection circuit whenever a positive edge is received on training_n for a given stream. It also ensures that the multiplier would not switch thereafter, thereby avoiding power from being wasted.

E. Wordlength Optimization

We optimize the word-lengths for our design subject to a constraint of 0.001% accuracy for the output DWT coefficients. Wordlength optimization is performed based on the range of data at each node and the extent by which it affects the output. For example, the number of fractional bits in the coefficients of the Haar wavelet filter is found to have a higher sensitivity to the output error than the number of fractional bits in the threshold block. Hence, gain block in the Haar filter is assigned a higher number of fractional bits. The wordlength optimization gives us more than 50% reduction on FDE, LUT2 and XORCY usage on the VirtexII Pro FPGA. Also, wordlength optimization increases the operating frequency of the circuit by 9%.

IV. RESULTS

The optimized spike sorting engine can run at a speed of 70MHz on the Xilinx VirtexII Pro FPGA while consuming 4307 LUTs, 4 Multipliers and 776 Delay elements. When synthesized for a 90nm ASIC the design is mapped to approximately 18K cells, occupying an area of 0.17mm². The design can be operated up to 175MHz at 1V. Given the difference in the operating frequency and the input data rate, we would like to use VDD scaling and interleaving. At a supply of 1V, the circuit can support up to 6000 parallel channels.

Fig6. Complexity Reduction of the Haar – 4 DWT

Fig7. Critical path in DWT (in bold), reduced after pipelining

Fig8. Logic restructuring to reduce switching activity
with a power consumption of 4.3 $\mu$W/channel, running at 150MHz. The leakage to switching power ration is 0.44 and the total power consumption is 22mW. 6000 channels is an unrealistic number for current neural recording systems. Hence we would like estimate the power of the circuit upon $V_{DD}$ scaling. If we scale $V_{DD}$ to 0.4V, the frequency of operation goes down to 6MHz which implies that around 1200 channels can be supported. The switching power is lowered from 18mW to 0.1mW. However, the leakage power, is not positively impacted by the reduction in frequency, and is estimated (from Spectre simulations on an inverter) to be 0.2mW. This implies a total power of 0.3mW when the system runs at 6MHz and 0.4V to support 1200 channels. This gives us a power consumption of 0.25 $\mu$W/channel for this system.

The design supports multiple training periods. Further the training phase on each channel is independent. This implies that training for a noisy channel can be done more often without affecting the processing on other channels. The most recent publication in ISSCC’08 [6] demonstrates a Spike Sorting Engine that consumes 0.1mW/channel in 0.35 micron process. The comparison with the current implementation is not conclusive since the spike sorting engine published in [6] follows a set of algorithms which do not fit necessarily fit in the five step spike sorting framework followed in most literature on this subject.

V. CONCLUSION

In this project we identify spike sorting algorithms best suited for hardware implementation. These algorithms are efficiently built in hardware using several optimization techniques that we learn in the course. Our design can support 1200 parallel channels with a power consumption of 0.25 $\mu$W/channel operating at 6MHz at 0.4V. The support of 1200 parallel channels implies that Neuroscientists can get more valuable information about the brain. Building the clustering and coefficient selection blocks comes as a natural extension to this project. We also intend to fabricate the design to an ASIC.

REFERENCES


APPENDIX

CUSTOM BLOCKS WITH POTENTIAL USE IN SEVERAL DESIGNS.

In this section we present some custom blocks that were built for our design. We believe that these blocks, if available as a part of the Synplify DSP blockset could be used in several other designs.

1) Modified Downsampler

In most streaming applications it is often required to split odd and even samples of the valid data into different data streams. The down sampler block currently available does not include an enable signal. This means that we cannot selectively pick odd and even samples of a certain portion of the input data stream. For the current design we build a down sampler that supports such an operation.

Also, in case of data interleaving using a regular downsampler would mean that some data streams could completely be lost. For using an N times downsampler on an M-way interleaved design it is necessary to retain M-

samples and then replicate these M-samples for the next N-1 cycles. We believe that the modifications above are applicable for several other stream based multi-rate systems.

2) Enable Signal Generation

In the current design we were in need of an enable signal that stays high for M cycles after a receiving an input trigger pulse. The output enable was to serve as a write enable for the memory. We consider three different implementations for the enable signal generation circuit. The first implementation is based on an M-delay line where the output enable is an OR of all the bits in the delay line. This is not a good implementation since the number of registers in the delay line grows linearly with the number of clock cycles for which the enable needs to be asserted. The second implementation is a state machine based approach where the state machine is triggered when the input high and maintains a high output for M cycles. An interleaved design requires replication of the state
machines. In such a case the state machine assumes a ‘0’ at the input and goes to the next state even if the input is directed to another state machine in the current cycle. Therefore, the state machine based approach is not suitable for interleaved data streams. The third alternative is a counter based circuit where the output, input and the select line (for interleaving) go through a combinational logic and feed the enable of the counter. This gives us an enable generation circuit which can be interleaved for multiple streams. In our design, we use the same counter to generate address values for the memory, thereby, saving on some hardware.

The above requirement of generating a signal which is valid for M-spikes when a pulse is received at the input is common for many applications. A specific example would be that of data transfer protocols. This custom block could find good use in such designs.

3) Memory with Read Enable

A very simple modification to the existing RAM block is to provide a read enable signal on the memory. In our design we gate the output of the memory with an enable signal. There is often a need to force the output of the memory to a ‘0’ if we do not intend to read from it. This simple modification would come in handy in such cases.