International Journal on Advanced Science Engineering Information Technology

Vol.14 (2024) No. 2
ISSN: 2088-5334

FPGA Implementation of a Pipelined Kalman Filter for Object Tracking in Two Dimensions

Jonghuy Choi [a], Yonghyeok Yoon [a], Taekyeong Song [a], Sunhee Kim [a,*]

[a] *Department of System Semiconductor Engineering, Sangmyung University, 31, Sangmyeongdae-gil, Dongnam-gu, Cheonan-si, Chungcheongnam-do, 31066, Republic of Korea*

Corresponding author: [*]happyshkim@smu.ac.kr

***Abstract*—In a location tracking system for a moving object, accuracy and real-time processing are essential factors. The Kalman filter, as a recursive function, stands out as one of the prominent algorithms for object tracking. It continuously compares measured data with predicted values based on the system characteristics in real-time. Then it corrects the error of the predicted values while considering the noise of both system and measured data. This paper focuses on designing a hardware based Kalman filter for object tracking in two dimensions. Following an analysis of the Kalman filter algorithm, the blocks capable of parallel processing are identified and configured to be processed in parallel, effectively reducing data processing time. The clock speed is enhanced by using the pipeline technique. In addition, the time-sharing technique is applied to increase the utilization of hardware resources and reduce the area. Data was processed at 32-bit floating points to uphold accuracy comparable to software-implemented Kalman filters. The proposed Kalman filter architecture is designed using Verilog HDL and then simulated in Synopsys VCS/Verdi. The accuracy is verified by comparing results with a software based Kalman filter designed using MATLAB. It was implemented using Zynq ZYNQ-7 ZC702 Programmable logic via Xilnix Vivado and can operate at 33MHz. It takes 44 clocks, or 1.32 us, to process one data. Therefore, it was confirmed that the designed Kalman filter hardware is suitable for real-time processing.**

***Keywords*—Kalman filter; object tracking; FPGA; parallel processing; pipeline.**

Manuscript received 11 Oct. 2023; revised 12 Jan. 2024; accepted 14 Feb. 2024. Date of publication 30 Apr. 2024.

I. INTRODUCTION

Research on moving object tracking has been actively conducted in various engineering fields, such as autonomous vehicles, robot control, communication systems, and navigation [1]–[4]. Measurement data is crucial for estimating an object's location, but it often contains significant noise both within and outside the system. The noise diminishes the accuracy of prediction and estimation. Consequently, several algorithms are under investigation to address this challenge, and the Kalman filter is a representative algorithm [5]–[7].

The Kalman filter is a recursive algorithm [8]–[10]. It distinguishes itself from typical filters by dynamically updating its filter coefficients, Kalman gain, over time [11]. It compares noisy measured data with predicted values based on system characteristics. Then, it corrects the predicted value using the prediction error and Kalman gain. It provides estimated values with noise effectively removed, responds in real-time to system state changes, and performs highly efficient calculations [8]–[11].

The Kalman filter is a model for linear systems. Since many systems are nonlinear systems, their application is limited. So, the extended Kalman filter (EKF) has been studied to model nonlinear systems [12]–[14]. Unlike the traditional Kalman filter, the EKF employs a nonlinear mathematical model for the system. It dynamically calculates and applies a linear model based on the previous estimate rather than a predetermined linearization model. Although the EKF is more complex than linear systems, it maintains the fundamental algorithmic principles of the Kalman filter [15], [16].

For more accurate tracking, advanced EKF algorithms integrating various sensors have been studied [17], [18]. Object tracking using artificial intelligence has recently been conducted [19]–[21]. They offer enhanced precision, but their complexity necessitates hardware implementation for real-time processing [22]–[24]. FPGA hardware implementation is facilitated within SoC programmable devices like Xilinx Zynq [25], [26]. This approach reduces communication overhead with software and enhances the efficiency of resource allocation between software and hardware [27]–[29].

In this paper, we design a Kalman filter hardware that estimates an object's position in two-dimensional space. In FPGA design, we focused on increasing overall resource efficiency through the utilization of pipeline and parallel processing techniques. In the next Section, we introduce the Kalman filter algorithm, and the system model and parameters used. Then, the proposed hardware structure is explained in detail. Section III presents the results of hardware implementation. Finally, conclusions are drawn in Section IV.

II. Materials and Method

A. Kalman Filter Algorithm

The Kalman filter algorithm comprises three main processes, as shown in Fig. 1: a prediction process, a Kalman filter gain calculation process, and an estimation (or update) process [30], [31].

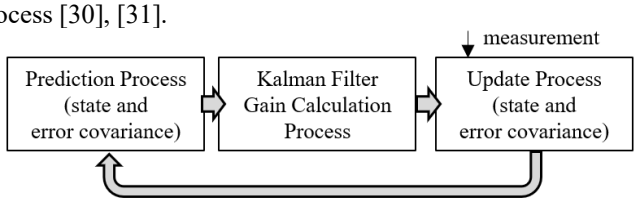

Fig. 1 Kalman filter algorithm

The prediction process involves predicting the state x of a system and its error covariance P. These are expressed through the following equations:

$$\hat{x}_k^- = A\hat{x}_{k-1} \quad (1)$$

$$P_k^- = AP_{k-1}A^T + Q \quad (2)$$

where the superscripts "-" and "^" mean the predicted value and the estimated value, respectively. The subscript "k" indicates the kth step. If k represents the current state, (k-1) signifies the previous state, and (k+1) signifies the next state. Prediction of the estimate of the current state x is derived from the estimate of the previous state and the system model A. System model A describes how a system changes over time. Therefore, A is a predetermined constant prior to the Kalman filter computation, and it is important to reflect the characteristics of the actual system mathematically accurately.

The process of predicting the error covariance P requires to consider system process noise Q, in addition to the previous error covariance and system model A. Similar to the system model A, system process noise Q is predetermined by the system prior to Kalman filter calculation. As noise arises from various factors, it must be treated as a design variable of the Kalman filter, and an appropriate value must be determined through a trial-and-error process. Noise is typically expressed in statistical terms. The Kalman filter assumes that noise adheres to a standard normal distribution with a mean of 0. Thus, only the variance of the noise needs to be ascertained. After the prediction is completed, the Kalman gain K is calculated as follows

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \quad (3)$$

Then, the system state x and its error covariance P are updated using the measurements z through the following equations.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \quad (4)$$

$$P_k = P_k^- - K_k H P_k^- \quad (5)$$

A variable H, one of the system models, signifies the relationship between the measurements z and the system state x. In equation (4), $H\hat{x}_k^-$ on the right side is the predicted value of the measurements computed from the expected state. Consequently, $z_k - H\hat{x}_k^-$ represents the difference between the actual measurements and the predicted measurements. So, it becomes the prediction error. The estimated value for the system state $\hat{x}_k$ is derived by correcting the state prediction value with the prediction error, achieved through the multiplication with the Kalman gain. Similarly, error covariance is also corrected using the Kalman gain.

The estimated value's accuracy depends on the Kalman gain's effectiveness in rectifying the predicted value. When calculating the Kalman gain, P, H, and R are used as in Equation (3). R signifies the uncertainty of the measurement values. Like system process noise Q, it is assumed to follow a normal distribution with a mean of 0. Therefore, only the measurement noise variance requires identification, and its value must be determined through iterative experimentation.

If the measurement noise R is large, the Kalman gain decreases. Consequently, in the correction of the system state outlined in Equation (4), the influence of the measurements decreases, resulting in a more gradual adjustment of the state estimate. Conversely, as the R decreases, the Kalman gain increases, increasing the impact of the measurements.

If the system noise Q is large, the error covariance P becomes large according to Equation (2). As the error covariance P increases, the term in the numerator of the Kalman gain becomes relatively larger compared to the denominator, so the Kalman gain increases. Consequently, during the correction of the system state, the influence of the measurements increases. If the error covariance P is small, the Kalman gain decreases, and the influence of the measurements decreases during the correction of the system state.

B. Mathematical Model for Object Tracking

To track an object moving in a two-dimensional plane, both the position and velocity along the horizontal and vertical axes are considered as the system state. Consequently, the system state x becomes a 4×1 matrix, as shown in the following equation

$$x = [p_x \; v_x \; p_y \; v_y]^T \quad (6)$$

where the subscripts "x" and "y" represent the horizontal and vertical axes, respectively, and p and v indicate position and velocity, respectively. The relationship between position and velocity is established as follows,

$$p_k = p_{k-1} + v_{k-1} \times \Delta t \quad (7)$$

Assuming that the speed is constant, the system model A is set as follows

$$A = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8)$$

where Δt represents the time interval between position measurements. In this paper, a time interval Δt is set at 1.

Since the measurements are the positions along the horizontal and vertical axes, the matrix H, which represents the relationship between the measured values and the state variables, is configured as a 2×4 matrix as follows.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (9)$$

Since both the system noise Q and the measurement noise R are both assumed to follow a normal distribution with a mean of 0, they are represented as diagonal matrices of 4×4 and 2×2, respectively, as follows.

$$Q = \begin{bmatrix} \sigma_{Q11}^2 & 0 & 0 & 0 \\ 0 & \sigma_{Q22}^2 & 0 & 0 \\ 0 & 0 & \sigma_{Q33}^2 & 0 \\ 0 & 0 & 0 & \sigma_{Q44}^2 \end{bmatrix} \quad (10)$$

$$R = \begin{bmatrix} \sigma_{R11}^2 & 0 \\ 0 & \sigma_{R22}^2 \end{bmatrix} \quad (11)$$

Mathworks' MATLAB is used to verify the algorithm. As verification data, coordinate values of a person's face in a 2D image are used. System models A, H, Q, and R must be predetermined before the Kalman filter calculation. On the contrary, the state x and error covariance P are recursively updated in real-time, so the initial values must be specified. In this paper, the initial values of x and P were set as follows

$$x = [0\ 0\ 0\ 0]^T \quad (12)$$

$$P = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix} \quad (13)$$

Based on repeated simulations, the covariances of Q and R are determined to be 1 and 50, respectively. Consequently, the following matrices are formulated

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (14)$$

$$R = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix} \quad (15)$$

By scrutinizing the simulation values for each step, the crucial decision regarding bit precision is made for hardware implementation. Due to the broad range of values involved, performing calculations with a 32-bit floating-point format rather than a 32-bit fixed-point format could enhance accuracy despite the increased hardware complexity.

C. Algorithm Analysis for Hardware Structure

In this paper, a hardware structure of the Kalman filter is devised to facilitate parallel processing, minimize latency, and optimize area by reusing hardware components. In the prediction process, a matrix multiplier is employed to compute the product of the 4×4 matrix A and the 4×1 matrix x to predict the state x. Additionally, for covariance prediction, a matrix multiplier and a matrix adder are utilized to process a 4×4 matrix and a 4×1 matrix. To predict covariance, matrix multiplication needs to be performed twice, which is executed sequentially. Specifically, using a single matrix multiplier, AP_{k-1} is calculated in the initial step, followed by the calculation of $AP_{k-1}A^T$ in the subsequent step.

Since addition typically requires less processing time than multiplication, multiplication and addition operations are combined within a single step. Consequently, a total of two steps are necessary to predict covariance. When calculating the two predicted values, x and P, there is no dependence between them. Consequently, the prediction for x and P can be executed in parallel. As a result, a total of two matrix multipliers and one matrix adder are required for the prediction process, which takes two steps.

Fig. 2 shows the essential hardware components and the steps during which each component is utilized. The matrix_mul_1 and matrix_mul_2 are matrix multipliers for a 4×4 matrix multiplied by a 4×1 matrix and for a 4×4 matrix multiplied by a 4×4 matrix, respectively. The matrix_add_1 is a matrix adder for two 4×4 matrices. In the prediction process, the matrix_mul_1 is used during the first step to compute $A\hat{x}_{k-1}$, and the matrix_mul_2 is used during two steps to compute $AP_{k-1}A^T$. The matrix_add_1 operates during the second step to compute P_k. This parallel processing reduces latency in the Kalman filter prediction process.

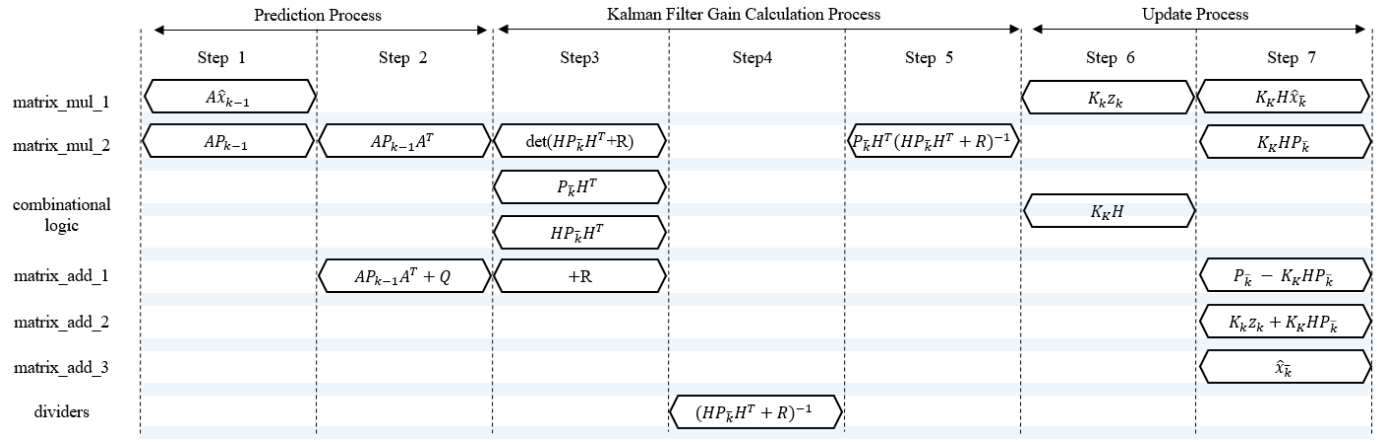

Fig. 2 Essential hardware components and their time schedule

To calculate $P_k^- H^T$ included in the Kalman gain, a matrix multiplier for a 4×4 matrix multiplied by a 4×2 matrix. However, since H is a constant matrix and only two of its eight elements are the non-zero value, $P_k^- H^T$ can be determined solely through positional transformation without any arithmetic operation as follows.

$$P_k^- H^T = \begin{bmatrix} P_{k,11}^- & P_{k,12}^- & P_{k,13}^- & P_{k,14}^- \\ P_{k,21}^- & P_{k,22}^- & P_{k,23}^- & P_{k,24}^- \\ P_{k,31}^- & P_{k,32}^- & P_{k,33}^- & P_{k,34}^- \\ P_{k,41}^- & P_{k,42}^- & P_{k,43}^- & P_{k,44}^- \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} P_{k,11}^- & P_{k,13}^- \\ P_{k,21}^- & P_{k,23}^- \\ P_{k,31}^- & P_{k,33}^- \\ P_{k,41}^- & P_{k,43}^- \end{bmatrix} \quad (16)$$

Similarly, the calculation of $HP_k^- H^T$ can also be achieved through a positional transformation as the following equation (17). These two positional transformations result in a 2x2 matrix.

$$HP_k^- H^T = \begin{bmatrix} P_{k,11}^- & P_{k,13}^- \\ P_{k,31}^- & P_{k,33}^- \end{bmatrix} \quad (17)$$

Positional transformation does not require separate hardware logic. When connecting the calculation result to the input of other logic, the position of elements can be converted and connected seamlessly. A 2x2 matrix adder is required to add a 2x2 matrix R to the calculation result. Since a 2x2 matrix adder is a subset of a 4x4 matrix adder, the matrix_add_1 is used.

To find the inverse of a 2×2 matrix, the determinant |M| must be calculated. Since the 2×2 matrix ($HP_k^- H^T + R$) is a diagonal matrix according to equations (13), (15), and (17), its determinant calculation requires one multiplier. Since a matrix multiplier includes several multipliers, one multiplier included in the matrix_mul_2 calculates the determinant. Then, each element of the matrix is divided by the determinant. As the matrix is diagonal, two dividers are required. The calculation of the Kalman gain is completed by performing matrix multiplication using the matrx_mul_2 for $P_k^- H^T$ and $(HP_k^- H^T + R)^{-1}$.

In the Kalman gain calculation process, two dividers are added. In one step, $P_k^- H^T$, $(HP_k^- H^T + R)$ and the determinants are calculated. Subsequently, division and matrix multiplication are performed in two separate steps. Therefore, a total of three steps are needed. Like in the prediction process, in the estimation process, there is no dependency between state and error covariance, both can be executed in parallel. To estimate covariance, matrix multiplication of the 4×2 matrix Kalman gain and 2×4 matrix H is required. This is accomplished using the matrix_mul_2. Simultaneously, for state update, matrix multiplication of the 4×2 matrix Kalman gain and 2×1 matrix z is calculated using the matrix_mul_1. The reason why $H\hat{x}_k^-$ is not calculated first is that it cannot be calculated with the matrix_mul_1. Unless a matrix multiplier is newly added, the covariance and state cannot be calculated simultaneously. In the second step of the covariance estimation, $K_k HP_k^-$ is calculated using the matrix_mul_2, followed by subtraction from P using the matrix add_1.

To estimate the state, $K_k H$ calculated in the previous step is multiplied by $\hat{x}_k^-$ using the matrix_mul_1, followed by two additional 4×1 matrix addition. In the final step of the estimation process, three 4×1 matrix adder with four elements is needed. As shown in Fig. 2, the proposed Kalman filter needs two matrix multipliers, three matrix adders, and two divisors and requires a total of seven steps.

A. Hardware Structure

The proposed Kalman filter hardware for 2D object tracking consists of two matrix multipliers, three matrix adders, two dividers, some combinational logic and controller shown in Fig. 2. There are measurements z as inputs to the system. The input measurements are converted into an IEEE 754 single precision format, and each operation is processed in a pipeline structure.

1) [4×4] × [4×4] Matrix Multiplier: To multiply a 4×4 matrix A and a 4×4 matrix B, each element in one row of A must be multiplied with each element in one column of B, and the resulting four values must be added to calculate one element of the resulting matrix C. Since C is a 4×4 matrix, this process must be repeated 16 times. In other words, a matrix multiplication of a 4×4 matrix and a 4×4 matrix requires 64 multiplication operations.

The matrix multiplier in the proposed hardware design consists of four multipliers. As shown in Fig. 3, matrix A and matrix B elements are fed into the multiplier, and the resulting element is computed as the sum of the products of corresponding elements of matrix A and matrix B. An element C [0][0] of a resulting matrix C is (A [0][0] ×B [0][0]) + (A[0][1] × B [1] [0]) + (A[0][2] × B[2][0]) + (A[0][3]×B[3][0]). Generally, C[i][j] is (A[i][0] ×B [0] [j]) + (A[i][1] ×B [1] [j]) + (A[i][2] ×B[2][j]) + (A[i][3] ×B[3][j]). Thus, four elements of one row of matrix A are arranged as the same number input port of the four multiplexers, respectively. When sel_mode_A = i, A[i][0], A[i][1], A[i][2], and A[i][3] are selected and used as multiplier inputs, respectively. On the other hand, the four elements of one column of matrix B are arranged as the same number input port of the four multiplexers, respectively. When sel_mode_B = j, B[0][j], B[1][j], B[2][j], and B[3][j] are selected and used as multiplier inputs, respectively. Therefore, eight elements required for C[i][j] are selected by the signals, sel_mode_A and sel_mode_B. In this case, the sel_mode_B changes its value every clock cycle, iterating through 0→1→2→3 four times, to select elements from different columns of B. The sel_mode_A changes its value every four clock cycles, iterating through 0→1→2→3 once, to select elements from different rows of B. This process computes C [0][0], C [0][1], C [0][2], … , C[3][2], C[3][3] in order. A total of 16 clock cycles are required.

This matrix multiplier, initially designed for 4×4 matrix multiplication by 4×4 matrix, is also used 4×2 matrix multiplication by 2×2 matrix. In this configuration, a total of 8 elements needs to be calculated for the resulting 4×2 matrix. So, the inputs of the multiplexer and the signals, sel_mode_A and sel_mode_B, must be changed. In Fig. 3, the inputs of the top two multipliers remain unchanged, while the inputs of the bottom two multipliers are modified.

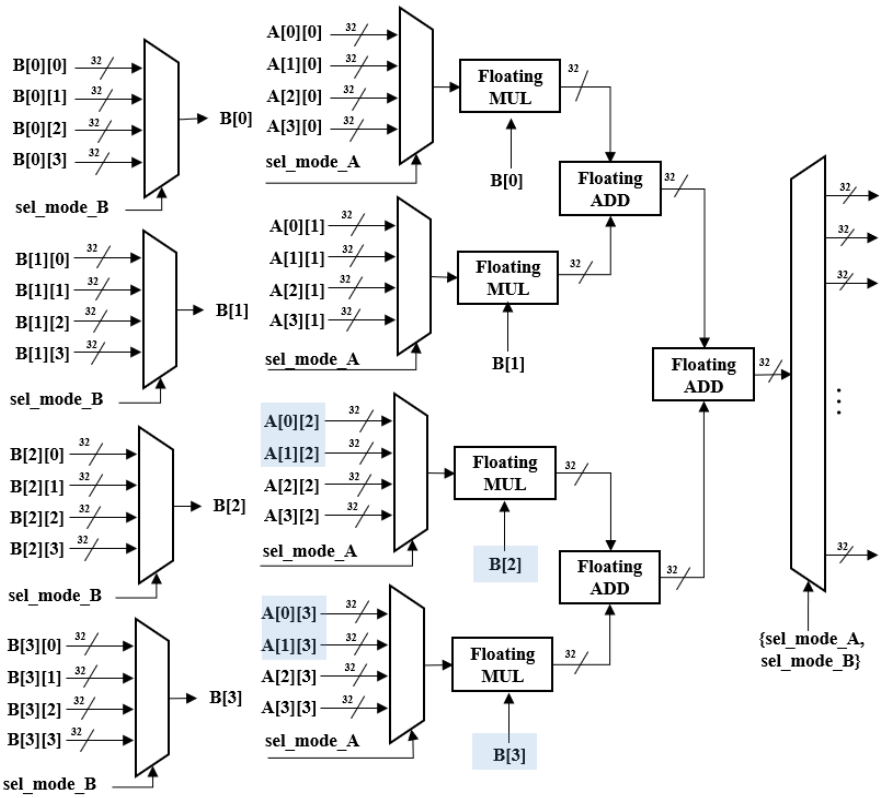

Fig. 3 Block diagram of a 4×4 matrix multiplier by a 4×4 matrix

The inputs of multiplexers that select the elements of matrix A are changed to A[2][0], A[3][0], A[2][1], and A[3][1] instead of A[0][2], A[1][2], A[0][3], and A[1][3], respectively. The multiplexers that select the elements of the matrix B are not used. Instead, the elements of B selected in the upper multiplexers are utilized as multiplier inputs. The sel_mode_B repeats sequentially from 0 to 1 twice every clock cycle, while the sel_mode_A changes from 0 to 1 once every two clock cycles. Therefore, the operation requires a total of 4 clock cycles to compute the resulting 4×2 matrix.

2) [4×4] ×[4×1] Matrix Multiplier: Fig. 4 shows a 4×4 matrix multiplier by a 4×1 matrix. In contrast to the 4×4 matrix multiplier by 4×4 matrix, this design does not require a multiplexer for matrix B. That is, only sel_mode_A changes sequentially from 0 to 3 every clock cycle, indicating the row selection from a matrix A. The calculation is completed in a total of 4 clocks. The calculation is completed in a total of 4 clock cycles, as each clock cycle processes one row of the matrix A. The matrix multiplier, originally designed for 4×4 matrix multiplication by 4×1 matrix, is repurposed for 4×2 matrix multiplication by 2×1 matrix. In Fig. 4, the inputs of the top two multipliers remain unchanged. On the other hand, the inputs of the bottom two multipliers are changed. The inputs of multiplexers that select the elements of matrix A are changed to A_{20}, A_{30}, A_{21} and A_{31} instead of A_{02}, A_{12}, A_{03}, and A_{13}, respectively. The multiplexers that select the elements of the matrix B are not used. Instead, the elements of B selected in the upper multiplexers are utilized as the multiplier inputs. The sel_mode_A changes from 0 to 1 once every clock. Therefore, a total of 2 clocks are required to complete the operation.

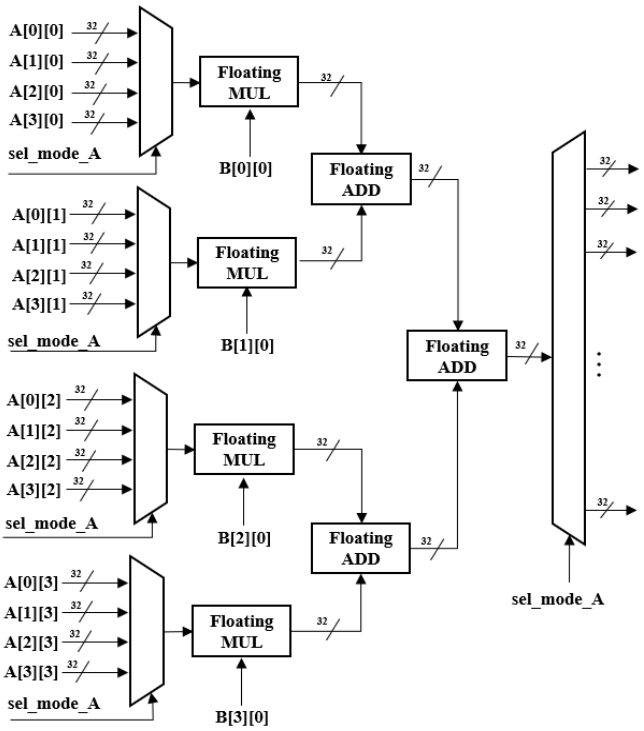

Fig. 4 Block diagram of a 4×4 matrix multiplier by a 4×1 matrix

3) Matrix adders: A matrix adder consists of adders between elements in each matrix position. Since the largest matrix of this Kalman filter is 4x4 matrix, at least 16 adders are required to perform matrix addition within one clock cycle. Three matrix additions are calculated in step 7 of the Kalman filter operation. In detail, two 4x1 matrix adders and

one 4x4 matrix adder are required, resulting in 24 adders. In 2 and 3 steps, the 4x4 matrix adder is utilized among them. Furthermore, matrix adders are also utilized as submodules of the matrix multipliers. A 4x4 matrix multiplier by 4×4 matrix and a 4×4 matrix multiplier by 4×1 matrix contains three 4×4 matrix adders and three 4×1 matrix adders, respectively.

4) Floating point multiplier: The matrix multipliers consist of floating-point multipliers. The floating-point multiplier operates by separately calculating the multiplication of mantissas and the addition of exponents. And then, normalization is done to meet single precision standards.

5) Floating point divider: Floating point dividers are needed to calculate the inverse of a matrix. Similar to the floating-point multiplier, the floating-point divider operates by separately handling the division of mantissas and the addition of exponents. And then, normalization is done to meet single precision standards.

6) Floating point adder: Unlike the floating-point multiplier and divider, the mantissas are shifted according to the difference in exponents, added together, and then normalized.

III. Results and Discussion

The proposed Kalman filter hardware is designed using Verilog HDL and functionally verified using Synopsys' VCS/Verdi. Its simulation results are compared with the software Kalman filter designed in Mathwork's MATLAB. As test data comprise the human face coordinates extracted from the video using OpenCV and are utilized as measurements to both Kalman filters. As simulation results, the estimated human face coordinates from the designed hardware Kalman filter differ from those of the software Kalman filter in the first decimal place. This difference can be attributed to various factors. While the software Kalman filter uses 64-bit double precision, the hardware Kalman filter uses 32-bit single precision. In addition, the designed hardware uses the clipping method among several rounding techniques. Despite these differences, considering that the object's coordinates are expressed in integer units, the error in the first decimal place is deemed sufficiently small. Table 1 compares the software and hardware results for the first cycle.

TABLE I
SIMULATION RESULTS COMPARISON

	SW Kalman filter	HW Kalman filter
K_k	$\begin{bmatrix} 0.8008 & 0 \\ 0.3984 & 0 \\ 0 & 0.8008 \\ 0 & 0.3984 \end{bmatrix}$	$\begin{bmatrix} 0.7979 & 0 \\ 0.3968 & 0 \\ 0 & 0.7979 \\ 0 & 0.3968 \end{bmatrix}$
$\hat{x}_k$	$\begin{bmatrix} 497.29 \\ 247.45 \\ 217.01 \\ 107.96 \end{bmatrix}$	$\begin{bmatrix} 495.35 \\ 246.44 \\ 216.16 \\ 107.54 \end{bmatrix}$
P_k	$\begin{bmatrix} 40 & 19 & 0 & 0 \\ 19 & 61 & 0 & 0 \\ 0 & 0 & 40 & 19 \\ 0 & 0 & 19 & 61 \end{bmatrix}$	$\begin{bmatrix} 42 & 21 & 0 & 0 \\ 21 & 58 & 0 & 0 \\ 0 & 0 & 42 & 21 \\ 0 & 0 & 21 & 58 \end{bmatrix}$

Fig. 5 shows the prediction and estimation value for the coordinates of the human face in the video. It is confirmed that the proposed hardware Kalman filter estimates well the object's location according to its movement in continuous motion.

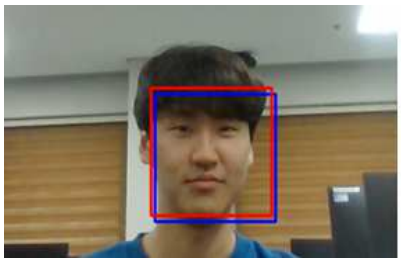

Fig. 5 Kalman filter results: prediction box(blue) and estimation box(red)

Fig. 6 shows the schematic of the top module confirmed by Verdi. After completing functional verification, Xilinx Vivado is used for FPGA implementation using Xilinx ZYNQ-7 ZC702 Evaluation board. Table 2 provides a summary of the number of components, such as slice LUTs table and slice registers, utilized by the design.

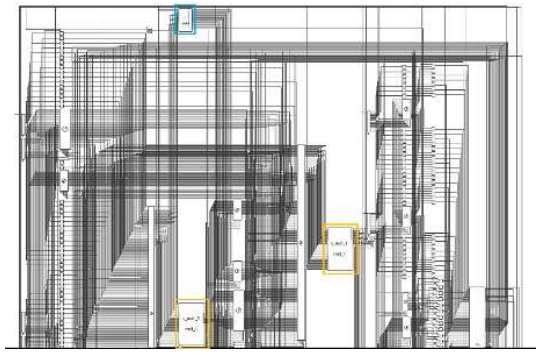

Fig. 6 Schematic of the designed Kalman filter

TABLE II
IMPLEMENTATION RESULTS

Components	Number
Slice LUTs (53200) table	47042
DSPs (220)	64
BUFGCTRL (32)	12
Slice Registers (106400)	12251

TABLE III
DESIGN TIMING SUMMARY

Timing Requirements	Time
Setup Worst Negative Slack	0.230 ns
Worst Hold Slack	0.081 ns
Worst Pulse Width Slack	14.500 ns

The clock period is set to 30 ns. Table 3 shows the design timing summary after the post-fit. These results demonstrate that the designed Kalman filter hardware meets the timing requirements, allowing it to operate at the desired clock frequency of 33.3 MHz. Additionally, it is expected to operate at a maximum clock frequency of 34.2 MHz.

The software Kalman filter operates on an i5-1135G7 CPU running at 2.42 GHz with 16 GB of RAM. It takes approximately 1.2ms to process one image data. In contrast, the designed hardware Kalman filter requires 44 clock cycles with a clock period of 30 ns (=33 MHz). This results in a total

processing time of 1.32 us. It was confirmed that the latency of the proposed hardware Kalman filter is approximately 1/1000th of the latency of the software Kalman filter.

The execution time of the software Kalman filter includes system overhead, which may not be directly comparable to the latency of the hardware Kalman filter. Considering that the input data for the hardware Kalman filter is stored in memory, the overhead due to communication with software is minimal. However, even if the CPU time is around 50% of the total execution time for the software Kalman filter, the latency of the software implementation is still approximately 600 us. Consequently, the hardware Kalman filter demonstrates significantly lower processing time than the software implementation. The designed Kalman filter is implemented using the programmable logic of ZYNQ-7 ZC702. If it is implemented in SoC, including processing systems, it is expected to enhance both efficiency and performance, making the Kalman filter more suitable for real-time applications.

IV. Conclusion

In this paper, the pipelined Kalman filter for object tracking in two dimensions is implemented in hardware. The hardware implementation offers advantages such as parallel processing and increased resource utilization through pipeline techniques and time sharing. In particular, when the same operation is performed using data of different sizes at different times, the hardware is designed based on the largest data size and selectively used to reduce the hardware area and increase efficiency. It is designed with Verilog HDL and compiled using Synopsys' VCS/Verdi. Its functionality is verified by comparing its simulation results with the Kalman filter in Mathwork's MATLAB. It is implemented on an FPGA using ZYNQ-7 ZC702. Approximately 47 k slice LUTs, 64 DSPs, and 12 k slice registers are used. The clock speed is 33MHz, and 44 clock cycles are required to process one data. Therefore, the processing time is 1.32 us. It is confirmed that the proposed hardware Kalman filter is more than 1000 times faster than the software Kalman filter. In the future, we plan to implement it in SoC to enhance its usability further.

References

[1] Y. Fang, A. Panah, J. Masoudi, B. Barzegar and S. Fatehi, "Adaptive Unscented Kalman Filter for Robot Navigation Problem (Adaptive Unscented Kalman Filter Using Incorporating Intuitionistic Fuzzy Logic for Concurrent Localization and Mapping)," *IEEE Access*, vol. 10, pp. 101869-101879, 2022, doi:10.1109/access.2022.3207925.

[2] Y. Li, C. Bian and H. Chen, "Object Tracking in Satellite Videos: Correlation Particle Filter Tracking Method with Motion Estimation by Kalman Filter," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1-12, Sept. 2022, Art no. 5630112, doi:10.1109/TGRS.2022.3204105.

[3] Q. Yu, B. Wang and Y. Su, "Object Detection-Tracking Algorithm for Unmanned Surface Vehicles Based on a Radar-Photoelectric System," *IEEE Access*, vol. 9, pp. 57529-57541, 2021, doi:10.1109/access.2021.3072897.

[4] A. -S. T. Hussain, M. Fadhil, T. A. Taha, O. K. Ahmed, S. A. Ahmed and H. Desa, "GPS and GSM Based Vehicle Tracking System," in *2023 7th International Symposium on Innovative Approaches in Smart Technologies (ISAS),* Istanbul, Turkiye, 2023, pp. 1-5, doi:10.1109/ISAS60782.2023.10391720.

[5] K. Feng, J. Li, D. Zhang, X. Wei and J. Yin, "Robust Cubature Kalman Filter for SINS/GPS Integrated Navigation Systems With Unknown Noise Statistics," *IEEE Access*, vol. 9, pp. 9101-9116, 2021, doi:10.1109/access.2020.3036423.

[6] G. Ciaparrone, F. L. Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, and F. Herrera, "Deep learning in video multi-object tracking: A survey," *Neurocomputing*, vol. 381, 2020, pp. 61-88, doi:10.1016/j.neucom.2019.11.023.

[7] Y. Nie, C. Bian, and L. Li, "Object Tracking in Satellite Videos Based on Siamese Network with Multidimensional Information-Aware and Temporal Motion Compensation," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1-5, 2022, Art no. 6517005, doi:10.1109/LGRS.2022.3211695.

[8] Q. Li, R. Li, K. Ji, and W. Dai, "Kalman Filter and Its Application," in *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, Tianjin, China, 2015, pp. 74-77, doi:10.1109/ICINIS.2015.35.

[9] M. S. Grewal, and A. P. Andrews, "Linear Optimal Filters and Predictors" in *Kalman Filtering: Theory and Practice with MATLAB*, 4th ed., New Jersey, USA: Wiley, 2015, pp. 169-238.

[10] P. Kim, "Kalman Filter," in *Essential Kalman filter,* Korea: A-Jin, 2011.

[11] J. Mochnac, S. Marchevsky and P. Kocan, "Bayesian filtering techniques: Kalman and extended Kalman filter basics," in *2009 19th International Conference Radioelektronika*, Bratislava, Slovakia, 2009, pp. 119-122, doi:10.1109/radioelek.2009.5158765.

[12] Y. Fang, L. Yu, S. Fei, "An improved moving tracking algorithm with multiple information fusion based on 3D sensors," *IEEE Access*, vol. 8, pp. 142295–142302, 2020, doi:10.1109/access.2020.3008435.

[13] P. S. Madhukar, and L. B. Prasad, "State Estimation using Extended Kalman Filter and Unscented Kalman Filter," in *2020 International Conference on Emerging Trends in Communication, Control and Computing (ICONC3)*, Lakshmangarh, India, 2020, pp. 1-4, doi:10.1109/iconc345789.2020.9117536.

[14] S. Yang, and M. Baum, "Extended Kalman filter for extended object tracking," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, LA, USA, 2017, pp. 4386-4390, doi:10.1109/icassp.2017.7952985.

[15] J. Khodaparast, "A Review of Dynamic Phasor Estimation by Non-Linear Kalman Filters," *IEEE Access*, vol. 10, pp. 11090-11109, 2022, doi:10.1109/access.2022.3146732.

[16] S. Feng, X. Li, S. Zhang, Z. Jian, H. Duan, and Z. Wang, "A review: state estimation based on hybrid models of Kalman filter and neural network," *Systems Science & Control Engineering*, vol. 11, no. 1, 2173682, 2023, doi:10.1080/21642583.2023.2173682.

[17] Y. Wang and X. Mu, "Dynamic Siamese Network With Adaptive Kalman Filter for Object Tracking in Complex Scenes," *IEEE Access*, vol. 8, pp. 222918-222930, 2020, doi:10.1109/access.2020.3043878.

[18] T. Kim, and T. H. Park, "Extended Kalman Filter (EKF) Design for Vehicle Position Tracking Using Reliability Function of Radar and Lidar," *Sensors*, vol. 20, no. 15, 2020, doi:10.3390/s20154126.

[19] Y. Liu, L. Zhang, Z. Chen, Y. Yan and H. Wang, "Multi-Stream Siamese and Faster Region-Based Neural Network for Real-Time Object Tracking," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 11, pp. 7279-7292, Nov. 2021, doi:10.1109/TITS.2020.3006927.

[20] H. Wang, W. Ma, S. Zhang, and W. Hao, "Hierarchical Feature Pooling Transformer for Efficient UAV Object Tracking," *IEEE Geoscience and Remote Sensing Letters*, vol. 20, pp. 1-5, 2023, Art no. 6010405, doi:10.1109/LGRS.2023.3314435.

[21] S. Kim, I. Petrunin and H. -S. Shin, "A Review of Kalman Filter with Artificial Intelligence Techniques," in *2022 Integrated Communication, Navigation and Surveillance Conference (ICNS)*, Dulles, VA, USA, 2022, pp. 1-12, doi:10.1109/ICNS54818.2022.9771520.

[22] Y. -S. Zhang, T. -H. Chen, Y. -S. Chiou, S. -L. Chen, W. -T. Chen, Y. -K. Lin, F. -J. Wen, and T. -L. Lin, "Design and Implementation of Real-Time Localization Algorithms Based on FPGA for Positioning and Tracking," in *2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, Yunlin, Taiwan, 2019, pp. 446-448

[23] A. K. Madsen, M. S. Trimboli and D. G. Perera, "An Optimized FPGA-Based Hardware Accelerator for Physics-Based EKF for Battery Cell Management," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, Seville, Spain, 2020, pp. 1-5, doi:10.1109/ISCAS45731.2020.9181073.

[24] W. -T. Chen, S. -L. Chen, Y. -S. Chiou, T. -L. Lin, F. -J. Wen and Y. -K. Lin, "FPGA-Based Implementation of Reduced-Complexity Filtering Algorithm for Real-Time Location Tracking," in *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big*

Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/ CBDCom/CyberSciTech), Fukuoka, Japan, 2019, pp. 721-726, doi:10.1109/DASC/PiCom/CBDCom/CyberSciTech.2019.00136.

[25] J. Soh and X. Wu, "An FPGA-Based Unscented Kalman Filter for System-On-Chip Applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 4, pp. 447-451, April 2017, doi:10.1109/TCSII.2016.2565730.

[26] F. Tian, X. Guo, and W. Fu, "Target Tracking Algorithm Based on Adaptive Strong Tracking Extended Kalman Filter," Electronics, vol. 13, no. 3:652, 2024, doi:10.3390/electronics13030652.

[27] B. Praveenkumar, and P. Eswaran, "FPGA implementation of multi-dimensional Kalman filter for object tracking and motion detection," *Engineering Science and Technology, an International Journal*, vol. 33, 2022, doi:10.1016/j.jestch.2021.101084.

[28] A. Mills, P.H. Jones, and J. Zambreno, "Parameterizable FPGA-based Kalman Filter Coprocessor using Piecewise Affine Modeling," in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Chicago, IL, USA, 2016, pp. 139–147, doi:10.1109/IPDPSW.2016.101.

[29] P. Zhang, W. Li, and X. Yang, "Efficient Implementation of Recursive Multi-Frame Track-Before-Detect Algorithm Based on FPGA," in: *Proc. 2019 International Conference on Control, Automation and Information Sciences (ICCAIS)*, Chengdu, China, 2019, pp. 1–6

[30] M. Pavlović, Z. Banjac and B. Kovačević, "Object Tracking in SWIR Imaging Based on Both Correlation and Robust Kalman Filters," *IEEE Access*, vol. 11, pp. 63834-63851, 2023, doi:10.1109/access.2023.3288694.

[31] C. T. Ginalih, A. S. Jatmiko, and R. Darmakusuma, "Simple Application of Kalman Filter On a Moving Object in Unity3D," in *2020 6th International Conference on Interactive Digital Media (ICIDM)*, Bandung, Indonesia, 2020, pp. 1-3, doi:10.1109/ICIDM51048.2020.9339662.