An Energy-Efficient Kalman Filter Architecture with Tunable Accuracy for Brain-Computer Interfaces

Guy Eichler
Dept. of Computer Science
Columbia University
New York, New York, USA
guyeichler@cs.columbia.edu

Joseph Zuckerman
Dept. of Computer Science
Columbia University
New York, New York, USA
jzuck@cs.columbia.edu

Luca P. Carloni
Dept. of Computer Science
Columbia University
New York, New York, USA
luca@cs.columbia.edu

***Abstract*—Kalman Filter (KF) is the most prominent algorithm to predict motion from measurements of brain activity. However, little effort has been made to specialize KF hardware for the unique requirements of embedded brain-computer interfaces (BCIs). For this reason, we present the first configurable KF hardware architecture that enables fine-grained tuning of latency and accuracy, thereby facilitating specialization for neural data processing in BCI applications and supporting design-space exploration. Based on our architecture, we design KF hardware accelerators and integrate them into a heterogeneous system-on-chip (SoC). Through FPGA-based experiments, we demonstrate an energy-efficiency improvement of 15.3x and 10^3x better accuracy compared to state-of-the-art implementations.**

***Index Terms*—Kalman Filter (KF), System-on-Chip (SoC), Accelerator Design, Brain-Computer Interface (BCI), FPGA.**

I. Introduction

Real-world brain-computer interfaces (BCIs) establish a connection between the brain and the outside world to improve quality of life [1]–[3]. They rely on high-resolution neural interfaces, high-throughput data acquisition, and real-time computation [4]–[8]. While resolution and throughput continue to grow exponentially with the increasing number of electrodes in neural interfaces [2], [5], [6], [9], [10], achieving a wireless, implant-based BCI system that supports mobility, real-time processing, and low power consumption remains a challenge.

In this work, we focus on the Kalman Filter (KF), the most widely used online algorithm for motion prediction in the BCI field [11]–[16]. The KF estimates movement from neural signals and outputs real-time predictions of the position and velocity of specific body parts. These predictions are used to bypass the central nervous system and actuate a prosthesis or move a body part [17], [18]. However, BCI research has primarily focused on improving KF accuracy using machine learning, with less attention to its integration in real-world BCI systems [13], [14], [19]–[21]. Moreover, existing hardware implementations of the KF are not tailored for BCI use [22]–[25]. They rely on general-purpose techniques that are not optimized for processing diverse, high-dimensional neural data [2].

For this reason, we introduce KalmMind, a configurable architecture for agile development and design-space exploration of KF hardware accelerators tailored to BCI applications.

Fig. 1 provides a high-level overview of a KalmMind accelerator in a BCI system. The KF computation is offloaded from the implanted chip to an external device, referred to as a *relay station*, via wireless transmission of neural data [5], [26]. The relay station must be: *(1)* mobile, to support user movement; *(2)* low power, to operate within body-area networks (BAN) [27]; and *(3)* real-time, to maintain prediction relevance [28].

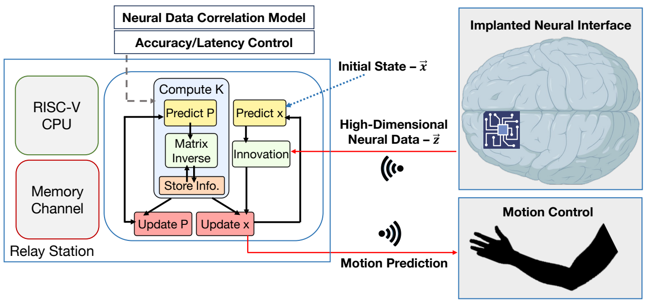


Fig. 1: KalmMind-based hardware accelerator in a BCI system.

To this end, KalmMind optimizes the main bottleneck of the KF: matrix inverse computation. It introduces a technique based on the Newton-Raphson method [29], [30] to control computational intensity by leveraging spatio-temporal correlations in neural activity across consecutive KF iterations [2], [3].

We demonstrate how KalmMind offers a flexible hardware architecture for thorough design-space exploration (DSE), aiming to minimize latency and maximize accuracy across neural datasets from different sources. We integrate additional approximation techniques inside the KF and implement a large group of configurable KF hardware accelerators [22], [31], [32].

We evaluate these accelerators by integrating them into a complete SoC that includes a RISC-V CVA6 processor [33] and prototyping the SoC on FPGA.

Our experiments show that KalmMind accelerators meet the real-time and low-power requirements of BCI systems. With its high design flexibility and fine-grained control over latency and accuracy, KalmMind opens up new opportunities for real-time KF predictions in embedded BCI systems.

We break down our main contributions as follows:

1) A reorganization of the KF algorithm that enables a modular and configurable hardware architecture.
2) A specialized approximation technique designed for use within a KF hardware accelerator in the BCI system.
3) A detailed design-space exploration to identify Pareto-optimal configurations in terms of latency and accuracy.
4) The implementation of KF accelerators that embed various approximation techniques and a comparative analysis.
5) The integration of KF hardware accelerators in a Linux-capable heterogeneous SoC, deployment on FPGA, and application to the processing of real brain data.

```
function KALMAN_FILTER($F, Q, H, R, \vec{x}_{n-1}, P_{n-1}, \vec{z}_n$)
    //Predict
    $\vec{x}_n = F \cdot \vec{x}_{n-1}$
    $P_n = F \cdot P_{n-1} \cdot F^t + Q$
    //Update
    $\vec{y} = \vec{z}_n - (H \cdot \vec{x}_n)$ //Innovation
    $S = H \cdot P_n \cdot H^t + R$
    $K = P_n \cdot H^t \cdot S^{-1}$ //Compute Kalman Gain
    $\vec{x}_n = \vec{x}_n + K \cdot \vec{y}$
    $P_n = (I - K \cdot H) \cdot P_n$
    return $\vec{x}_n, P_n$
```

Fig. 2: The Kalman Filter algorithm.

II. BACKGROUND

Computation in the BCI System. Online BCI applications compute over high-dimensional data that streams in from multiple *channels*. In invasive BCIs, these channels are implanted micro-electrodes that record electrical activity of localized groups of neurons [4]–[6]. While using invasive BCI systems is the most promising approach to improve the resolution of neural data [3], [4], running BCI applications close to the source becomes a very challenging task due to harsh biological constraints. In order to prevent cellular damage to the brain tissue, the power consumption of the implanted chip cannot increase the surrounding temperature by more than $1-2$°C [34].

For this reason, BCI systems have been decomposed into two main components: an implanted chip and an external relay station [5], [26]. The implanted chip records neural data and communicates it wirelessly to the relay station. The relay station executes applications in real time and can consume only up to $\sim 200mW$ within the BAN [7], [27], [35]. These BCI applications include machine learning (ML) algorithms that decode the neural data and classify it according to the desired application [13], [14], [36]. For example, BCI applications can be used in the fields of vision [37], authentication [38] and most commonly in motion decoding [13], [14], [16].

Motion Decoders for BCI. Traditionally, BCI applications of motion decoding from neural data have utilized linear methods for predicting kinematics (e.g., the Kalman Filter), which provide stable but limited accuracy [13], [39]. New methods of motion decoding use non-linear deep neural networks [40]–[42]. While these methods can provide high accuracy in some cases, they can impose long training times and also suffer from overfitting, due to the high complexity and variability of neural data [13], [40]. As a result, another group of decoders is emerging. These decoders use linear methods, such as the Kalman Filter (KF), as the main decoding component in combination with modern non-linear ML techniques to improve accuracy [14], [19], [43]–[45]. In this way, these decoders not only utilize the robustness of well-proven algorithms like the KF but also have the improved accuracy of non-linear methods.

The Kalman Filter Algorithm. The KF is a recursive algorithm that uses a series of measurements over time to predict the state of desired variables [16]. Fig. 2 reports the main function that is executed at each time step of the algorithm. Each step outputs a prediction state vector ($\vec{x}_n$), which holds a value for each of the desired variables, and the updated covariance matrix (P_n), which estimates the current accuracy of the results.

We use x as the dimension of the state vector and z as the dimension of the measurement vector. The KF algorithm receives 5 matrices as inputs: *(i)* the previous $P_{x\times x}$, *(ii)* $F_{x\times x}$ – the state transition model, which represents the probability of transitioning from one state to another, *(iii)* $Q_{x\times x}$ – the process noise covariance, which determines the uncertainty in the transition between states, *(iv)* $H_{z\times x}$ – the observation model over the measurements, which models the relationship between the different states and the observations made, and *(v)* $R_{z\times z}$ – the observation noise covariance, which describes the error in the measurements. In the traditional KF, the matrices F, Q, R, H remain constant between consecutive iterations of the KF and constitute the *KF model*. At each iteration, the KF receives the previous $\vec{x}_{n-1}$ and a new measurement vector $\vec{z}_n$.

TABLE I: The Accuracy of the KF with Different Methods

Accuracy Metric	Gauss [50]	IFKF [23]	Taylor [22]	SSKF [31]	Newton [29]
MSE	3.8×10^{-12}	53.8	0.05	0.1	6.6×10^{-6}
MAE	7×10^{-7}	2.7	0.08	0.06	0.0004
*Max. Difference (%)	0.008	2.2×10^{4}	9.7×10^{2}	5.3×10^{2}	4
*Avg. Difference (%)	0.0001	350	9	4.8	0.035

*These scores are normalized with respect to the original KF output [13].

The KF executes the "predict" step and then the "update" step. Calculating the Kalman gain K by inverting the matrix $S_{z\times z}$ lies at the core of the KF computation.

Kalman Filter Design for BCI. With the increase in the number of channels in BCIs [2], the size of the measurements increases. Consequently, the KF requires large matrix operations, particularly the inversion of a large matrix. For edge devices, this requirement is often addressed by a hardware KF implementation [24], [25], [46]. In many cases, the KF incorporates an approximation method, sacrificing accuracy to meet throughput and power constraints [22], [23], [32]. We aim to design a hardware accelerator that introduces no more than $\sim$10% error compared to the standard KF, ensuring adequate control over fine motor tasks with an embedded BCI [47]–[49].

TABLE I reports KF accuracies when integrated with several candidate computation techniques from literature, which we implemented in software. For all methods, the KF predicts motion based on neural data from the brain of a non-human primate (NHP) [13], [51] and runs for 100 KF iterations. The KF output is compared to the one provided by Glaser *et al.* [13].

Gaussian elimination (Gauss) [50] is the standard method for matrix inversion [8], [24], [25] and provides the most accurate result, as it calculates the matrix inverse directly. However, Gauss suffers from high complexity ($O(n^3)$) and internal dependencies that limit parallel processing. Inverse Free KF (IFKF) [23], Taylor expansion of K (Taylor) [22], Steady-State KF (SSKF) [31] and the Newton-Raphson method (Newton) [29] offer different approximation techniques of K or S^{-1}. IFKF provides the worst accuracy, because it requires dimensionality reduction of the measurements and assumes minimal cross-correlation, despite the high correlation in simultaneous neural data measurements [2], [3]. Other methods provide sufficient accuracy when tested on neural data. Nevertheless, Newton provides the best accuracy among these approximations. Our work is the first to embed Newton inside the KF. In Section III, we explain our approach in detail.

III. KALMMIND

Our goal is to design KF hardware accelerators that are specialized for the requirements of an embedded BCI system.

Our architecture offers design flexibility and control over the potential trade-off between computational accuracy and latency.

Reorganization of the KF Algorithm. Traditionally, the KF begins by computing initial predictions of $\tilde{x}_n$ and P_n, then updates them with additional information from the KF model at each iteration (Fig. 2). While the main bottleneck of the KF is the computation of K (line 8), K is independent from the current $\tilde{x}_n$, the current $\tilde{z}_n$ and the innovation y (line 6). K only depends on the initial prediction of P_n (line 4) and S (line 7). As a result, we reorganize the KF in order to leverage parallel processing and allow more flexibility for hardware design.

In Fig. 1, we identify the main modules of the KF and their dependencies, then isolate the computation of K (compute K), enabling its easy modification with different computation techniques. Specifically, since K is independent from the input measurements, new measurements can be processed in parallel to the compute K module. In addition, we can switch between different matrix inversion modules. In some cases, we can even pre-compute K or S^{-1}, pre-load them into the memory of the device, and avoid their online computation [31]. Since our KF is designed for BCI, we propose a technique to efficiently compute the inverse matrix by storing and propagating information from earlier iterations of processed neural data.

Calculation of the Matrix Inverse. We define *calculation* as directly calculating the matrix inverse without using any approximations. Gauss is the standard calculation method for the matrix inverse. While generally accurate, it relies on floating-point divisions, which can introduce numerical errors. Moreover, each element in the inverse depends on the calculation of other elements, limiting the ability to decompose the calculation and exploit parallel processing. Alternative methods such as LU factorization [52], Cholesky decomposition (Cholesky) [53] and QR decomposition (QR) [54] have been proposed. These methods can reduce both the number of operations and the numerical errors. However, they still demonstrate dependencies between operations and experience increased memory usage.

Matrix Inverse Approximation. Reducing computational latency is typically achieved by using *approximation* techniques [22], [23], [55]. Approximation requires fewer computational steps and may reduce dependencies between operations, thereby facilitating more parallel processing.

The Newton-Raphson method (Newton) is one of the oldest methods for approximating the matrix inverse [29], [56]. As it requires simple matrix multiplications, it allows for parallel processing of the computation [30]. The method is recursive, and increasing the recursion depth with more iterations improves the approximation of the inverse [57]. Specifically, an iterative approximation of a matrix inverse is given by:

$$V_{i+1} = f\_approx(V_i, A) \tag{1}$$

where $f\_approx(\cdot)$ computes an approximated matrix inverse V_{i+1} from the previous approximation V_i and the matrix to invert A. The Newton method defines the iterative process as:

$$V_{i+1} = V_i \cdot (2I - A \cdot V_i), \quad i = 0, 1, \ldots, m-1 \tag{2}$$

where the iterative process executes m iterations and V_m is the final output of the process. Since it does not involve divisions, approximation is less prone to numerical errors.

For the Newton method to converge after a minimal number of iterations, it is crucial to select a reliable *initial seed*. The initial seed V_0 must comply with the constraint:

$$\|I - A \cdot V_0\|_2 < 1 \tag{3}$$

which means that the seed is not too far from the optimal A^{-1}.

Combining Calculation and Approximation. Neural datasets are highly diverse, as they are recorded from different brain regions and neural interfaces, each requiring varying levels of computational accuracy and real-time performance.

We propose a new technique, which interleaves two different methods between consecutive KF iterations: one for calculation of the inverse and another for approximation of the inverse. The advantage of our technique lies in balancing a highly accurate but costly calculation with a faster, less accurate approximation.

At each iteration of the KF, one of the two methods is selected. The frequency of calculating the inverse is set by a user-configuration parameter (*calc_freq*). The inverse is calculated at every n-th iteration of the KF where $n \ \% \ calc\_freq = 0$.

In approximation, S_n is the matrix to invert at the n-th iteration of the KF (Fig. 2). The choice of V_0 is based on one of the two policies we develop, which initialize the approximation using a matrix inverse computed at a previous KF iteration. We propose two *seed policies* as follows:

$$V_{1,n} = S_{n-1}^{-1} \cdot (2I - S_n \cdot S_{n-1}^{-1}) \tag{4}$$

$$V_{1,n} = S_j^{-1} \cdot (2I - S_n \cdot S_j^{-1}), \quad j = n - n \ \% \ calc\_freq \tag{5}$$

where $V_{1,n}$ is the result of the first internal iteration of inverse approximation in the n-th iteration of the KF. Equation (4) defines a straightforward policy for initializing the approximation with $V_0 = S_{n-1}^{-1}$, the matrix inverse from the previous KF iteration (n-1). Equation (5) defines a policy where only a calculated inverse can serve as a seed, mitigating potential inaccuracies inherent in an approximated inverse. It sets $V_0 = S_j^{-1}$ where j is the last iteration that has utilized an inverse calculation.

These policies are particularly effective for BCI, as they utilize the strong temporal and spatial correlations between neural data measurements, leveraging an inverse matrix computed for previous measurements to approximate the inverse matrix for current measurements [2], [3].

Overall, using approximation with a small number of recursive iterations is expected to yield lower accuracy but improved latency. Frequent use of calculation should enhance accuracy with higher latency. Nonetheless, calculation may introduce numerical errors, potentially degrading accuracy compared to approximation with a high count of inner iterations.

IV. Configurable Hardware Acceleration

High-Level Architecture. The KalmMind accelerator consists of three main functions: *load*, *compute* and *store* (Fig. 3a). It exposes 7 memory-mapped configuration registers that control its communication with main memory and, in particular, the matrix operations executed by the *compute* function (Fig. 3b).

The x_dim and z_dim registers configure the dimensions of the matrices (F, Q, H, R, P) and the sizes of the vectors $(\vec{x}, \vec{z})$ to be expected by the accelerator. The chunks and batches registers configure the number of KF iterations to be done in one invocation of the accelerator and provide fine-grained control of the number and size of its direct-memory access (DMA) transactions. The approx, calc_freq and policy registers control the dataflow of computation inside the accelerator by selecting the data path used to invert S at each iteration of the KF (Fig. 3b). Specifically, calc_freq sets the frequency (in terms of KF iterations) of calculating the inverse; approx determines how many internal iterations of approximation will be executed per KF iteration; policy sets the seed policy according to Equation (4) or Equation (5).

Communication and Local Memories. The *load* function loads F, Q, H, R and the initial $\vec{x}_0, P_0$ from main memory (Section II) and stores them in private local memories (PLMs) inside the accelerator [58]. The PLMs are implemented as multi-bank memories that expose multiple read/write ports; their size is configurable at design time. The matrices F, Q, H, R can be reused in consecutive iterations of the KF without reloading them from main memory. For each subsequent DMA transaction, chunks configures the number of measurement vectors to be loaded from main memory. *load* receives (chunks×z_dim) measurements and stores them in the PLM. The *store* function sends the computed state vectors $(\vec{x}_n)$ and the state covariance matrices (P_n) to main memory. The batches register configures the total number of DMA transactions to be executed in one invocation of the accelerator.

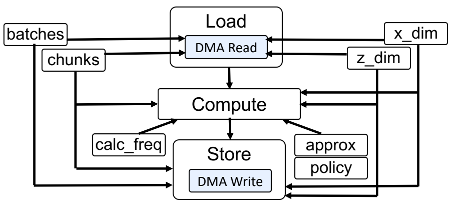
KF Computation. Fig. 3b shows how the *compute* function implements the KF with blue arrows to mark the computation path of the Kalman gain K. The state vector $\vec{x}_n$ is small, as it represents motion kinematics (Section II); hence, every computation with dimension x_dim is relatively short. As a result, we chose to implement these computations with the objective of reusing hardware resources instead of increasing throughput. With the exception of the matrix inverse, all matrix operations in Fig. 3b are fully pipelined. The inner-most accumulation loops are not unrolled in order to save hardware resources.

At each KF iteration, *compute* uses a double-buffer to store both the previous and new $\vec{x}_n$ and P_n (Fig. 3b). The double-buffers are swapped at the end of every iteration. The total number of iterations in one invocation is set to (batches×chunks).

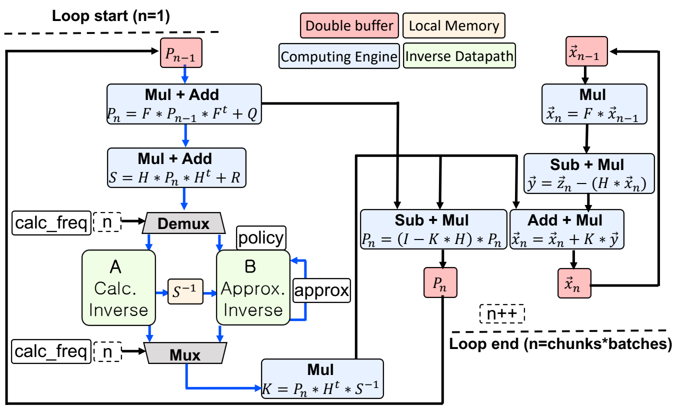
Thanks to our modular architecture, we are also able to replace the computation of K at design time with an approximation technique (Taylor) or a constant K (SSKF) and easily change the datatype between floating-point and fixed-point [32].

Implementation of the Matrix Inverse. At each KF iteration, a matrix inversion method is chosen according to the current KF iteration (n) and the configuration of calc_freq.

Path A in Fig. 3b implements a calculation method for matrix inversion. We refactor the computation to minimize the number of read/write interactions with the PLM and to be able to fully pipeline the computation of the inner-most loop. In Section V, we implement this path with Gauss, Cholesky and QR. We also replace it with a pre-computed constant S^{-1} (SSKF Inverse),



(a) Main accelerator functions.



(b) The compute function.

Fig. 3: A configurable KF hardware accelerator architecture.

inspired by the work of Malik *et al.* [31].

Path B in Fig. 3b implements an approximation method. For most of our accelerators, we implement the Newton method. We use 8 multiply-and-accumulate (MAC) units in parallel to accelerate the computation of matrix multiplications. In its first internal iteration, path B uses the seed specified by policy. If set to 0, the policy is set to Equation (5) and the most recently calculated inverse from path A is used as the seed. If set to 1, the policy is set to Equation (4) and the inverse from the previous KF iteration is used instead. Path B performs a fixed number of internal iterations as set by approx.

Overall, path B is expected to provide a better throughput compared to path A when approx is small, while the accuracy of path A per iteration should be higher. When approx grows larger, the accuracy is improved and can surpass that of path A, due to numerical errors in the inverse calculation. Interleaving the two paths in different ways offers various combinations of latency and accuracy in the KF.

V. Experimental Evaluation

Methodology: Three datasets of electrocorticography neural data from distinct brain regions are tested: the motor cortex of a non-human primate (NHP) [51], the somatosensory cortex of an NHP [42], and the hippocampus of a rat [59]. These datasets were originally tested with a KF using an inverse method that combines Gauss with LU factorization implemented with Python NumPy [13]. NumPy is the *reference* implementation. When a pure Gauss is employed in all KF iterations, it represents the *baseline*. Hardware accelerators are designed in C/C++ with Vivado HLS 2019.2. We name the accelerators according to their embedded calculation/approximation methods or with a designated name. The accelerators use 32-bit floating-point data types unless specified otherwise.

TABLE II: Accuracy Ranges with Three Neural Datasets

	MSE	MAE	Max Diff.
Motor	2.1×10^{-13}–1.1×10^{-6}	2×10^{-7}–1.6×10^{-4}	4.3×10^{-5}–1.91
Soma.	2.2×10^{-13}–9.9×10^{-6}	2.3×10^{-7}–5.1×10^{-4}	3.5×10^{-5}–5.3
Hippo.	3.1×10^{-11}–7.1×10^{-11}	1.2×10^{-6}–2.2×10^{-6}	8.2×10^{-5}–2.1×10^{-3}
Baseline	4.8×10^{-13}, 3×10^{-13}, 3.5×10^{-11}	2.7×10^{-7}, 2.7×10^{-7}, 1.4×10^{-6}	1.1×10^{-4}, 8.5×10^{-5}, 3.8×10^{-4}

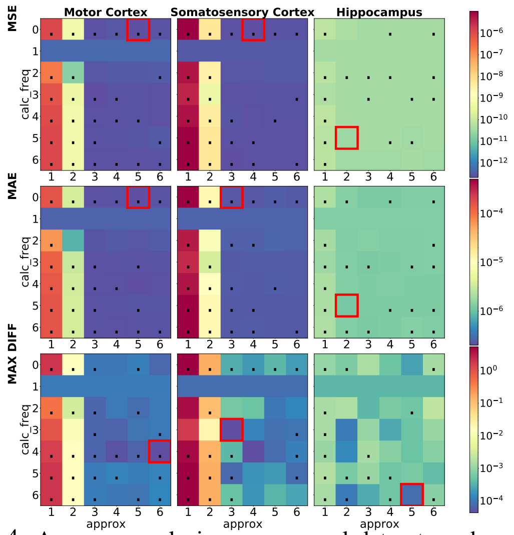


Fig. 4: Accuracy analysis across neural datasets and metrics.

SoC Integration: Hardware accelerators are synthesized with Vivado 2019.2 and leverage the open-source ESP platform [60]. ESP provides a tiled SoC architecture connected by a network-on-chip (NoC) with a library of heterogeneous components to facilitate "mix-and-match" SoC design. By following the accelerator integration flow [61], ESP is utilized for its implementations of DMA, memory-mapped registers, and interrupts that allow for seamless usage of an accelerator within a complete heterogeneous SoC architecture. ESP is also leveraged to generate FPGA prototypes of complete SoCs combining accelerators along with a 64-bit CVA6 RISC-V processor [33], an I/O tile, and a memory channel tile. The SoCs are tested on the Xilinx Virtex UltraScale XCVU440 FPGA board with a clock frequency of 78MHz, which is set according to the critical path of CVA6. We develop custom Linux software applications that run on the CVA6 and invoke the accelerators.

Accuracy Analysis. The predictions provided by the Gauss/Newton accelerator are analyzed with different configurations. The motor dataset dimensions are $\{x{=}6, z{=}164\}$, the somatosensory dataset dimensions are $\{x{=}6, z{=}52\}$, and the hippocampus dimensions are $\{x{=}6, z{=}46\}$. We run the accelerator in simulation for 100 iterations on each of the datasets. For `approx`, we use a range of 1-6, which means that Newton can have up to 6 internal iterations. For `calc_freq`, we use a range of 0-6 which means that Gauss can run in each iteration of the KF (`calc_freq=1`), every `calc_freq` iterations (`calc_freq=2−6`), or only at the first iteration (`calc_freq=0`). We compare the output at each KF iteration to the output from the reference and calculate three accuracy metrics: *(i)* Mean Square Error (MSE), *(ii)* Mean Absolute Error (MAE), and *(iii)* the normalized maximum difference between one output and its expected value (MAX DIFF).

Fig. 4 visualizes the accuracy results: purple denotes the highest accuracy while red denotes the lowest. For each pair of `calc_freq` and `approx`, we report the better result between

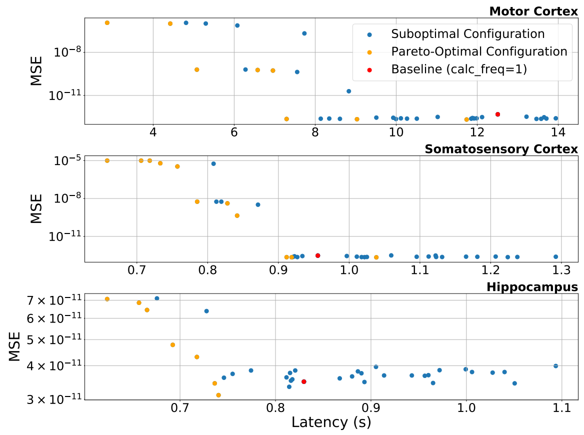


Fig. 5: Latency vs. accuracy with the Gauss/Newton accelerator.

the seed policies (`policy=1` is marked with a dot). For each metric and dataset, the highest accuracy is outlined in red.

The results show that, thanks to the flexibility of the accelerator, we can adjust the computation across a wide range of accuracies. TABLE II summarizes the accuracy ranges to which the accelerator can be configured for each dataset and metric. In all cases, we can even find a configuration that provides better accuracy than the baseline. This occurs because Newton can reduce the numerical errors introduced by Gauss by avoiding floating-point divisions. The best result is for MAX DIFF and the hippocampus, with a 78% improvement in accuracy.

It is important to note that, with the same set of configurations, each neural dataset achieves its best accuracy with a different configuration. In addition, the neural datasets from the NHP correspond to different accuracy ranges compared to the dataset from the rat. This highlights the importance of matching a neural dataset with a specific KF configuration to achieve sufficient accuracy and optimize performance. A configurable KF, such as the one we propose, offers this flexibility and fosters more effective and specialized BCI applications.

Accuracy vs. Latency. We combine the previous analysis with the latency of the Gauss/Newton accelerator when running on FPGA. Fig. 5 shows the results for each of the datasets with the MSE metric. The plots highlight Pareto-optimal points, each representing a specific tradeoff between accuracy and latency. The Pareto-optimal points result from different interleaving patterns of the two matrix inverse methods; for example, the point that provides the least latency in each plot has `approx=1` and `calc_freq=0`, while the point with the best accuracy in each plot has `approx`$\geq$`2`. In each plot, we can find configurations with comparable or better accuracy than the baseline with less latency and ultimately a better performance of the KF.

Overall, this analysis provides valuable insights for a KF hardware design tailored to specific application constraints.

Comparative Analysis of KF Implementations. We use KalmMind to design a set of hardware accelerators. TABLE III summarizes their FPGA resources, power, performance range, energy range, and accuracy range, measured while running 100 KF iteration on FPGA. It also includes the baseline implementation on an Intel i7 processor with a clock frequency of 3.7GHz, and on the CVA6 with a clock frequency of 78MHz (the same as the accelerators). We experiment with the largest

TABLE III: FPGA Resources and Performance across KF Implementations/Accelerators

Type	Method	LUT	FF	BRAM	DSP	Power [W]	Perf. [sec]	Energy [J]	Accuracy [MSE]
Software	Intel i7	N/A	N/A	N/A	N/A	78.6	0.065	5.1	3.8×10^{-12}
	CVA6	43996	29922	36	27	0.177	1927	341	1.3×10^{-12}
Hardware: Calc./Approx. Datapath	Gauss/Newton	22119	18725	228	252	0.185	$2.8-8.9$	$0.52-1.64$	$1.03\times10^{-12}-1.1\times10^{-6}$
	Cholesky/Newton	22429	20126	360	268	0.207	$2.8-11.5$	$0.58-2.38$	$1.05\times10^{-12}-1.1\times10^{-6}$
	QR/Newton	24842	21259	385	258	0.236	$3.04-9.6$	$0.72-2.27$	$1.02\times10^{-12}-1.1\times10^{-6}$
	Gauss/Newton FX32	19646	12131	195.5	217	0.146	$4.25-4.25$	0.354	$5.9\times10^{-2}-0.46$
	Gauss/Newton FX64	34831	26109	369	534	0.18	$2.44-11.3$	$0.44-2.04$	$1.9\times10^{-5}-0.24$
Hardware: One-way Datapath	LITE	15591	13405	146.5	193	0.114	2.688	0.306	1.14×10^{-6}
	LITE FX64	14782	8075	267	347	0.11	2.268	0.249	1.14×10^{-6}
	SSKF/Newton	18798	16961	204.5	240	0.158	$0.53-11.6$	$0.08-1.82$	$9.9\times10^{-13}-6.3\times10^{-5}$
	SSKF	8403	6752	19.5	102	0.051	0.03	0.0015	7.63×10^{-3}
	Taylor	15006	13437	118	230	0.155	1.203	0.186	2.3×10^{-3}
	Gauss-Only	12386	10290	102.5	153	0.098	12.507	1.225	1.3×10^{-12}

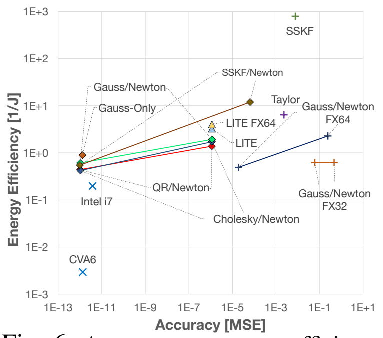


Fig. 6: Accuracy vs. energy efficiency.

dataset, the motor dataset, which requires a KF to run in less than $50ms$ per iteration for real-time BCI execution [28], [51].

One group of accelerators is implemented with unique calculation/approximation datapaths, using Gauss, QR, and Cholesky calculation methods alongside Newton approximation. The other group is implemented with one-way datapaths, where each accelerator uses either a calculation or approximation method: *(1)* LITE is designed to run Newton with one internal iteration and a seed from the previous KF iteration. In the first KF iteration, LITE loads a pre-computed seed from main memory. *(2)* SSKF embeds the method by Malik *et al.* [31]. The method is BCI-specific and approximates a constant K instead of the compute K module (Fig. 1). *(3)* In SSKF/Newton, we approximate a constant S^{-1}_{const}. We added the option to run Newton in order to improve the accuracy of S^{-1}_{const}. *(4)* Taylor integrates the method by Liu *et al.* [22], which approximates K at every KF iteration, avoiding the matrix inverse. *(5)* Gauss-Only always calculates the inverse with our optimized implementation of Gauss. *(6)* We provide accelerators with 64-bit (FX64) and 32-bit (FX32) fixed-point data types [32].

All accelerators, except Gauss-Only, demonstrate real-time execution by completing 100 KF iterations in under $5sec$. All accelerators meet the low-power constraint, consuming up to $\sim 200mW$. This proves that KalmMind successfully produces accelerators suitable for real-time embedded BCI systems.

Fig. 6 shows the tradeoff between accuracy and energy efficiency. The energy efficiency was calculated by inverting the energy result. Among the accelerators that use both calculation and approximation, Gauss/Newton has the best energy efficiency with a $10\times$ improvement compared to Intel i7 and $655\times$ compared to CVA6. SSKF provides the best energy efficiency with a $346\times$ improvement over Gauss/Newton. This is expected as SSKF is the only accelerator that does not compute K. However, the accuracy of SSKF is $10^9\times$ worse than Gauss/Newton and $10^3\times$ worse than LITE. Our SSKF/Newton offers the widest range of accuracy, achieving up to $15.3\times$ better energy efficiency compared to the standard Gauss (Gauss-Only).

Overall, KalmMind compensates for higher average power consumption, resulting from a more complex datapath, by offering substantially better energy efficiency and greater control over accuracy. KalmMind-based accelerators can be roughly divided into three tiers of accuracy. As accuracy decreases from the highest to the lowest tier, energy efficiency improves. Depending on the needs of the application, the neural data, and device constraints, the optimal KF accelerator can be chosen.

VI. Discussion and Related Work

The KF has been used frequently for motion decoding in BCI [13], [14], [16], [21], [31], [47], [62], [63]. The KF models we use to demonstrate KalmMind are trained according to the method of Wu *et al.* [16], and provided by Glaser *et al.* [13].

Usually, BCI applications utilize a Gauss-based KF [13], [14], [19]–[21]. They use the KF with additional ML models and continuously update the KF model. KalmMind can optimize the KF part of these applications for real-world embedded BCIs by managing accuracy, latency, and energy efficiency.

In Section II, we evaluate 4 different approximation techniques from literature [22], [23], [29], [31]. While using Newton [29] achieves the most accurate results for motion prediction from neural data, we use it in Section III to design a BCI-tailored technique that relies on propagating information between consecutive iterations of the KF. We are not aware of other implementations that use Newton within the KF. We suppose that this is because, prior to this work, no suitable seed policy has been identified. In addition, we design optimized accelerators for fixed-point datatypes [32], Taylor [22] and SSKF [31]; SSKF has been previously used for BCI [47], [62].

Other hardware implementations of the KF have been created to address computation in a mobile edge setting [8], [22], [24], [25], [32], [46], [64]–[67]. They are optimized for speed and low power consumption, but most are designed for domains other than BCI, have been tested with relatively small measurement vectors, and rely on costly calculations rather than approximations. However, SCALO [8] is an accelerator-rich distributed BCI system, where the inverse matrix for the KF is computed using Gauss. With KalmMind, they could leverage approximations with sufficient accuracy, reduce data dependencies, and enhance the scalability of the BCI system.

KalmMind is the first architecture to facilitate the design of configurable KF hardware accelerators for BCI, offering flexibility and uniquely supporting fine-grained control over latency and accuracy to address the diversity of brain data.

VII. Conclusion

We designed KalmMind to advance research on hardware architectures for embedded BCIs and to accelerate the development of practical, real-world BCI systems and specialized solutions for BCI applications. The contributions of this work have been released to the public domain[1].

[1]https://github.com/GuyEichler/KalmMind

References

[1] R. P. Rao, "Brain Co-Processors: Using AI to Restore and Augment Brain Function," *Handbook of neuroengineering*, 2020.

[2] A. E. Urai *et al.*, "Large-Scale Neural Recordings Call for New Insights to Link Brain and Behavior," *Nature neuroscience*, 2022.

[3] I. H. Stevenson *et al.*, "How Advances in Neural Recording Affect Data Analysis," *Nature neuroscience*, 2011.

[4] Y. Wang *et al.*, "Implantable Intracortical Microelectrodes: Reviewing the Present with a Focus on the Future," *MNE*, 2023.

[5] N. Zeng *et al.*, "A Wireless, Mechanically Flexible, 25μm-Thick, 65,536-Channel Subdural Surface Recording and Stimulating Microelectrode Array with Integrated Antennas," in *VLSI*, 2023.

[6] E. Musk *et al.*, "An Integrated Brain-Machine Interface Platform with Thousands of Channels," *JMIR*, 2019.

[7] G. Eichler *et al.*, "MasterMind: Many-Accelerator SoC Architecture for Real-Time Brain-Computer Interfaces," in *ICCD*, 2021.

[8] K. Sriram *et al.*, "SCALO: An Accelerator-Rich Distributed System for Scalable Brain-Computer Interfacing," in *ISCA*, 2023.

[9] W. Choi *et al.*, "A 1,024-Channel, 64-Interconnect, Capacitive Neural Interface Using a Cross-Coupled Microelectrode Array and 2-Dimensional Code-Division Multiplexing," in *VLSI*, 2023.

[10] J.-H. Cha *et al.*, "A Reconfigurable Sub-Array Multiplexing Microelectrode Array System with 24,320 Electrodes and 380 Readout Channels for Investigating Neural Communication," in *ISSCC*, 2022.

[11] C. Urrea *et al.*, "Kalman Filter: Historical Overview and Review of its Use in Robotics 60 Years after its Creation," *Journal of Sensors*, 2021.

[12] X. Gao *et al.*, "RL-AKF: An Adaptive Kalman Filter Navigation Algorithm based on Reinforcement Learning for Ground Vehicles," *Remote Sensing*, 2020.

[13] J. I. Glaser *et al.*, "Machine Learning for Neural Decoding," *Eneuro*, 2020.

[14] A. D. Degenhart *et al.*, "Stabilization of a Brain–Computer Interface via the Alignment of Low-dimensional Spaces of Neural Activity," *Nature biomedical engineering*, 2020.

[15] M. Śliwowski *et al.*, "Decoding ECoG Signal into 3D Hand Translation Using Deep Learning," *J. Neural Eng.*, 2022.

[16] W. Wu *et al.*, "Neural Decoding of Cursor Motion Using a Kalman filter," *Advances in neural information processing systems*, 2002.

[17] M. Vilela *et al.*, "Applications of Brain-Computer Interfaces to the Control of Robotic and Prosthetic Arms," *Handbook of clinical neurology*, 2020.

[18] A. Moly *et al.*, "An Adaptive Closed-Loop ECoG Decoder for Long-Term and Stable Bimanual Control of an Exoskeleton by a Tetraplegic," *J. Neural Eng.*, 2022.

[19] X. Zhang *et al.*, "Reinforcement Learning-Based Kalman Filter for Adaptive Brain Control in Brain-Machine Interface," in *EMBC*, 2021.

[20] EMBC, "A Stabilized Dual Kalman Filter for Adaptive Tracking of Brain-Computer Interface Decoding Parameters," 2013.

[21] V. Gilja *et al.*, "A High-Performance Neural Prosthesis Enabled by Control Algorithm Design," *Nature neuroscience*, 2012.

[22] Y. Liu *et al.*, "Efficient Mapping of a Kalman Filter into an FPGA Using Taylor Expansion," in *FPL*, 2007.

[23] K. S. Babu *et al.*, "Inverse Free Kalman Filter Using Approximate Inverse of Diagonally Dominant Matrices," *IEEE Control Systems Letters*, 2019.

[24] A. Valade *et al.*, "A Study about Kalman Filters Applied to Embedded Sensors," *Sensors*, 2017.

[25] M. Ordubağ *et al.*, "Model-Based Kalman Filter Design on an FPGA," in *ELECO*, 2021.

[26] Y. Gilhotra *et al.*, "A Wireless Subdural Optical Cortical Interface Device with 768 Co-Packaged Micro-LEDs for Fluorescence Imaging and Optogenetic Stimulation," in *CICC*, 2024.

[27] A. Y. Dogan *et al.*, "Multi-Core Architecture Design for Ultra-Low-Power Wearable Health Monitoring Systems," in *DATE*, 2012.

[28] J. P. Cunningham *et al.*, "A Closed-Loop Human Simulator for Investigating the Role of Feedback Control in Brain-Machine Interfaces," *Journal of neurophysiology*, 2011.

[29] A. Ben-Israel, "An Iterative Method for Computing the Generalized Inverse of an Arbitrary Matrix," *Mathematics of Computation*, 1965.

[30] V. Pan *et al.*, "Efficient Parallel Solution of Linear Systems," in *STOC*, 1985.

[31] W. Q. Malik *et al.*, "Efficient Decoding with Steady-State Kalman Filter in Neural Interface Systems," *IEEE TNSRE*, 2010.

[32] P. T. L. Pereira *et al.*, "Architectural Exploration for Energy-Efficient Fixed-Point Kalman Filter VLSI Design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2021.

[33] F. Zaruba *et al.*, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *ITVL*, 2019.

[34] P. D. Wolf *et al.*, "Thermal Considerations for the Design of an Implanted Cortical Brain–Machine Interface (BMI)," *Indwelling Neural Implants: Strategies for Contending with the In Vivo Environment*, 2008.

[35] G. Udovičić *et al.*, "Wearable Technologies for Smart Environments: A Review with Emphasis on BCI," in *SoftCOM*, 2016.

[36] J. Lee *et al.*, "Hierarchical Optimal Transport for Multimodal Distribution Alignment," in *NeurIPS*, 2019.

[37] A. Lozano *et al.*, "Neurolight: A Deep Learning Neural Interface for Cortical Visual Prostheses," *Int J Neural Syst*, 2020.

[38] L. Feng *et al.*, "Brain Password: A Secure and Truly Cancelable Brain Biometrics for Smart Headwear," in *MobiSys*, 2018.

[39] D. Sussillo *et al.*, "Making Brain–Machine Interfaces Robust to Future Neural Variability," *Nature communications*, 2016.

[40] F. Liu *et al.*, "Deep Learning for Neural Decoding in Motor Cortex," *Journal of Neural Engineering*, 2022.

[41] L. Yao *et al.*, "Fast and Accurate Decoding of Finger Movements from ECoG through Riemannian Features and Modern Machine Learning Techniques," *Journal of Neural Engineering*, 2022.

[42] A. S. Benjamin *et al.*, "Modern Machine Learning as a Benchmark for Fitting Neural Responses," *Front. Comput. Neurosci.*, 2018.

[43] M. Asgharpour *et al.*, "Regularized Kalman Filter for Brain-Computer Interfaces Using Local Field Potential Signals," *J. Neurosci. Methods*, 2021.

[44] F. R. Willett *et al.*, "Principled BCI decoder design and parameter selection using a feedback control model," *Scientific reports*, 2019.

[45] G. Revach *et al.*, "KalmanNet: Neural Network Aided Kalman Filtering for Partially Known Dynamics," *IEEE Trans. Signal Process.*, 2022.

[46] P. Babu *et al.*, "FPGA Implementation of Multi-Dimensional Kalman Filter for Object Tracking and Motion Detection," *JESTECH*, 2022.

[47] Z. Irwin *et al.*, "Neural Control of Finger Movement via Intracortical Brain–machine Interface," *J. Neural Eng.*, 2017.

[48] A. K. Vaskov *et al.*, "Cortical Decoding of Individual Finger Group Motions Using ReFIT Kalman Filter," *Frontiers in neuroscience*, 2018.

[49] D. Shin *et al.*, "Prediction of Muscle Activities from Electrocorticograms in Primary Motor Cortex of Primates," *PloS one*, 2012.

[50] N. J. Higham, "Gaussian Elimination," *Wiley Interdisciplinary Reviews: Computational Statistics*, 2011.

[51] J. I. Glaser *et al.*, "Population Coding of Conditional Probability Distributions in Dorsal Premotor Cortex," *Nature communications*, 2018.

[52] J. Dongarra *et al.*, "High Performance Matrix Inversion Based on LU Factorization for Multicore Architectures," in *MTAGS*, 2011.

[53] A. Krishnamoorthy *et al.*, "Matrix Inversion Using Cholesky Decomposition," in *SPA*, 2013.

[54] A. Irturk *et al.*, "An Efficient FPGA Implementation of Scalable Matrix Inversion Core using QR Decomposition," 2009.

[55] V. V. Vazirani, *Approximation Algorithms*. Springer, 2001.

[56] G. Schulz, "Iterative berechung der reziproken matrix," *ZAMM-Journal of Applied Mathematics and Mechanics*, 1933.

[57] F. Toutounian *et al.*, "An Iterative Method for Computing the Approximate Inverse of a Square Matrix and the Moore–Penrose Inverse of a Non-Square Matrix," *Appl. Math. Comput.*, 2013.

[58] C. Pilato *et al.*, "System-Level Optimization of Accelerator Local Memory for Heterogeneous Systems-on-Chip," *TCAD*, 2017.

[59] K. Mizuseki *et al.*, "Multi-Unit Recordings from the Rat Hippocampus Made During Open Field Foraging," *CRCNS. org*, 2009.

[60] P. Mantovani *et al.*, "Agile SoC Development with Open ESP," in *ICCAD*, 2020.

[61] D. Giri *et al.*, "Accelerator Integration for Open-Source SoC Design," *IEEE Micro*, 2021.

[62] B. Jarosiewicz *et al.*, "Advantages of Closed-Loop Calibration in Intracortical Brain–Computer Interfaces for People with Tetraplegia," *J. Neural Eng.*, 2013.

[63] M. Asgharpour *et al.*, "Regularized Kalman filter for brain-computer interfaces using local field potential signals," 2021.

[64] D. Pritsker, "Hybrid Implementation of Extended Kalman Filter on an FPGA," in *RadarCon*, 2015.

[65] B. Xu *et al.*, "A Resource Saving FPGA Implementation Approach to Fractional Kalman filter," *IET Control Theory & Applications*, 2022.

[66] F. Sandhu *et al.*, "FPGA-Based Implementation of Kalman Filter for Real-Time Estimation of Tire Velocity and Acceleration," *IEEE Sens. J.*, 2017.

[67] J. Soh *et al.*, "An FPGA-Based Unscented Kalman Filter for System-on-Chip Applications," *IEEE Trans. Circuits Syst. II: Express Briefs*, 2016.