Optimal FPGA Implementation of Dense Extended Kalman Filter for Simultaneous Cell State Estimation

Luke Nuculaj, Adam Kidwell, Connor Homayouni, Alex Fillmore, Darrin Hanna, and Jun Chen
Department of Electrical and Computer Engineering
Oakland University
Rochester Hills, MI, USA
e-mail: {lukenuculaj | bkidwell | chomayouni | afillmore | dmhanna | junchen}@oakland.edu

***Abstract*—This work considers hybrid embedded implementations of the dense extended Kalman filter (DEKF) – a novel state estimation technique which reduces the traditional Kalman filter's time complexity from $\mathcal{O}(N^3)$ to $\mathcal{O}(N)$ – to estimate the state-of-charge (SOC) of 16 serially-connected lithiom-ion battery cells. Design space exploration (DSE) is used to recommend an optimal hybrid binding considering execution time and energy as the primary objectives. High-level synthesis (HLS) tools are used to inform the DSE and to aid in implementation. Furthermore, each implementation's performance is gauged and compared to our MATLAB model of the DEKF and our DSE. Our results indicate close precision and accuracy to MATLAB with RMS near 0.2% with execution times as low as 52 μs.**

I. Introduction

With the rising popularity of electric vehicles in recent years, it is important to consider the battery cells – typically lithium-ion [1], [2] – that power them. These cells are organized into groups and must be carefully monitored so as to inhibit the aging process of the battery pack, as well as to mitigate the possibility of thermal runaway caused by excessive charging/discharging. To quantify the level of charge left in each cell, state-of-charge (SOC) is the universally adopted metric. However, because lithium-ion cells have a complex internal chemistry, measuring the SOC is a challenging feat. Kalman filtering provides us with the ability to instead estimate each cell's SOC given a pre-conceived mathematical model and voltage measurements. Applying the traditional Kalman filtering algorithm to the large-scale estimation problem is computationally inefficient. Instead, a novel variant of the extended Kalman filter – the dense extended Kalman filter (DEKF) [3] – is explored in this work, its computational efficiency verified by an optimal hybrid embedded system implementation, and its estimation performance validated by ground-truth simulation results.

Embedded implementations of Kalman filters for battery SOC estimation is an active area of research. Hong et. al. [4] introduce research exploring deployment of the Extended Kalman Filter (EKF) and the unscented Kalman Filter (UKF) on a Raspberry Pi with 1-12 cells. Their research employs a more sophisticated model for acquiring battery data. Their final results indicate runtimes in the order of 10^1 s and a mean absolute error of just below 3%. Cui et. al. [5] deploy the square root cubature Kalman filter on an embedded MCU for SOC estimation. They achieve an execution time of about 2.5 s with root mean square error reaching just below 0.7%. In Oehler et. al.'s paper [6], the authors pursue an embedded implementation of an EKF using a more accurate electrochemical battery model rather than an equivalent circuit model. They achieve a runtime in the order of 10^1 ms with a mean relative error of 1.5% or less. Our implementations of the DEKF are able to achieve run times in the order of 10^2 μs or less with RMS near 0.2%.

For our implementations, we utilize an XC7Z020CLG400-1 Xilinx FPGA containing an ARM microprocessor. Techniques such as nonlinear integer programming (NLIP) and probability density functions (PDF) are utilized for design space exploration (DSE) to determine an optimum partition in this hybrid system. We use high-level synthesis (HLS) tools to inform our DSE and to speed up implementation allowing us to efficiently compare two full implementations of different partitions.

II. Battery Cell State Estimation

A. Model of a Battery Cell

This work considers the case of N serially-connected battery cells, adopting a first-order equivalent circuit model (ECM) [7]–[9] to characterize a single cell's dynamics, where the open-circuit voltage (V_{oc}), open-circuit resistance (R_o), terminal voltage (y), total cell current (u), relaxation resistance (R_p) and relaxation capacitance (C_p) are all model parameters. Discrete-time cell dynamics are specified by

$$s^i_{k+1} = s^i_k - \eta^i \frac{T_s}{3600C^i} u^i_k \tag{1a}$$

$$V^i_{k+1} = \left(1 - \frac{T_s}{R^i_p C^i_p}\right) V^i_k + \frac{T_s}{C^i_p} u^i_k \tag{1b}$$

$$y^i_k = V^i_{oc,k} - V^i_k - u^i_k R^i_o. \tag{1c}$$

where the superscript i denotes the ith cell, s^i is the ith cell's state of charge (SOC), V^i is the ith cell's relaxation voltage, η^i is the Coulombic efficiency, C^i is the cell capacity with unit of Amp-hours, and u^i_k is the ith cell's total current at time step k.

Denoting $x^i := \begin{bmatrix} s^i & V^i \end{bmatrix}^T$, the linear state equation for a single cell can be compactly represented as

$$x^i_{k+1} = A^i x^i_k + B^i u^i_k, \tag{2}$$

$$A^i = \begin{bmatrix} 1 & 0 \\ 0 & 1 - \frac{T_s}{R^i_p C^i_p} \end{bmatrix} \quad B^i = \begin{bmatrix} -\eta^i \frac{T_s}{3600 C^i} \\ \frac{T_s}{C^i_p} \end{bmatrix}. \tag{3}$$

The measurement function (1c) is nonlinear, since the open circuit voltage V_{oc} is nonlinear as a function of SOC – typically stored in a lookup table [10].

Consider the collection of state update equations across all N cells

$$X_{k+1} := \begin{bmatrix} x^1_{k+1} \\ x^2_{k+1} \\ \vdots \\ x^N_{k+1} \end{bmatrix} = \begin{bmatrix} A^1 x^1_k + B^1 u^1_k \\ A^2 x^2_k + B^2 u^2_k \\ \vdots \\ A^N x^N_k + B^N u^N_k \end{bmatrix},$$

and define the sparse state vector as $X_k = \begin{bmatrix} x^1_k & x^2_k & \dots & x^N_k \end{bmatrix}^T$, which is the state vector for the entire battery pack at time step k. This allows for the state dynamics across the pack to be represented as

$$X_{k+1} = AX_k + BU_k, \tag{4}$$

where $A \in \mathbb{R}^{2N \times 2N}$ = $\mathbf{blkdiag}(\{A^i\}_{i=1}^N)$, and $B \in \mathbb{R}^{2N \times N}$ = $\mathbf{blkdiag}(\{B^i\}_{i=1}^N)$, and current matrix U_k is defined as $\begin{bmatrix} u^1_k & u^2_k & \dots & u^N_k \end{bmatrix}^T$. The measurement function, in the case of pack-level dynamics, is a scalar quantity representative of the pack's terminal voltage defined as

$$y_k = \sum_{i=1}^{N} \left(V^i_{oc,k} - V^i_k - u^i_k R^i_o \right) := h(X_k, U_k). \tag{5}$$

B. Dense Extended Kalman Filter

This section describes the process for estimating over N serial-connected battery cells with the EKF, which has a complexity of $O(N^3)$ since the sizes of the covariance matrices and vectors grow proportionally to N.

a) Sparse Prediction Model and Time Update: Because of the inherent deviations of real-life systems from mathematical models, (4) and (5) take on the form

$$\begin{aligned} X_{k+1} &= AX_k + BU_k + W_k \\ y_k &= h(X_k, U_k) + v_k, \end{aligned}$$

where $W_k := \begin{bmatrix} w^1_k & w^2_k & \dots & w^N_k \end{bmatrix}^T$, w^i_k ($\sim \mathcal{N}(0, Q^i_k)$) is the process noise of the the ith cell, and v_k ($\sim \mathcal{N}(0, R_k)$) is the measurement noise, the latter two satisfying zero-mean Gaussian distributions with covariances Q^i_k and R_k.

However, the state vector $X_k \in \mathbb{R}^{2N}$, meaning the predicted covariance matrix associated thereof contains $2N^2+N$ unique elements ($P_k = P^T_k \in \mathbb{R}^{2N \times 2N}$). Modern automotive-grade MCUs typically have RAM space in the range of 128 to 512 KB [11], [12]. For $N = 16$, the worst-case requires 9.6% of the total on-chip SRAM, and 468 μs of total execution time. As reported in [3], memory consumption grows quadratically in N, and computational latency grows cubically in N. While the sparse Kalman filtering algorithm is possible on automotive-grade MCUs, it is an energy-hungry algorithm inducing a large amount of memory overhead onto the system. [3] introduces the novel DEKF, which sees the memory consumption and computational latency growing linearly in N, removing nearly all of the overhead involved while maintaining estimation accuracy.

Algorithm 1 DEKF Algorithm: Time Update

procedure TIME($\hat{x}^+_{\mu,k}$, $P^+_{\mu,k}$, U_k, $\hat{X}^+_k$, k)
  $\Gamma_k \leftarrow$ computing (6);
  $\Gamma^\dagger_k \leftarrow$ computing pseudoinverse;
  $A_{\mu,k} \leftarrow \Gamma^\dagger_k A \Gamma_k$; $B_{\mu,k} \leftarrow \Gamma^\dagger_k B$;
  $\hat{x}^-_{\mu,k+1} \leftarrow A_{\mu,k} \hat{x}^+_{\mu,k} + B_{\mu,k} U_k$;
  $\hat{X}^-_{k+1} \leftarrow \hat{X}^+_k + \Gamma_k (\hat{x}^-_{\mu,k+1} - \hat{x}^+_{\mu,k})$;
  $P^-_{\mu,k+1} \leftarrow A_{\mu,k} P^+_{\mu,k} A^T_{\mu,k} + Q_\mu$;
return $\hat{x}^-_{\mu,k+1}, P^-_{\mu,k+1}, \hat{X}^-_{k+1}$
end procedure

The modified dense model of the system is

$$\begin{aligned} x_{\mu,k+1} &= A_\mu x_{\mu,k} + B_\mu U_k + w_{\mu,k} \\ y_{\mu,k} &= \frac{1}{N} h(X_k, U_k) + v_{\mu,k}, \end{aligned}$$

where $x_\mu \in \mathbb{R}^2$, $\Gamma \in \mathbb{R}^{2N \times 2}$, $A_\mu \in \mathbb{R}^{2 \times 2} = \Gamma^\dagger A \Gamma$, $B_\mu \in \mathbb{R}^{2 \times N} = \Gamma^\dagger B$, $w_{\mu,k} \sim \mathcal{N}(0, \Gamma^\dagger Q \Gamma)$, $v_{\mu,k} \sim \mathcal{N}(0, \frac{1}{N^2} R_k)$. Γ is what is referred to as the "RFF matrix", and is formally defined as

$$\Gamma_k = \frac{\partial X_k}{\partial x_{\mu,k}} = \begin{bmatrix} \gamma^1_s & 0 & \gamma^2_s & 0 & \dots & \gamma^N_s & 0 \\ 0 & \gamma^1_V & 0 & \gamma^2_V & \dots & 0 & \gamma^N_V \end{bmatrix}^T, \tag{6}$$

where γ^i_s is the RFF associated with the i^{th} cell's SOC, and γ^i_V the RFF associated with the i^{th} cell's relaxation voltage. For the predicted covariance, the DEKF draws equivalence of a dense covariance $P_\mu \in \mathbb{R}^{2 \times 2}$ to the sparse covariance in the following manner

$$\begin{aligned} &\Gamma_k P^-_{\mu,k+1} \Gamma^T_k = \\ &\quad = \mathbb{E}\left[\Gamma_k (x_{\mu,k+1} - \hat{x}^-_{\mu,k+1})(x_{\mu,k+1} - \hat{x}^-_{\mu,k+1})^T \Gamma^T_k \right] \\ &\quad = \mathbb{E}\left[(X_{k+1} - \hat{X}^-_{k+1})(X_{k+1} - \hat{X}^-_{k+1})^T \right] \\ &\quad = P^-_{k+1}. \end{aligned}$$

This equivalence to the sparse EKF means we can estimate over the dense state with mean and covariance x_μ and P_μ, which are a fixed size no matter how large N is. The only variable in the dense formulation that grows in proportion to N is Γ, which contains $2N$ unique elements. The algorithm is expressed in its entirety in Algorithms 1 and 2. The fixed size of the estimated distribution along with the linear growth of Γ allows for implementation on an embedded system with significantly lower available memory.

Algorithm 2 DEKF Algorithm: Measurement Update

1: **procedure** MEAS($\hat{x}^-_{\mu,k+1}$, $P^-_{\mu,k+1}$, $\hat{X}^-_{k+1}$, $y_{\mu,k+1}$, k)
2: $H_\mu \leftarrow$ evaluates measurement Jacobian at $\hat{X}^-_{k+1}$;
3: $K_{\mu,k+1} \leftarrow$ computing dense Kalman gain;
4: $\hat{x}^+_{\mu,k+1} \leftarrow \hat{x}^-_{\mu,k+1} + K_{\mu,k+1}(y_{\mu,k+1} - H_\mu(\hat{x}^-_{k+1} - \hat{x}^+_k))$;
5: $\hat{X}^+_{k+1} \leftarrow \hat{X}^-_{k+1} + \Gamma_k(\hat{x}^+_{\mu,k+1} - \hat{x}^-_{\mu,k+1})$;
6: $P^+_{\mu,k+1} \leftarrow (I - K_{\mu,k+1}H_\mu)P^-_{\mu,k+1}$;
7: $k \leftarrow k+1$
8: **return** $\hat{x}^+_{\mu,k+1}, P^+_{\mu,k+1}, \hat{X}^+_{k+1}$
9: **end procedure**

III. Design Space Exploration

Pursuing an optimal partition, we divide the DEKF into three constituent sub-tasks (time update (TU), Kalman gain (KG), and measurement update (MU)), establish programmable software (PS) on the ARM core and programmable logic (PL) on the FPGA as our binding elements, and aim to optimize over energy and execution time. For clarity, TU is the entirety of Algorithm 1, KG is lines 2 and 3 in Algorithm 2, and MU is the rest of Algorithm 2. The problem can be cast as the following nonlinear binary integer program

$$x^* = \arg\min_x \ (c(x))^T x + d^T \mathbb{1}(x) \quad \text{s.t. } x \in \Omega,$$

$$f : \{0,1\}^n \to \mathbb{R}, \ \ \Omega \subseteq \{0,1\}^n,$$
$$\Omega = \{x \in \{0,1\}^n \mid A_{eq}x - b_{eq} = 0\},$$

The optimization variable is a binary vector, each index denoting whether or not a particular task is bound to a certain element. For example, consider the case of α processing elements and β tasks. Then, $\mathbf{card}(x) = \alpha * \beta$, and the binding of task q to processing element r is indicated by index number $\alpha*(q-1)+r$ of x, where $q \in [1,\beta]$, $r \in [1,\alpha]$. The following equality constraint ensures that each task is only bound once:

$$A_{eq}(i,j) = \left\{ \begin{array}{ll} 1, & \text{if } i = \lfloor \frac{j-1}{\alpha} \rfloor + 1 \\ 0, & \text{otherwise.} \end{array} \right\} \quad b_{eq} = \mathbf{diag}(I_\beta),$$

where $A_{eq} \in \{0,1\}^{\beta \times \alpha\beta}$ and $b_{eq} \in \{0,1\}^\beta$.

The objective function consists of two parts. The first is a weighted nonlinear combination of objectives pertaining to execution time $(c_T(x))$ and energy $(c_J(x))$:

$$(c(x))^T x = (w_T c_T(x) + w_J c_J(x))^T x. \tag{8}$$

Through kernel density estimation (KDE), an approximate probability density function is obtained for each task's execution time in PS (considering pipeline flushing, cache hit ratio, etc.). We compute how likely a task j on PS is to execute faster than on the equivalent PL implementation and vice versa:

$$A^j_{PL} = P(t^j_{PL} \geq t^j_{PS}) = \int_0^{t^j_{PL}} \hat{p}^j_{PS}(x)dx. \tag{9a}$$

$$A^j_{PS} = P(t^j_{PL} < t^j_{PS}) = \int_{t^j_{PL}}^{\infty} \hat{p}^j_{PS}(x)dx. \tag{9b}$$

Termed "performance risks", these provide the objective values which are stored in c_T from (8). To estimate energy, we model each implementation as a pulse-wave comprised of active (dynamic power) and latent (static power) periods, computing $P_{avg} = \lambda(P_{st} + P_d) + (1-\lambda)P_{st}$, and multiply by execution time to get energy. Here, $\lambda \in [0,1]$ is the portion of time the system is in an active state. Nonlinear dependence on x arises from edge cases requiring that the cache is flushed, which shifts some of the probability density functions associated thereof, ultimately contributing to execution time.

The second term is nonlinear, accounting for every binding's communication costs $d \in \mathbb{Z}^{\alpha^\beta}$, and computing the following inner product:

$$d^T \mathbb{1}(x) = \sum_{i=1}^{\alpha^\beta} d(i)\mathbb{1}_{x_i}(x), \tag{10}$$

$$\mathbb{1}(x) = \begin{bmatrix} \mathbb{1}_{x_1}(x) & \mathbb{1}_{x_2}(x) & ... \mathbb{1}_{x_{\alpha^\beta}}(x) \end{bmatrix}^T, \tag{11}$$

$$\mathbb{1}_{x_i}(x) : \{0,1\}^{\alpha\beta} \to \{0,1\} = \left\{ \begin{array}{ll} 1, & \text{if } x = x_i \\ 0, & \text{otherwise.} \end{array} \right\}.$$

In essence, (10) functions as a look-up table, where the input is the binding, and the output is the communication overhead.

Our objective function is non-differentiable and discontinuous with respect to x, meaning that traditional (simplex method) and gradient-based (interior-point method) techniques are not applicable in finding the optimal binding. For this case, however, $\alpha = 2$ and $\beta = 3$, so there is no need to employ heuristic methods – the objective function can be evaluated over all 8 bindings, and from there, x^* can be discovered by inspection.

A. DEKF Optimal Binding

With a combination of preliminary implementation details obtained from HLS, wall-clock timing, and data sheets for the target part, we gain a set of initial estimates for the power consumption (Tables I and II) as well as distributions of execution times for each task on PS

TABLE I
PL: Expected Power Consumption (mW)

	TU	KG	MU
Static	107	107	104
Total	349	373	205

TABLE II
PL: Expected Execution Time (μs)

	TU	KG	MU
Time	39.18	0.524	0.183

Using the previously discussed techniques, we can discover by inspection that executing all tasks using PL is the optimal binding over the objectives of time and energy. For comparison, we also implement the second-most Pareto-dominant

solution, which is binding TU to PS, and the rest of the tasks to PL. The estimated execution time and consumed energy for both bindings is labeled in the plot. The following sections explore the implementation of the full PL and hybrid bindings, comparing them across all objectives, and comparing them back to the model procured from DSE.

IV. Implementation

A. Sensor Emulation

To generate the battery data, we use the MATLAB model from [3] and simulate 16 cells over 600 0.1 s time steps. We extract the updated states X_k^+, cell currents U_k, and average terminal voltage measurements y_μ. The embedded implementations consult the latter two to make their predictions which are compared to X_k^+. Relaxation resistances and capacitances are kept constant across all cells, but capacities and efficiencies are normally randomized to induce heterogeneity onto the pack. The data is generated with process noise $W_k \sim \mathcal{N}(0, 10^{-10} \cdot I)$ and measurement noise $v_{\mu,k} \sim \mathcal{N}(0, \frac{1}{16^2} \cdot 10^{-5})$. To access the data in our embedded system, we store the sensor data as .csv files in a micro-SD card. For validation, we also store the resulting predicted SOC on the SD card.

B. High-level Synthesis

Per our DSE, we designed two components using HLS - one containing the KG and MU algorithms, and one containing the entire DEKF. For simplicity, we will refer to these as our hybrid implementation and our PL implementation, respectively. To maximize precision, our HLS components are generated with full floating point accuracy on 64-bit doubles. Table III presents the performance of our two components in size and speed. As expected, the PL-only component is bigger

TABLE III
HLS component size and speed

	Size (LUTs)	Execution Time (μs)
Hybrid	11,304	7.090
PL	19,691	41.400

and takes longer to execute. This is due to the inclusion of the TU algorithm which is our most computationally expensive algorithm. Note that these results are with no performance-related directives. Our exploration of directives yielded only results with lower quality. Additional hardware must also be included to facilitate memory transfer between PS and PL such as block-ram controllers and CDMA's. Naturally, this additional hardware scales with the number of variables that must be shared across PS and PL.

V. Results

Our final results for speed, size, and power utilization can be found in tables IV, V, and VI respectively. As expected, our PL implementation yields the best results with the lowest execution time and lowest power consumption. Interestingly, due to fewer inputs and outputs, it uses less total hardware than the Hybrid implementation when considering the additional hardware required to interact with the PS. In the next subsections we will more closely examine our performance relative to the DSE and our outputs relative to the expected outputs from the MATLAB model.

TABLE IV
Final timing results (μs)

	TU	KG	MU	CDMA Write	PL	CDMA Read	Total
Hybrid	72	–	–	25	8	8	113
PS	68	24	8	–	–	–	100
PL	–	–	–	6	43	3	52

TABLE V
Final Size Results

Size (LUTs)	Digital Logic	Memory	Full BD	% Util.
Hybrid	13,218	18,333	32,448	61
PL	18,504	12,867	32,021	60

A. Comparison to DSE

For our final timing results, the largest error by DSE on this front is an ∼89.3% error in hybrid execution time between what the DSE predicted and what was observed. DSE underestimated the execution time of the hybrid implementation due to initially poor estimates for the delay t_{mem} caused by memory accesses after flushing the cache. Furthermore, our DSE assumed t_{mem} to be a static value, which in reality is a stochastic value [13].

The greatest oversight by the DSE is the dynamic power estimation for the full PL implementation. Table I predicts the expected power consumption of the full PL implementation to be 927 mW, while Table VI tells us that it's actually closer to 1964 mW, yielding a ∼111.9% error. Table VI suggests that the missing link was the power consumed by the Zynq Processing System, which is still required to facilitate the hardware execution interface with the SD card. The DSE did not factor this in; the removal of the Zynq Processing System from the resulting power rating brings down the estimated error to ∼24.7%. The remaining error could be a result of shared resources (data dependencies and potentially re-usable hardware elements could overlap) or the lack of a sophisticated power model to approximate the combined power consumption of each task in PL (additional factors should be taken into account, like resource utilization, switching activity, fanout, and so on [14]).

TABLE VI
Final Power Results

Power (W)	Static	Dynamic	Zynq PS	Total	Total (PL Only)
Hybrid	0.155	1.891	1.272	2.046	0.774
PL	0.152	1.812	1.266	1.964	0.698

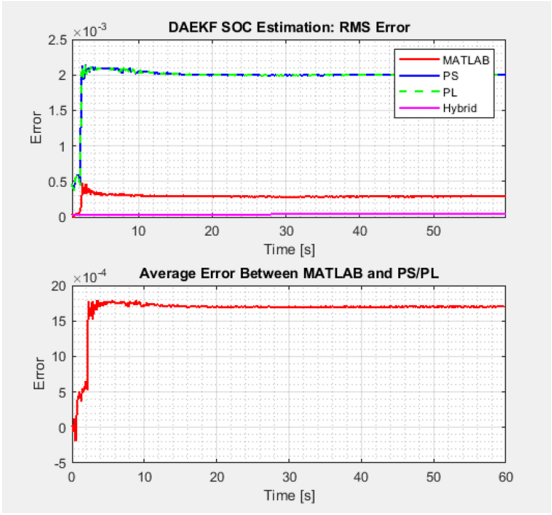

Fig. 1. Estimation performance and average numerical error.

B. Estimation Performance

The pack characteristics see the average terminal voltage (y_μ) monotonically decrease at a value of ~ 3.3V. As for the net currents, the 16 current trajectories are distributed symmetrically around and converging to 4.6A. This behavior simulates the effects of balancing currents, where $\sum_{i=1}^{N} \beta^i = 0$. For the estimation performance, we plot four trajectories (MATLAB, full PL, full PS, and hybrid) and observe the numerical error incurred by reduced precision in the embedded system. Fig. 1 shows MATLAB as exhibiting estimation error on the order of 10^{-4}, whereas the pure implementations garner an error of $\sim 2 \cdot 10^{-4}$ and an average error delta (second subplot) of approximately $1.7 \cdot 10^{-3}$. It should be noted that the PS and PL implementations overlay one another perfectly, with a computed absolute error of exactly 0.

The hybrid system's error trajectory in magenta visibly outperforms the other three implementations. One interpretation of this result is theoretical, which would suggest a minimal error trajectory that drifts over time is the result of over-reliance on the model predictions. In other words, Q_μ may either be 0, or R_μ may be excessively large. Another interpretation is the embedded side, where a hybrid implementation may be freeing up resources which are now utilized for extended-precision computation. However, the latter would not explain the apparent dissimilarity in the error trajectories between the hybrid and the others.

VI. Conclusions

In embedded implementations, speed, power efficiency, and accuracy are crucial for deployment. Using the novel DEKF and hardware acceleration, we successfully created several systems achieving run times as fast as 52 μs with RMS averaging around 0.2%. Compared to the run time in the order of milliseconds presented in [6] and the RMS near 0.7% in [5], this result shows great potential for the DEKF in the context of Kalman filter based SOC estimation.

In the future, this research can be improved in several ways. First, our model for DSE should be updated to match the implementation more closely. Additionally, more objectives can be added to the DSE including size, development time, and cost. Relating to cost, additional configurations could be considered such as cheaper FPGA's and cheaper MCU's. Our algorithms could be modified to use less precision and evaluated to see the trade-off between final accuracy and speed/size. Finally, further HLS directive exploration may yield more optimal results.

References

[1] X. Chen, W. Shen, T. Vo, Z. Cao, and A. Kapoor, "An overview of lithium-ion batteries for electric vehicles," *IEEE*, pp. 230–235, 2012.

[2] H. Askari, A. Khajepour, M. B. Khamesee, and Z. L. Wang, "Embedded self-powered sensing systems for smart vehicles and intelligent transportation," *Nano Energy*, vol. 66, p. 104103, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2211285519308109

[3] L. Nuculaj and J. Chen, "Simultaneous cell state estimation via dense adaptive extended kalman filter," *[Manuscript submitted for publication]*, 2024.

[4] S. Hong, M. Kang, H. Park, J. Kim, and J. Baek, "Real-time state-of-charge estimation using an embedded board for li-ion batteries," *Electronics*, vol. 11, no. 13, p. 2010, Jun 2022.

[5] X. Cui, Z. He, E. Li, A. Cheng, M. Luo, and Y. Guo, "State-of-charge estimation of power lithium-ion batteries based on an embedded micro control unit using a square root cubature kalman filter at various ambient temperatures," *International Journal of Energy Research*, vol. 43, no. 8, p. 3561–3577, Apr 2019.

[6] F. Oehler, K. Nürnberger, J. Sturm, and A. Jossen, "Embedded real-time state observer implementation for lithium-ion cells using an electrochemical model and extended kalman filter," *Journal of Power Sources*, vol. 525, p. 231018, Mar 2022.

[7] Z. Pei, X. Zhao, H. Yuan, Z. Peng, and L. Wu, "An equivalent circuit model for lithium battery of electric vehicle considering self-healing characteristic," *Journal of Control Science and Engineering*, vol. 2018, 2018.

[8] S. S. Madani, E. Schaltz, and S. Knudsen Kær, "An electrical equivalent circuit model of a lithium titanate oxide battery," *Batteries*, vol. 5, no. 1, p. 31, 2019.

[9] H. He, R. Xiong, X. Zhang, F. Sun, and J. Fan, "State-of-charge estimation of the lithium-ion battery using an adaptive extended kalman filter based on an improved thevenin model," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 4, pp. 1461–1469, 2011.

[10] H. He, X. Zhang, R. Xiong, Y. Xu, and H. Guo, "Online model-based estimation of state-of-charge and open-circuit voltage of lithium-ion batteries in electric vehicles," *Energy*, vol. 39, no. 1, pp. 310–318, 2012, sustainable Energy and Environmental Protection 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S036054421200014X

[11] Y. Yokoyama, "Design optimization for low-power and highly reliable embedded srams on advanced cmos platforms."

[12] M. Vidlak, L. Gorel, M. Furmanik, and P. Makys, "Analysis and comparison of the advanced pmsm model-based motor control strategies," in *2022 ELEKTRO (ELEKTRO)*, 2022, pp. 1–9.

[13] A. J. Smith, "Cache memories," *ACM Computing Surveys (CSUR)*, vol. 14, no. 3, pp. 473–530, 1982.

[14] V. Degalahal and T. Tuan, "Methodology for high level estimation of fpga power consumption," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, 2005, pp. 657–660.