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

Modular Particle Filtering FPGA Hardware Architecture for Brain Machine Interfaces

John Mountney
Electrical & Computer Engineering
Temple University, Philadelphia, PA

Iyad Obeid
Electrical & Computer Engineering
Temple University, Philadelphia, PA

Dennis Silage
Electrical & Computer Engineering
Temple University, Philadelphia, PA

Abstract—**As the computational complexities of neural decoding algorithms for brain machine interfaces (BMI) increase, their implementation through sequential processors becomes prohibitive for real-time applications. This work presents the field programmable gate array (FPGA) as an alternative to sequential processors for BMIs. The reprogrammable hardware architecture of the FPGA provides a near optimal platform for performing parallel computations in real-time. The scalability and reconfigurability of the FPGA accommodates diverse sets of neural ensembles and a variety of decoding algorithms. Throughput is significantly increased by decomposing computations into independent parallel hardware modules on the FPGA. This increase in throughput is demonstrated through a parallel hardware implementation of the auxiliary particle filtering signal processing algorithm.**

I. Introduction

Encoding a biological signal or decoding a stimulus for an ensemble of neurons is the primary objective of the brain-machine interface (BMI). Most BMIs are implemented by adaptive filtering algorithms that use the individual firing rates of neurons in the ensemble to estimate the underlying affector signal. Linear adaptive algorithms such as the Weiner filter, least mean squares (LMS) algorithm and the Kalman filter have been employed to achieve pseudo electromyographic signal reconstruction. Nonlinear approaches to the estimation problem for BMIs include the extended Kalman filter, the unscented Kalman filter, point process adaptive filters and particle filters [1].

It has been suggested that particle filters are better suited for neural signal processing than other estimation algorithms since they can be applied to non-Gaussian, non-stationary environments [2]. However, this increase in accuracy comes at the expense of computational complexity. For this reason, particle filters have not been currently employed in real-time BMI applications.

This work describes a parallel hardware architecture that significantly increases throughput over sequential processing. This hardware architecture is designed with a field programmable gate array (FPGA) as the target device. The reconfigurability of FPGAs allows the device to be reprogrammed to support various types of neurons and neural parameters as well as incorporating an array of particles.

II. The Bayesian Auxiliary Particle Filter

A major deficiency in particle filtering is the effect that outliers or unexpected observations have on filter performance [3]. Auxiliary particle filters address this problem by performing resampling at time $t-1$ using the current observation at time t before updating the particles. More specifically, the Bayesian auxiliary particle filter (BAPF) introduced by Liu and West [4] uses a two-stage weight update procedure. The first stage weights $g(t)$ are computed as an intermediate step used in the resampling process. The second stage weights $w(t)$ are used to compute the state estimate $\hat{\mathbf{x}}(t)$ as a weighted sum.

The BAPF algorithm for neural decoding can be defined according to the following steps:

1) The state estimate of the n^{th} particle is updated by drawing a new sample about its previous estimate with a specific variance σ_1^2. This is accomplished through Equation 1 by adding zero-mean random samples Θ^n to each element in the state vector.

$$\hat{\mathbf{x}}^n(t) = \hat{\mathbf{x}}_r^n(t-1) + \Theta_1^n \tag{1}$$

2) The first stage weight $g^n(t)$ of the n^{th} particle is defined to be proportional to the product of its second stage weight from the previous iteration $w^n(t-1)$ and the likelihood of observing the current neural firing activity $N(t)$ given its state estimate $p^n(N(t)|\hat{\mathbf{x}}^n(t))$.

$$g^n(t) \propto w^n(t-1)p^n(N(t)|\hat{\mathbf{x}}^n(t)) \tag{2}$$

The first stage weights are then normalized so all P particle weights sum to unity.

3) Particles are then resampled according to their normalized first stage weights and assigned a new state estimate $\hat{\mathbf{x}}_r^n(t)$. The resampling process replicates highly weighted particles and discards particles with low weights. This process was introduce in the sampling importance resampling (SIR) particle filter [5].

4) The particle state is then updated once more by drawing random samples about its resampled estimate. This is similar to Step 1 except that the random samples are drawn from a distribution with a different variance σ_2^2.

$$\hat{\mathbf{x}}_r^n(t) = \hat{\mathbf{x}}_r^n(t) + \Theta_2^n \tag{3}$$

5) The second stage weights are then computed as a ratio of the likelihood computed using the resampled estimates to the likelihood computed using the original estimates.

$$w^n(t) \propto \frac{p^n(N(t)|\hat{\mathbf{x}}_r^n(t))}{p^n(N(t)|\hat{\mathbf{x}}^n(t))} \tag{4}$$

6) A final estimate $\hat{\mathbf{x}}(t)$ of the system state is then computed as a sum of all P particle estimates weighted by their respective normalized second stage weights

$$\hat{\mathbf{x}}(t) = \sum_{n=1}^{P} w^n \hat{\mathbf{x}}_r^n(t) \tag{5}$$

A. Verification of the BAPF

To demonstrate the improved decoding accuracy of the BAPF over existing methods, a simulated experiment is presented. One dimensional animal position $s(t)$ is predicted on a 300 cm track while simultaneously compensating for neural plasticity by observing simulated neuronal activity. The BAPF predicts $s(t)$ and the receptive field center $\mu_j(t)$ of K hippocampal place cell neurons whose expected firing rates are described by the following tuning function

$$\lambda_j(t) = exp\left\{\alpha - \frac{(\mu_j(t) - s(t))^2}{2\xi^2}\right\} \tag{6}$$

The neural firing times are assumed to arrive according to an inhomogeneous Poisson process at an average rate of $\lambda_j(t)$ for the j^{th} neuron. The likelihood of particle n is therefore defined as

$$p^n(N(t)|\hat{\mathbf{x}}^n(t)) = \prod_{j=1}^{K} (\lambda_j \Delta(t))^{\Delta N_j(t)} e^{-\lambda_j(t)\Delta(t)} \tag{7}$$

where $\Delta N_j(t)$ is either 1 or 0 (neuron j fires or it does not fire) over a sampling period $\Delta t = 1$ ms for $j = 1 \ \ldots \ K$ observed neurons. Figure 1(a) compares the mean square error (MSE) between the true position and predicted position obtained from the Weiner filter, the SIR particle filter and the BAPF as a function of the number of neurons. In figure 1(b), 50 neurons were decoded as and the average mse is given as a function of the number of particles. This demonstrates the improvement in the BAPF decoding accuracy as more particles are included in the decoding process.

Figure 1 suggests that a high number of observed neurons decoded with a large number of particles will yield the best decoding accuracy by the BAPF. If these values are too large, the number of computations to be executed for a single iteration of the BAPF can not be performed in real-time using a sequential processor. This indicates that an alternative implementation of the BAPF is needed for BMIs.

III. Hardware Implementation

If real-time execution of the BAPF is desired, then a parallel dedicated hardware implementation may be required. Implementing digital signal processing (DSP) algorithms in dedicated hardware eliminates the need to sequentially fetch and decode instructions. Additionally, a hardware architecture can exploit the computational independence between particles to perform multiple parallel operations simultaneously.

FPGAs provide an ideal platform for DSP implementation, combining the reprogrammability, architectural flexibility and system-level integration of general purpose processors with the performance offered by customizable hardware. Even with current clock speeds well below conventional processors, reconfigurable logic can yield substantially superior throughputs when made massively parallel [6].

FPGAs are customizable logic devices that are comprised of configurable logic blocks (CLBs), programmable interconnections and input/output (I/O) cells. Each CLB may include: look-up tables (LUTs), multiplexers, flip-flops, basic logic elements (AND, XOR), block random access memory (BRAM) and shift registers. In addition to the CLBs, current devices incorporate dedicated fixed point multipliers for efficient implementation of arithmetic functions.

An FPGA uses a general routing matrix (GRM) to configure the interconnects between each of its CLBs. The GRM is an array of static memory cells which store values that control the configurable logic elements and interconnects between the CLBs. These configuration values can be reloaded when the functionality of the device needs to be altered.

Described next is a parallel FPGA hardware architecture for utilizing the BAPF in real-time neural signal processing. The design is a multi-rate system which is both modular and scalable. The modularity allows new BAPF architectures to be easily reprogrammed into the device to support a variety of functions. The scalability allows the number of observed neurons and the number of particles to be set according to the desired application.

A. BAPF Top-Level Diagram

Figure 2 shows the top-level FPGA configuration for the BAPF. There are four major components: 1) the controller, 2) the random number generator (RNG), 3) the particle processors and 4) the resampler. Each of these components can be easily replaced with another module to accommodate a new neural signal processor.

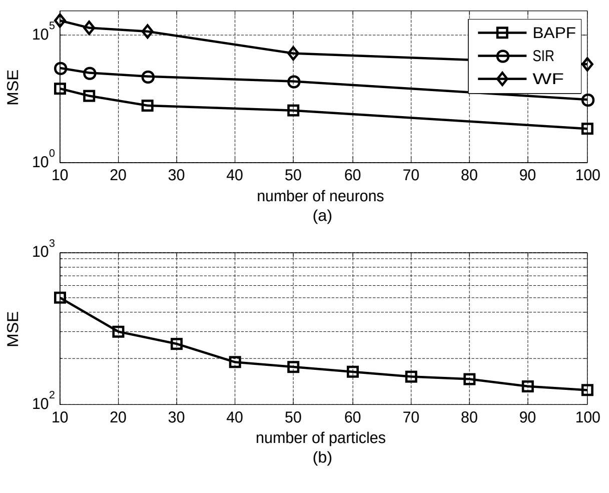

Fig. 1. Top- Comparison of MSE between the Weiner filter, SIR and BAPF. Bottom- Estimation accuracy improves as the number of particles increases.

The controller is modeled as a finite state machine (FSM). The FSM provides select lines (sel) to multiplexers that determine signal routing through the datapath. The FSM also regulates the status of enable (en) and reset (rst) signals of data registers. The RNG is a stand-alone dedicated module. It operates independent of all other units in the system. The RNG produces random variates according to the specific evolution of $\hat{\mathbf{x}}(t)$.

Each of the P particles used to estimate $s(t)$ are configured in parallel. The scalable design allows many particle processors to be replicated easily in hardware. They share the same control logic, which permits all particle processors to perform the same computations simultaneously.

B. Particle Processors

An expanded view of one particle processor from Figure 2 is shown in Figure 3. The *state vector estimate* block contains logic that stores an estimate of every element in $\hat{\mathbf{x}}^n(t)$. The number of neurons in the observed ensemble and the number of parameters being estimated per neuron determines the length of the state vector M. As M changes with the filter application, so does the FPGA hardware that supports the state vector.

Likewise, the conditional probabilities that define the likelihood equations will also vary with the BAPF application. For example, the simulated hippocampal place cell firings described in Section II are defined to arrive according to an inhomogeneous Poisson process with an expected arrival rate of $\lambda(t)$ defined by Equation 6. However, if a different tuning model is required for a different class of neurons, the *compute likelihood* submodule can be replaced with one to compute $\lambda(t)$ specific to their firing behavior.

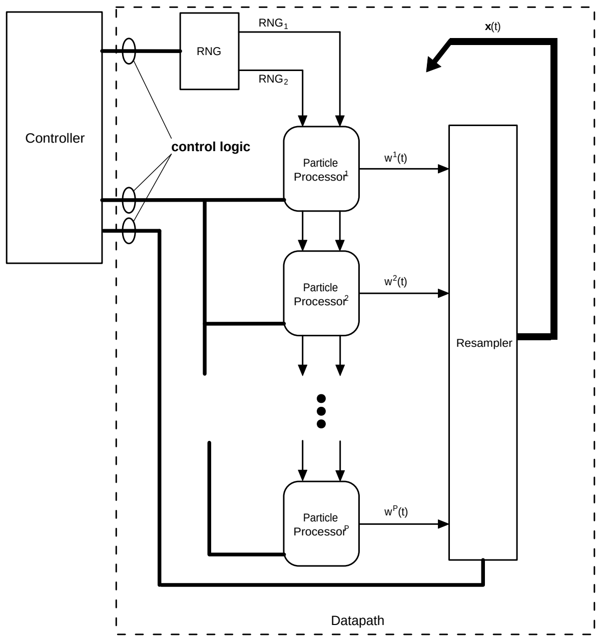

Fig. 2. Modular top-level diagram of the BAPF.

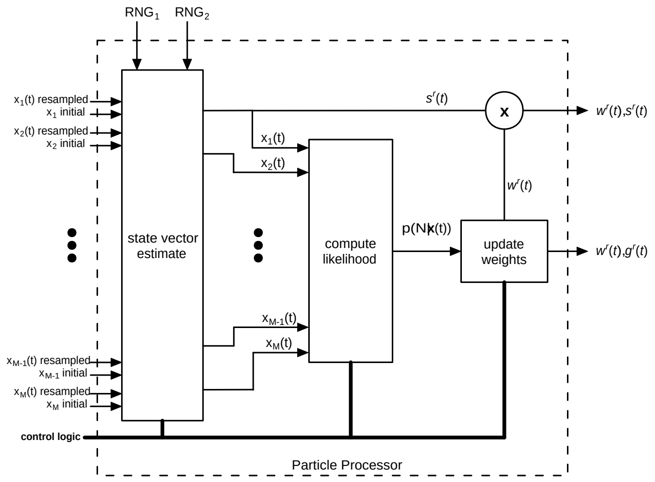

Fig. 3. Top-level diagram of a particle processor.

1) Parallel State Vector Update: Steps 1 and 4 of Section II require each particle to update its $\hat{\mathbf{x}}^n(t)$ by adding a random sample to each element of the state estimate. These random samples are stored along a tapped delay line which allows $\hat{\mathbf{x}}^n(t)$ to be formed as the sum of delayed samples and the previous estimate through a feedback loop. The RNG module of Figure 2 runs at a rate much higher than the rest of the datapath so all random variables are available for processing before executing Steps 1 and 4.

Figure 4 is an expanded view of the state vector module of Figure 3. Updating each element of $\hat{\mathbf{x}}^n(t)$ occurs simultaneously in all P particles of the BAPF. Steps 1 and 4 require a single clock each to execute.

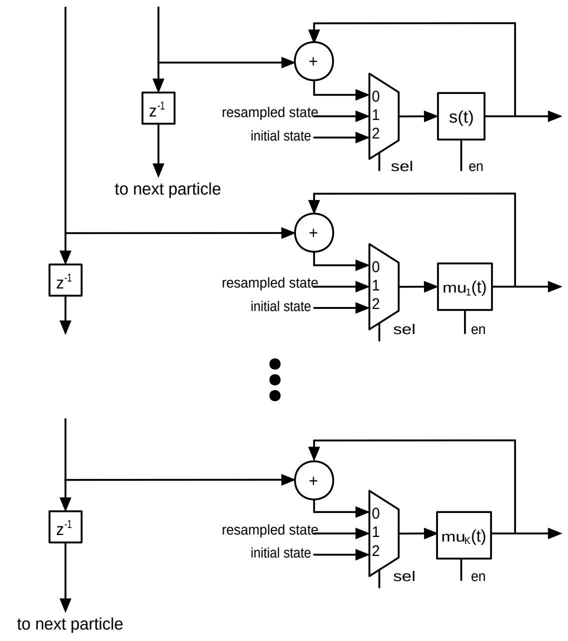

Fig. 4. Parallel registers provide simultaneous updates of all elements in $\hat{\mathbf{x}}^n(t)$.

2) Computing the Likelihood: The likelihood function of Equation 7 for the n^{th} particle is defined as the product of all neuronal likelihoods estimated by the n^{th} particle. This first involves the computation of two exponential functions for each neuron in parallel. Then, multiplying all of the individual neuronal likelihoods. A fast hardware approximation of e^x is employed to compute the exponential in only 10 clock cycles. Using parallel multipliers, the final product of neuronal likelihoods is computed in $\lceil log_2(K) \rceil$ clock cycles. Normalization is executed in $\lceil log_2(P) \rceil$ clock cycles through parallel additions.

3) Random Sampling and Final Weighted Estimate: The resampler of Figure 2 uses the cumulative sum of the normalized first stage weights as the bin edges of a histogram. A uniform random sample on the interval $(0, 1)$ is generated for each particle. The uniform samples of all particle are then sorted into their appropriate bin, which determines the particle number it replicates into its state vector. The parallel hardware design of the resampling process of Step 3 requires $P+2$ clock cycles to resample all P particles.

During Step 6 each particle multiplies its current estimate of $\hat{s}^n(t)$ by its normalized second stage weight $\hat{w}^n(t)$. These products are summed using parallel additions. Computing the weighted estimate requires $\lceil log_2(P) \rceil + 2$ clock cycles.

IV. Results

To demonstrate the increase in throughput facilitated by the FPGA architecture of the BAPF, execution times required to obtain an estimate from a single iteration of the BAPF algorithm in hardware and software are compared. Table I summarizes the number of clock cycles required to execute a single iteration of the BAPF for predicting animal position. In this experiment, $\Delta t = 1$ ms, $K = 75$ and P is varied from 50 to 5000. The FPGA hardware clock is 500 MHz. It is compared to the execution time required by a sequential Intel Core2 Duo Pentium processor with a 2.5 GHz clock in Figure 5. The hardware processing times were determined by Table I. The software throughput was determined by computing the average of 20 trials of each configuration.

TABLE I
Throughput latency defined by the number of clock cycles required to execute a single iteration of the BAPF

	Number of Clock Cycles
Step 1	1
Step 2	$\lceil log_2(K) \rceil + \lceil log_2(P) \rceil + 20$
Step 3	$P + 2$
Step 4	1
Step 5	$\lceil log_2(K) \rceil + \lceil log_2(P) \rceil + 21$
Step 6	$\lceil log_2(P) \rceil + 2$
Total	$P + 2\lceil log_2(K) \rceil + 3\lceil log_2(P) \rceil + 47$

Given the sampling period $\Delta t = 1$ ms, it is imperative that the computation time required to execute an iteration of the BAPF be less than Δt. If an iteration can not be computed within this duration, then real-time implementation can not be employed. Figure 5 shows that as the number of particles approaches approximately 200 the software computation time exceeds Δt. However, the hardware computation time remains well below Δt even as the number of particles is increased to 5000.

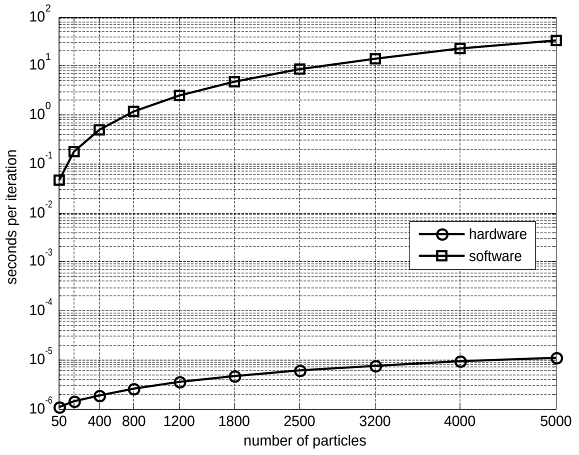

Fig. 5. Throughput comparison between hardware and software implementations of the BAPF as a function of the number of particles

V. Conclusion

The parallel hardware implementation presented greatly reduces execution time of the BAPF. The modular architecture provides a means for rapid prototyping of BAPFs for a variety of applications and observed neural ensembles. The scalable design allows the size of the neural ensemble to be changed as wells as the number of particles in the filter. The parallel FPGA architecture presented provides sufficient computational throughput for utilizing the BAPF in a real-time BMI.

References

[1] J. Sanchez and J. Pricipe, *Brain-Machine Interface Engineering*. Princeton, New Jersey: Morgan and Claypool, 2007.

[2] B. Ristic, S. Arulampalam, and Neil, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Norwood, Massachusetts: Artech House, 2004.

[3] J. Candy, *Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods*. Hoboken, New Jersey: Wiley and Sons, 2009.

[4] J. Liu and M. West, "Combined parameter and state estimation in simulation-based filtering," in *Sequential Monte Carlo Methods in Practice. New York* (A. Doucet, J. Freitas, and N. Gordon, eds.), Springer-Verlag, New York, 2000.

[5] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *Radar and Signal Processing, IEE Proceedings F*, vol. 140, pp. 107–113, Apr 1993.

[6] R. Esposito, J. Mountney, L. Bai, and D. S. , "Parallel architecture implementation of a reliable (k,n) image sharing scheme," in *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*, pp. 27–34, Dec. 2008.