33rd Annual International Conference of the IEEE EMBS
Boston, Massachusetts USA, August 30 - September 3, 2011

Massively Parallel Neural Signal Processing: System-on-Chip Design with FPGAs

Karthikeyan Balasubramanian and Iyad Obeid, *Member, IEEE*

***Abstract*—This work discusses the architectural layout and performance results of a SoC design for parallel neural signal processing. Architectural framework for scalability and scalar reconfigurability are presented. Architectural requirements for massive parallelism in neural recordings are presented. Prototype architecture with dual processors and multi-level reconfigurable platform design is presented. Functional modules of the platform include real-time spike detector and sorter for several hundreds of neural channels. Performance of the platform for a 300 channel interface is also discussed.**

I. Introduction

Neural networks of the brain often demonstrate spectacular processing efficiency under various circumstances, ranging from simpler sensory responses to much higher levels of complex cognition tasks. One of the key mechanisms for superior brain functionality is the hierarchical and massive parallelism exhibited by neuronal units [1]. These neuronal interconnections exhibit modalities of both localized organization and global functionality, tuned by synapses that are plastic and adaptive. Several researchers are attempting to understand the functionality of the brain using these neural signals [2][3][4][5], technically as a method of system identification using reverse engineering. Furthermore, neural prosthetics and brain-machine interfaces (BMIs), in general, harness signals that flow in these interconnections to control external devices. While these advancements greatly accelerated in signal acquisition and interface techniques, they are often limited by the number of recording channels that can be processed in parallel and in real-time [6]. Information retrieval in real-time from several neural interface channels is needed for successful BMIs of the future as well to further the understanding of brain functionality under the perspective of parallel architecture. In recent years, interest in developing neural signal processing algorithms that can be realized in hardware as well as the hardware architecture that can operate massively parallel signals is growing.

Few of the hardware substrates that are put to use for neural decoding include ASICs, DSP chips, FPGAs and GPUs. However, the architectural requirements and the choice of a particular hardware platform are determined, generally, by the experiment in question and the suitability of available algorithms.

In this work, the generic architectural layout needed for neural signal processing in a massive scale is discussed. A System-on-Chip (SoC) approach is presented for FPGA hardware realization that features (a) process parallelizability (b) multi-scale and partial reconfigurability, and (c) real-time operability. A prototype design incorporating a dual processor system and essential neural signal processing routines such as real-time spike detection and sorting is presented. The system operation is demonstrated for a 300 channel neural interface.

II. Massive Parallelism: Architectural Layout

Parallel processing and concurrent computation have been available for more than five decades and, multi-core and multiple processor platforms have been proliferating in recent years. These are intended to parallelize the underlying hardware operations for faster and efficient processing. Despite their huge operating efficiency, they were not found suitable for translational research such as BMIs and prosthetic device developments. This is partly due to their heavy reliance on their instruction set architecture and the intermediate hardware-software interface to achieve parallelism.

Real-time embedded systems and GPUs are used in several neural processing applications where the computational processes are mostly SIMD (single instruction multiple data). Typically, they operate on standard computational processes. On the other hand, DSP chips have strong knowledge base of software for various signal processing applications, but are often limited by the amount of parallelization possible at the software level given a fixed instruction set architecture. VLIW (very long instruction word) architectures explore instruction level parallelism, but still under the aegis of fixed architecture.

Parallelization can be achieved at various levels of a SoC design. Typically, code-level parallelization relies on compilers that can identify possible degrees of parallelism in the code and the suitability of parallel code execution on a given hardware. They offer the least degree of flexibility in terms of parallelization. Multiple processor systems use "hypervisors" to distribute the computational load across the different processors available. The most fine-grained parallelism could be achieved at the algorithm level, given the algorithm offers suitability.

This work was supported in part by the NSF Career Award under Grant No. 0846351.

Karthikeyan Balasubramanian is a Ph.D. candidate at the Neural Instrumentation Lab, Temple University, Philadelphia, PA 19122 USA (email: bkintex@temple.edu)

Iyad Obeid is with the Neural Instrumentation Lab, Temple University, Philadelphia, PA 19122 USA (phone: 215-204-3795; email: iobeid@temple.edu)

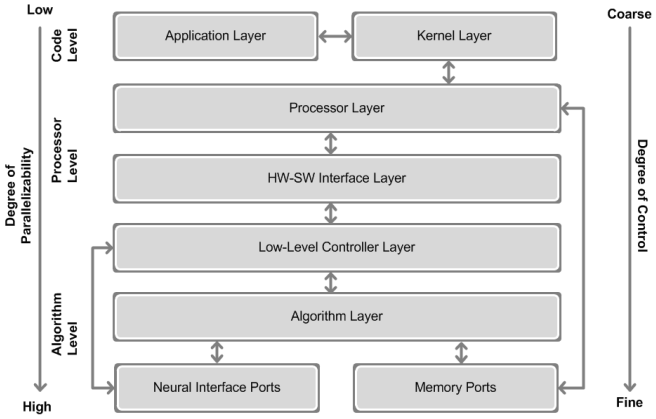

Fig.1 Schematic of an architectural layout for BMI hardware with neural signal processing in real-time and in parallel across thousands of channels. High levels of parallelizability at the algorithm level and coarse controls via application level routines are some of the essential features needed for scalable, reconfigurable and real-time operable architectures

Highly parallelizable algorithms for neural signal processing are still nascent. But, at the same time, several prevailing concepts of standard processing methodology can be tailored to function in conjunction with parallel architectures to meet the signal processing needs of a neural interface. In general, the architectural layout for massively parallel neural signal processing can be confined to (a) scalable and hardware parallelizable algorithms, (b) reconfigurable and adaptive architecture at hardware level and (c) operate in real-time and in synchrony across thousands of channels with software level accessibility.

Contemporary FGPAs possess several thousands to few million "uncommitted" logic gates that can be configured (and, reconfigured) towards specific tasks. The suitability of FPGAs towards achieving the architectural layout for neural signal processing is discussed in subsequent sections.

A. Scalability and Hardware Parallelization

Algorithms that claim to be scalable and parallelizable typically need substrates with spatial parallelism. Implementation-dependent hardware algorithms for various signal processing routines should be able to be implemented either as inter-dependent processes or completely independent instances on the same substrate offering physical scalability and parallelism. The raw computational logic cells of FPGAs offer substantial spatial parallelism. Alternatively, algorithms that use hardware controllers for their data-path can utilize the high fan-out capacities of FPGAs to drive multiple instances of an algorithm. Looping constructs of processing algorithms can be unrolled into parallel processing entities within an FPGA to address the hardware parallelization requirement. Finally, the hardware should support mixed-signal and multi-rate processing.

B. Reconfigurable and Adaptive Architecture

The choice of an algorithm or a combination of algorithms for neural signal processing mostly relies on ad-hoc selection processes by researchers. To facilitate selection of algorithms for real-world applications and to estimate their suitability for hardware operations, the architectural layout should possess the property of reconfigurability. Hardware implemented algorithms should have necessary interface constructs that allow partial or complete reconfiguration. Also, the architecture should offer the flexibility of recruiting portions of the system, when necessity arises. This can be analogous to the situation of muscle motor unit recruitment or distributed computing where additional elements are included or excluded based on instantaneous process load.

C. Real-time Operation and Software Interface

While it is a standard requirement for BMIs to operate in real-time in most of the applications, it is often related to the software that controls the platform. Real-time operation managed in entirety via software is dependent on number of subordinate peripherals, interrupt latency and task response time. For the purposes of massive parallelism in real-time, a software based solution may not be adequate. Hence, a hybrid approach that has hierarchical controller design with a low-level hardware controller targeted for hardware algorithms and a high-level controller based on software interface. High-level negotiation tasks should be controlled at the software layer while the interrupts generated by algorithm modules should be managed by dedicated hardware controllers.

III. SoC DESIGN FOR NEURAL SIGNAL PROCESSING

The SoC design implemented on an FPGA allows multiple scales of hardware customization and reconfiguration. Typically, at the lowest level, algorithm level parallelization is achieved and simultaneously at the highest level, code parallelization is possible. Due to the fundamental reconfigurable feature of FPGA substrates, resource sharing is highly customizable and lends itself as a suitable platform for SoC designs.

The prototype design presented here comprises two soft-core processors that manage the underlying tasks of neural signal processing. A non-linear energy operator-based (NEO) [7] spike detector and a fuzzy-logic based spike sorter [8] were implemented as the processing elements (referred as p-cores) with individual hardware controllers that manage the data flow and control flow within these p-cores. The processors interface with these p-cores via software drivers, and in conjunction with a real-time kernel monitors instantaneous outputs of these p-cores needed for further information retrieval. Communication between the processor and the p-cores are enabled via the standard processor local bus (PLB). Fig.2 shows the schematic of the design implemented in a Virtex-5 LX110T FPGA.

A. Spike Detector

The function of the spike detector includes (a) detection of spiking events and (b) extraction of valid spike

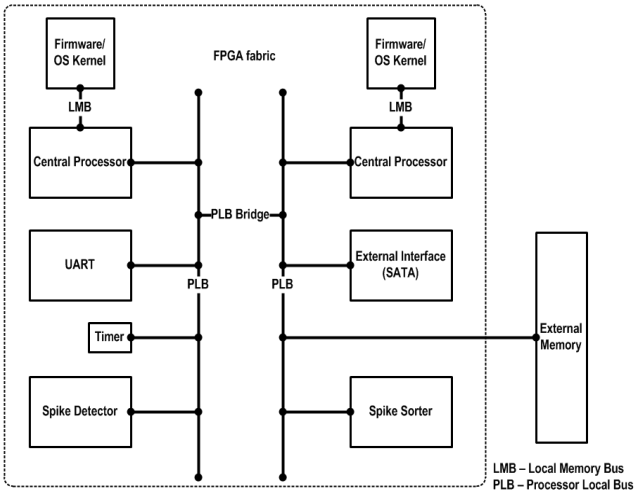

Fig.2 Schematic of SoC design implemented on Virtex-5 LX110T FPGA. The central processors are connected to the firmware layer via the LMBs. Communication between the processors and other subsequent layers are channelized through PLBs. Two PLBs are interconnected using a PLB-bridge. Both processors behave as masters of their respective PLBs, while all the other peripherals and p-cores are connected as slaves. Embedded system architectures offer the advantage of synchronizing various functions of a neural signal processor, as well as allowing signal integration over multiple channels

waveforms from the input neural signals. Neural signals from the data acquisition module, preceded by the recording channel ID, are transferred to input buffer of the detector module. The channel ID is temporarily stored in an appropriate register so that a detected spike event can be labeled when the waveform is extracted. The NEO was implemented as the preprocessor to the neural signal, whose mathematical representation is given by (1),

$$\psi[n] = x^2[n] - x[n+\delta].x[n-\delta] \qquad (1)$$

where, $x[n]$ is the digitized neural signal and $1 \leq \delta \leq 4$ is an integer. The preprocessor essentially emphasizes the probable spike events and simultaneously attenuates the noisy regions. The data then flows to the threshold comparator. Upon detection of a spiking event, the comparator logic flags a register for the real-time kernel to recognize. Spike waveform data is channelized into the output buffer by a *select* logic. A buffer to retain partial waveforms, cache LUT and the cache buffer are included to ensure extraction of complete spike waveforms.

B. *Spike Sorter*

Extracellular neural recordings from each channel contain action potential spikes from more than one neuron source. The primary objective of sorting process is to classify spikes amongst themselves as well as to separate biologically relevant spikes from noisy spike-like artifacts using extracted features. Spike data from the output buffer of the spike detector module is fed into the input buffer of the sorter by the central processor. The normalized spike data is then presented to the spike sorter module and the fuzzy spike score is determined. The output buffer that contains the spike score can be accessed by the processor or can be fed to a linear comparator for further processing. The linear comparator used preset threshold to compare the output fuzzy spike scores and ascertain a class label to the spike (for example, if the score range between 0.51 and 0.75, the spike might be from neuron-A and so on).

C. *Controller Design*

The real-time firmware functions as the top-level controller for the platform. Low-level controllers manage the control signals within the module. The flag signals that emanate from the p-cores are monitored by the real-time firmware and subsequently instructional control signals are issued to them via the PLB channel. Upon reset, the real-time firmware creates four threads, one for each of the tasks (see Fig.3), (a) to manage the data flow between the data acquisition and the input of the spike detector (DETI), (b) to monitor the spike detection (DETO), (c) to manages the data flow between the detector and sorter modules (SRTI) and (d) to monitor the spike sorting process (SRTO). The threads DETI and DETO are generated from the firmware of the Processor-1 and the remaining two threads are assigned from the Processor-2. A round-robin scheduling was used to share the processor time between the tasks. Both the processors operate in parallel and controls the p-cores connected to them.

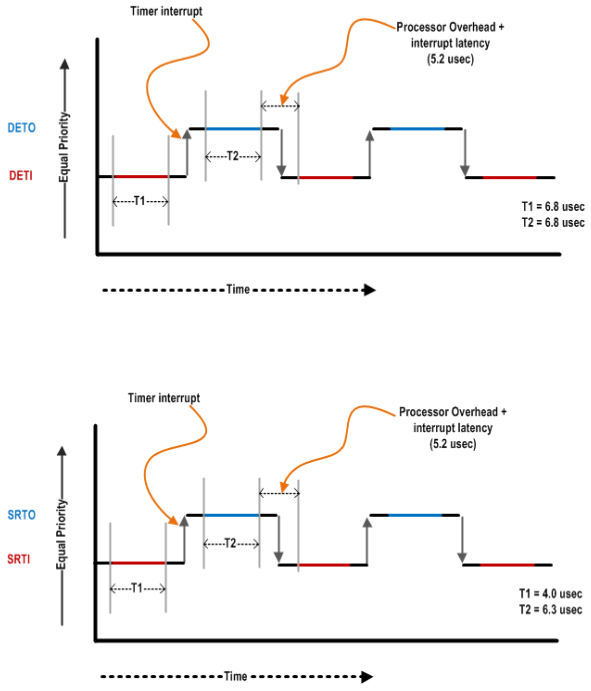

Fig.3 The top diagram shows the sequencing of the threads DETI and DETO by the Processor-1. Each thread operates for a pre-determined time and control transfer repeats till the processor is reset or turned off. The threads are added with processor overhead as determined by the compiler to enable proper scheduling. The bottom diagram is the thread scheduling in Processor-2. The interrupt latency was determined to be 5.2 usec. Time consumption by the different tasks are also given.

TABLE I
HARDWARE RESOURCE CONSUMPTION BY THE P-CORES AND THE SOFT-CORE PROCESSOR SYNTHESIZED FOR THE VIRTEX-5 LX110T

Module	Hardware	Consumption	Percent Utilization
Spike Detector	Slice Registers	774	2%
	Slice LUTs	4454	6%
	DSP Slices	18	28%
	BlockRAM	32 KB	1%
	Total Slices	**1855**	**10%**
Spike Sorter	Slice Registers	854	1%
	Slice LUTs	5294	7%
	DSP Slices	1	1%
	BlockRAM	288 KB	5%
	Total Slices	**2035**	**11%**
Microblaze Processor	Slice LUTs	1928	4%
	Total Slices	**482**	**8%**
Total Resource utilization			**29%**

IV. RESULTS

Implementation was carried out through high-level synthesis of the modules, i.e., the behavioral codes of the various functions were developed as high-level language codes which were then converted into hardware synthesizable descriptions. Hardware resources consumed by the platform in a Virtex-5 FPGA are given in Table I.

The platform was tested using a set of 100 channels of neural data, each comprising 10 seconds of extracellular neural recording. The test data set was created from pre-recorded extracellular signals obtained from the three animal species: an owl monkey, a macaque and a rat. The data were pre-loaded on to an external RAM and synchronously accessed for processing. The data stored in the RAM memory were originally sampled at 31.25 kHz.

To determine the percent correct detection of the spike detector and the percent correct classification of the sorter when they are operating in synchronous with the processor, data vectors from the p-cores were stored in to an external storage via the SATA port. In case of the spike sorter, spike scores and the channel ID were logged in the external memory and were then compared with the software results (see Table II).

V. DISCUSSIONS

Simultaneous recording of neural activity is a widely used and necessary process in understanding neural interactions and the functions of the brain. The number of electrodes used in invasive recordings has been doubling every 7 years since the introduction of multi-electrode recording in the 1950s. It has also been understood that the mutual information obtained from interacting neurons increases with increasing number of observed neurons, allowing better decoding of brain signals [9]. These factors emphasize the importance of highly parallelized computational infrastructure needed for neural signal processing.

Considering the various hardware choices, FPGAs offer higher degrees of parallelism, partly because of the inherent spatial parallelism of the substrate itself. Furthermore, the SoC design discussed here utilizes a low level controller for the p-cores and a high level controller to manage processor-level functions. This hybrid controller overcomes the unwanted interrupt latency experienced by processor-managed architectures. The low level control signals are managed by the modules themselves without the need for processor interrupts. Higher level control signals uses interrupts for task scheduling without disrupting the data pipeline in the modules. This allows the architecture to have improved bandwidth efficiency. The processor running at 100 MHz and the p-cores running at ~10 MHz, the platform is capable of handling 300 recording channels in parallel.

TABLE II
PERFORMANCE OF THE HARDWARE PLATFORM COMPARED WITH THE SOFTWARE IMPLEMENTATION

Parameter	Platform	Matlab
Number of spikes detected	206,422	206,422
Percent correct classification	91%	91%

Mean square error in fuzzy score estimates of the hardware and software implementations were less than 0.001, hence not significant.

VI. CONCLUSION

Computational demands for parallel neural recordings are becoming critical with the increase in the number of recording channels. A generic system-on-chip architecture was presented. Scalable and parallelizable algorithms are key elements for a successful neural signal processing platform. A standard non-linear energy operator-based spike detector and a novel fuzzy logic-based spike sorter were implemented in the system. FPGA substrate was used for implementing the design and offers higher degrees of scalability, reconfigurability and parallelizability.

REFERENCES

[1] O. Sporns, Networks of the brain. Cambridge, Mass, MIT Press, 2011.

[2] J. P. Donoghue, "Connecting cortex to machines: recent advances in brain interfaces," *Nat Neurosci,* 2002

[3] M. D. Linderman*, et al.*, "Signal Processing Challenges for Neural Prostheses," *Signal Processing Magazine, IEEE,* vol. 25, pp. 18-28, 2008

[4] N. Thakor, "Frontiers of Neuroengineering with focus on brain machine interface and neural prostheses" in *BIBE 2008. 8th IEEE International Conference on*, 2008, pp. 1-2

[5] G. Charvet*, et al.*, "A modular 256-channel Micro Electrode Array platform for in vitro and in vivo neural stimulation and recording: BioMEA" in *EMBC, 2010 Annual International Conference of the IEEE*, 2010, pp. 1804-1807

[6] K. Balasubramanian and I. Obeid, "Reconfigurable embedded system architecture for next-generation Neural Signal Processing," in *EMBC, 2010 Annual International Conference of the IEEE*, 2010, pp. 1691-1694

[7] I. Obeid and P. D. Wolf, "Evaluation of spike-detection algorithms for a brain-machine interface application," *Biomedical Engineering, IEEE Transactions on,* vol. 51, pp. 905-911, 2004.

[8] K. Balasubramanian and I. Obeid, "Fuzzy logic-based spike sorting system," *Journal of Neuroscience Methods,* vol. In Press, Uncorrected Proof.

[9] I. H. Stevenson and K. P. Kording, "How advances in neural recording affect data analysis," *Nat Neurosci,* vol. 14, pp. 139-142, 2011.