Hindawi Publishing Corporation
EURASIP Journal on Applied Signal Processing
Volume 2006, Article ID 89186, Pages 1–12
DOI 10.1155/ASP/2006/89186

A New Pipelined Systolic Array-Based Architecture for Matrix Inversion in FPGAs with Kalman Filter Case Study

Abbas Bigdeli, Morteza Biglari-Abhari, Zoran Salcic, and Yat Tin Lai

Department of Electrical and Computer Engineering, the University of Auckland, Private Bag 92019, Auckland, New Zealand

Received 11 November 2004; Revised 20 June 2005; Accepted 12 July 2005

A new pipelined systolic array-based (PSA) architecture for matrix inversion is proposed. The pipelined systolic array (PSA) architecture is suitable for FPGA implementations as it efficiently uses available resources of an FPGA. It is scalable for different matrix size and as such allows employing parameterisation that makes it suitable for customisation for application-specific needs. This new architecture has an advantage of $O(n)$ processing element complexity, compared to the $O(n^2)$ in other systolic array structures, where the size of the input matrix is given by $n \times n$. The use of the PSA architecture for Kalman filter as an implementation example, which requires different structures for different number of states, is illustrated. The resulting precision error is analysed and shown to be negligible.

Copyright © 2006 Hindawi Publishing Corporation. All rights reserved.

1. INTRODUCTION

Many DSP algorithms, such as Kalman filter, involve several iterative matrix operations, the most complicated being matrix inversion, which requires $O(n^3)$ computations (n is the matrix size). This becomes the critical bottleneck of the processing time in such algorithms.

With the properties of inherent parallelism and pipelining, systolic arrays have been used for implementation of recurrent algorithms, such as matrix inversion. The lattice arrangement of the basic processing unit in the systolic array is suitable for executing regular matrix-type computation. Historically, systolic arrays have been widely used in VLSI implementations when inherent parallelism exists in the algorithm [1].

In recent years, FPGAs have been improved considerably in speed, density, and functionality, which makes them ideal for system-on-a-programmable-chip (SOPC) designs for a wide range of applications [2]. In this paper we demonstrate how FPGAs can be used efficiently to implement systolic arrays, as an underlying architecture for matrix inversion and implementation of Kalman filter.

The main contributions of this paper are the following.

(1) A new pipelined systolic array (PSA) architecture suitable for matrix inversion and FPGA implementation, which is scalable and parameterisable so that it can be easily used for new applications
(2) A new efficient approach for hardware-implemented division in FPGA, which is required in matrix inversion.
(3) A Kalman filter implementation, which demonstrates the advantages of the PSA.

The paper is organised as follows. In Section 2, the Schur complement for the matrix inversion operation is described and a generic systolic array structure for its implementation is shown. Then a new design of a modified array structure, called PSA, is proposed. In Section 3, the performance of two approaches for scalar division calculation, a direct division by divider and an approximated division by lookup table (LUT) and multiplier, are compared. An efficient LUT-based scheme with minimum round-off error and resource consumption is proposed. In Section 4, the PSA implementation is described. In Section 5, the system performance and results verification are presented in detail. Benchmark comparison and the design limitations are discussed to show the advantages as well as the limitations of the proposed design. In Section 6, Kalman filter implementation using the proposed PSA structure is presented. Section 7 presents concluding remarks.

2. MATRIX INVERSION

Hardware implementation of matrix inversion has been discussed in many papers [3]. In this section, a systolic-array-based inversion is introduced to target more efficient implementation in FPGAs.

2.1. Schur complement in the Faddeev algorithm

For a compound matrix $\mathbf{M}$ in the Faddeev algorithm [4],

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ -\mathbf{C} & \mathbf{D} \end{bmatrix}, \tag{1}$$

where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are matrices with size of $(n\times n)$, $(n\times l)$, $(m \times n)$, and $(m \times l)$, respectively, the Schur complement, $\mathbf{D}+\mathbf{C}\mathbf{A}^{-1}\mathbf{B}$, can be calculated provided that matrix A is nonsingular [4].

First, a row operation is performed to multiply the top row by another matrix $\mathbf{W}$ and then to add the result to the bottom row:

$$\mathbf{M}' = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ -\mathbf{C}+\mathbf{W}\mathbf{A} & \mathbf{D}+\mathbf{W}\mathbf{B} \end{bmatrix}. \tag{2}$$

When the lower left-hand quadrant of matrix $\mathbf{M}'$ is nullified, the Schur complement appears in the lower right-hand quadrant. Therefore, $\mathbf{W}$ behaves as a decomposition operator and should be equal to

$$\mathbf{W} = \mathbf{C}\mathbf{A}^{-1} \tag{3}$$

such that

$$\mathbf{D}+\mathbf{W}\mathbf{B} = \mathbf{D}+\mathbf{C}\mathbf{A}^{-1}\mathbf{B}. \tag{4}$$

By properly substituting matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$, the matrix operation or a combination of operations can be executed via the Schur complement, for example, as follows.

(i) Multiply and add:

$$\mathbf{D}+\mathbf{C}\mathbf{A}^{-1}\mathbf{B} = \mathbf{D}+\mathbf{C}\mathbf{B} \tag{5}$$

if $\mathbf{A} = \mathbf{I}$;

(ii) Matrix inversion:

$$\mathbf{D}+\mathbf{C}\mathbf{A}^{-1}\mathbf{B} = \mathbf{A}^{-1} \tag{6}$$

if $\mathbf{B} = \mathbf{C} = \mathbf{I}$ and $\mathbf{D} = \mathbf{0}$.

2.2. Systolic array for Schur complement implementation

Schur complement is a process of matrix triangulation and annulment [5]. Systolic arrays, because of their regular lattice structure and the parallelism, are a good platform for the implementation of the Schur complement. Different systolic array structures, which compute the Schur complement, are presented in the literature [3, 6–8]. However, when choosing an array structure one must take into account the design efficiency, structure regularity, modularity, and communication topology [9].

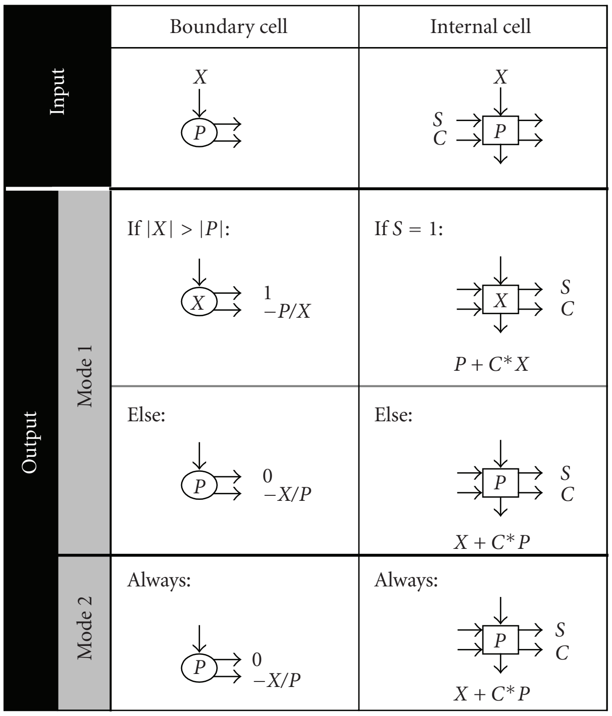


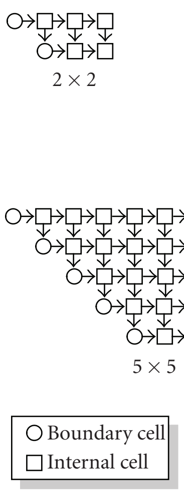
Figure 1: Operations of boundary cell and internal cell.

The array structure presented in [6] is taken as the starting point for our approach. It consists of only two types of cells, the boundary and internal cells. The structure in [3] needs three types of cells. The cell arrangement in the chosen structure is two-dimensional while the cells in [7] are connected in three-dimensional space with much higher complexity.

The other consideration when choosing the target structure was the type of operations in the cells. In the preferred structure [6], all the computations executed in cells are linear, while [8] would require operations such as square and square root calculations.

A cell is a basic processing unit that accepts the input data and computes the outputs according to the specified control signal. Both the boundary and internal cells have two different operating modes that determine the computation algorithms employed inside the cells. *Mode 1* executes matrix triangulation and *mode 2* performs annulment. The operating mode of the cell depends on the comparison result between the input data and the register content in the cell. The cell operations are described in Figure 1.

To create a systolic array for Schur complement evaluation, $\mathbf{E} = \mathbf{D}+\mathbf{C}\mathbf{A}^{-1}\mathbf{B}$, cells are placed in a pattern of an inverse trapezium shown in Figure 2. The systolic array size is controlled by the size of output matrix $\mathbf{E}$, which is a square matrix in case of matrix inversion. The number of cells in the top row is twice the size of $\mathbf{E}$ and the number of internal cells



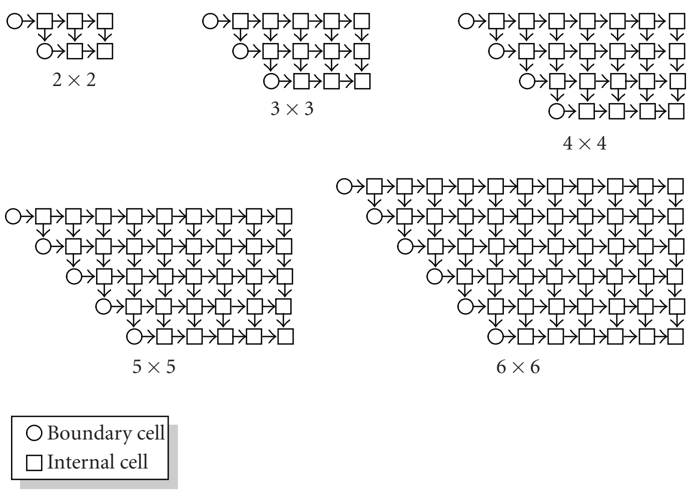


Figure 2: Cells layout in systolic array for different output matrix sizes.

in the bottom row is the same as the size of **E**. The number of boundary cells and layers is equal to the size of matrix **E**.

Inputs are packed in a skewed sequence entering the top of the systolic array. Outputs are produced from the bottom row. Data and control signals are transferred inside the array structure from left to right and top to bottom in each layer through the interconnections. Dataflow is synchronous to a global clock and data can only be transferred to a cell in a fixed clock period. For example, to invert a 2 × 2 matrix with Schur complement, let **E** be

$$\mathbf{E} = \mathbf{D} + \mathbf{C}\mathbf{A}^{-1}\mathbf{B},$$
$$\begin{bmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \end{bmatrix} = \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix} + \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^{-1} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}. \quad (7)$$

Then the matrix is fed into the systolic array in columns. **A** and **B** require *mode 1* cell operation, while **C** and **D** are computed in *mode 2*. The result can be obtained from the bottom row in skewed form that corresponds to the input sequence. Figure 3 gives an illustration.

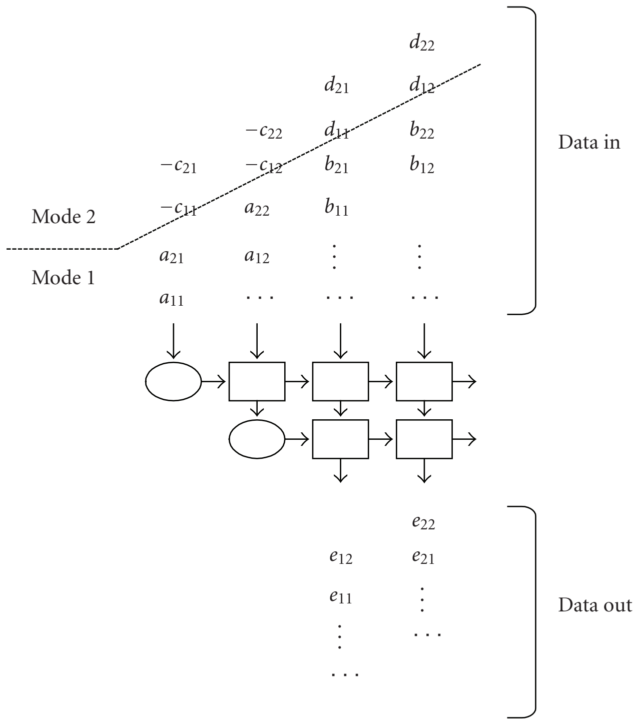


Figure 3: Dataflow in systolic array of 2 × 2 matrix size.

2.3. Modifying systolic array structure

A new systolic array can be constituted from other array structures to achieve certain specifications with the following four techniques [6].

(i) *Off-the-peg* maps the algorithm onto an existing systolic array directly. Data is preprocessed but the array design is preserved. However, data may be manipulated to ensure that the algorithm works correctly under array structure.

(ii) *Cut-to-fit* is to customise an existing systolic array to adjust for special data structures or to achieve specific system performance. In this case, data is preserved but array structure is modified.

(iii) *Ensemble* merges several existing systolic arrays into a new structure to execute one algorithm only. Both data and array structures are preserved, with dataflow transferring between arrays.

(iv) *Layer* is similar to the ensemble technique. Several existing systolic arrays are joined to from a new array, which switches its operation modes depending on the data. Only part of the new array will be utilised at one time.

In order to overcome the problem of the growth of the basic systolic array presented in Section 2.2 with the size of input matrices, a modified PSA is proposed in this section.

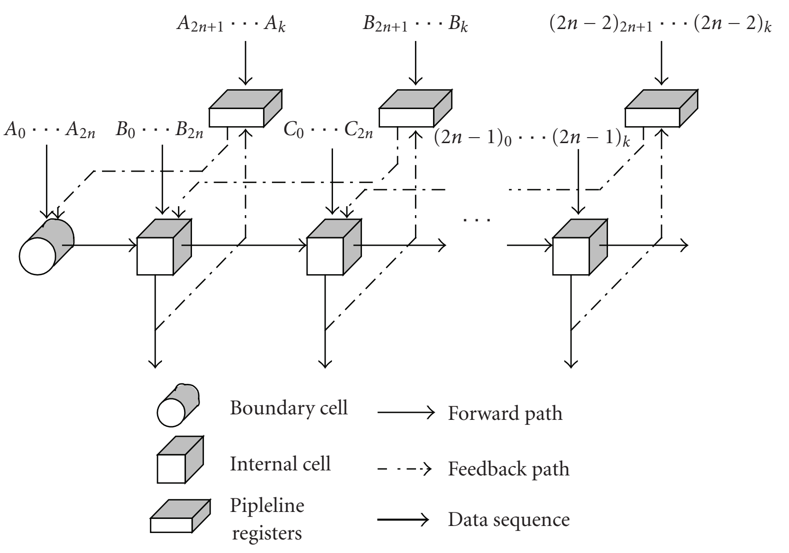


FIGURE 4: PSA dataflow in 3D visualization form.

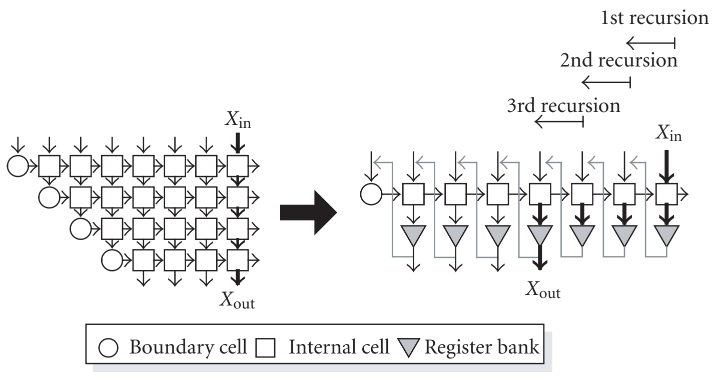


FIGURE 5: Demonstration of feedback dataflow.

When comparing two consecutive layers in the basic array from Figure 2, it can be noted that the cell arrangement is identical except the lower layer has one less internal cell than its immediate upper layer. This leads to the conclusion that the topmost layer is the only one that has the processing capabilities of all other layers and could be reused to do the function of any other layer given the appropriate input data into each cell. In other words, the topmost layer processing elements can be reused (shared) to implement functionality of any layer (logical layer) at different times. Obviously, for this to be possible, the intermediate results of calculation from logical layers have to be stored in temporary memories and made available for the subsequent calculation. The sharing of the processing elements of the topmost layer is achieved by transmitting the output data to the same layer through feedback paths and pipeline registers. The dataflow graph of the PSA is shown in Figure 4.

In the PSA, the regular lattice structure of basic systolic array is simplified to only include the first (topmost/physical) layer. Referring to Figure 4, data first enters in the single cell row and the outputs are passed to the registers in the same column. These registers, which store the temporary results, are connected in series and also provide feedback paths. The end of the register column connects to the input ports of the cell in the adjacent column and the feedback data becomes the input data of the adjacent cell. The corresponding dataflow paths in two different array structures are shown in Figure 5, highlighted in bold arrows. The data originally passing through the basic systolic array re-enters the same single processing layer four times during three recursions.

In order to implement the PSA structure for an $n \times n$ matrix, the required number of elements is

(i) the number of boundary cells $C_{bc} = 1$,
(ii) the number of internal cells $C_{ic} = 2n - 1$,
(iii) the number of layers in a column of register bank $R_L = 2(n - 1)$,
(iv) the total number of registers $R_{tot} = 2(n - 1)(2n - 1)$.

The exact structure of the PSA for the example from Figure 5 is presented in Figure 6. As can be seen when the input

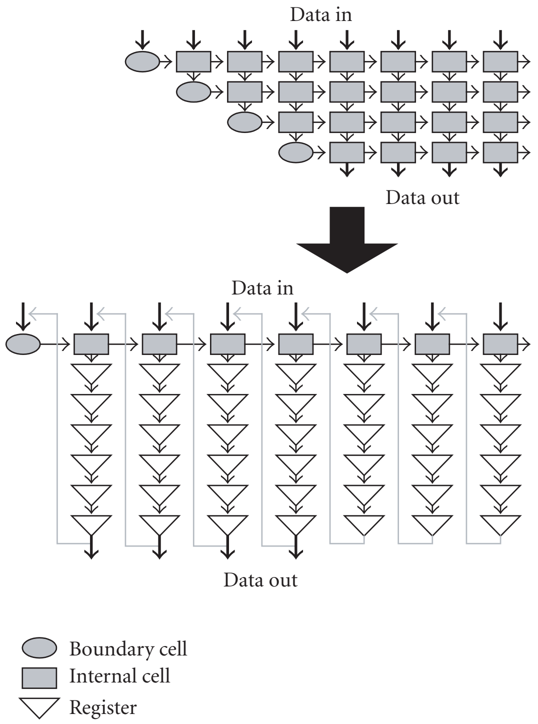


Figure 6: Modifying systolic array of PSA structure.

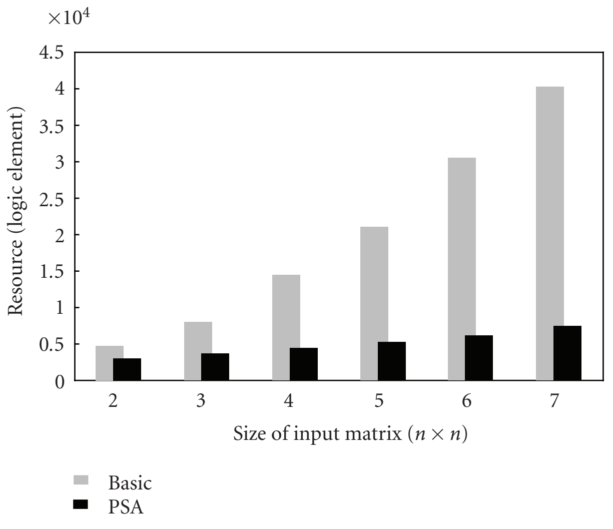


Figure 7: Logic resource usage comparison between the PSA and basic systolic array.

matrix size increases, the number of cells required to build the PSA increases by $O(n)$, which is much smaller than $O(n^2)$ as it is the case in other systolic array structures. The price paid is the number of additional registers used for storage of intermediate results. However, as the complexity of registers is much lower than that of systolic array cells, substantial savings in the implementation of the functionality can be achieved as it is illustrated in Figure 7 for different sizes of matrices. Resource utilisation is expressed in a number of logic elements of an FPGA device used for implementation.

3. DIVISION IN HARDWARE

3.1. Division with multiplication

Scalar division represents the most critical arithmetic operation within a processing element in terms of both resource utilisation and propagation delay. This is particularly typical for FPGAs, where a large number of logic elements are typically used to implement division. For the efficient implementation of division, which still satisfies accuracy requirements, an approach with the use of LUT and an additional multiplier has been proposed and implemented.

Noting that numerical result of "a divided by b" is the same as "a multiplied by $1/b$," the FPGA built-in multiplier can be used to calculate the division if an LUT of all possible values of $1/b$ was available in advance.

FPGA devices provide a limited amount of memory, which can be used for LUTs. Due to the fact that 1 and b can be considered integers, the value of $1/b$ falls into a decreasing hyperbolic curve, while b tends to one, and so the value difference between two consecutive numbers of $1/b$ decreases dramatically. To reduce the size of the LUT, the inverse value curve can be segmented into several sections with different mapping ratios. This can be achieved by storing one inverse value, the median of the group, in the LUT to represent the results of $1/b$ for a group of consecutive values of b. This process is illustrated in Figure 8. The larger the mapping ratio, the smaller amount of memory needed for the LUT. Obviously, such segmentation induces precision error. The way to segment the inverse curve is important because it directly affects the result accuracy. Further reduction in the memory size is achieved by storing only positive values in the LUT. The sign of the division result can be evaluated by an XOR gate.

On an Altera APEX device, when combining the LUT and multiplier into a single division module, a 16 bit by 26 bit multiplier consumes 838 logic elements (LEs), operating at 25 MHz clock frequency and total memory consumption of 53 248 memory bits for the specific target FPGA device. The overall speed improvement achieved through using the DLM method is 3.5 times when compared to using a traditional divider. Because of the extra hardware required for efficiently addressing the LUT, the improvement in terms of LEs is rather modest. The hardware-based divider supplied by Altera, configured as 16 bit by 26 bit, consumes 1 123 LEs when it is synthesised for the same APEX device.

3.2. Optimum segmentation scheme

Since b is a 16-bit number (used in 1.15 format), there are $(2^{15} - 1) = 32\,767$ different values of $1/b$. The performance of various linear and nonlinear segmentation approaches are evaluated in the priority of precision error and resource consumption.

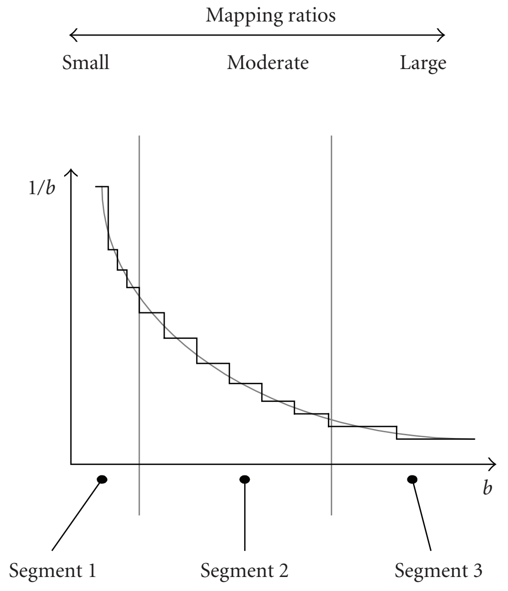


FIGURE 8: A simple demonstration of segments in different mapping ratios.

TABLE 1: The optimum segmentation scheme.

Segmentation	*Mapping ratio*
1–511	1 : 1
512–1 023	1 : 2
1 024–2 047	1 : 4
2 048–4 095	1 : 8
4 096–8 191	1 : 16
8 192–16 383	1 : 32
16 384–32 767	1 : 64

Absolute error is calculated by subtracting the true value of the inverse $1/b$ from the LUT output. Average error is the mean of the absolute error among the 32 767 data. Since the value of $1/b$ retrieved from the LUT is later multiplied by a in order to generate the division result, any precision error in LUT will be eventually magnified by the multiplier. Therefore, the worst-case error is more critical than the average precision error. The worst-case error can be calculated as follows: worst-case error of $1/b_k$ = absolute error of $(1/b_k) \times b_{k-1}$.

The error analysis was performed to investigate both the absolute error in average and the worst-case. As a result of this analysis an optimum segmentation scheme, tabulated in Table 1, was determined. It provides the minimum precision required of a typical hardware-implemented matrix inversion operation. This was verified by means of simulation using Matlab-DSP blockset for a number of applications. The resulting LUT holds 4 096 inverse values with a 26-bit word length in 16.10 data format.

4. PIPELINED SYSTOLIC ARRAY IMPLEMENTATION

The implementation block diagram of the PSA structure is shown in Figure 9. Datapath Architecture is illustrated in Figure 10. The interfacing of the control unit and the other internal and external cells are shown in Figure 11.

4.1. Control unit

The control unit is a timing module responsible for generating the control signals at specific time instances. It is synchronous to the system clock. Counters are the main components in the control unit. The I/O data of control unit are listed below.

Inputs

(i) 1-bit system clock: *clk* for synchronisation and the basic unit in timing circuitry.
(ii) 1-bit reset signal: *reset* to reset the control unit operation. Counters will be reset to the initial values and restart the counting sequences.

Outputs

(i) 1-bit cell operation signal mode to decide the cell operation mode: “1” for *mode 1* and “0” for *mode 2*.
(ii) 1-bit register clear signal: *clear* to activate the content-clear function in cell internal registers: “1” for enable and “0” for disable.
(iii) 1-bit multiplexer select signal: *sel* for controlling the input data sources selection in data path multiplexers: “1” for input from matrix and “0” for input from the feedback path.

Since the modules in the PSA are arranged in systolic structure and connected synchronously, generation of the control signals required to operate these modules should be also in regular timing patterns. Figure 12 demonstrates the required control signals for operating the PSA in different sizes.

5. DESIGN PERFORMANCE AND RESULTS

5.1. Resource consumption and timing restrictions

Compared to other systolic arrays in the literature, the small logic resource consumption is the main advantage of the proposed PSA structure. For example, for inverting an $n \times n$ matrix, the PSA requires to instantiate $2n$ cells while the systolic array in Figure 2 requires $(n^2 + \sum_{k=1}^{2n-1} k)$ cells.

Because of feedback paths in the design and single cell layer structure in the PSA, the number of processing elements required for implementation has been reduced and therefore the hardware complexity changed from $O(n^2)$ to $O(n)$.

A generic PSA has a customisable size and configurable structure. The final size of the PSA can be estimated by adding the resource consumption of each building block or

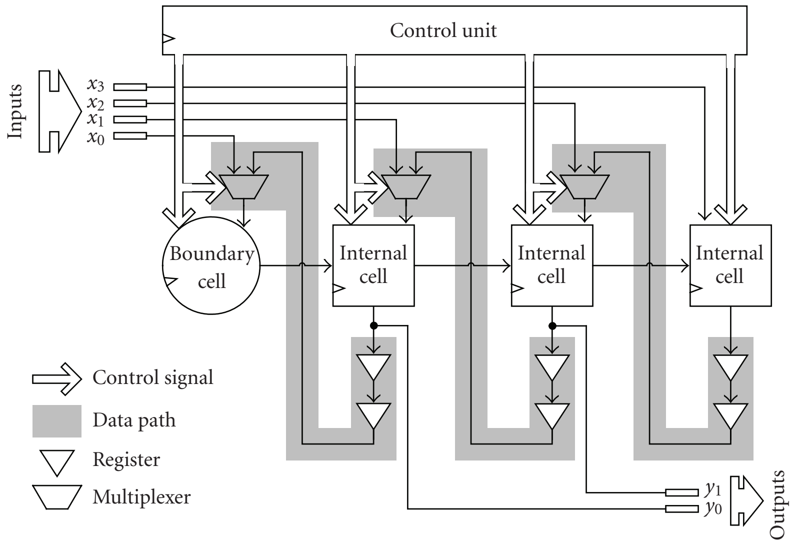


FIGURE 9: The PSA structure block diagram.

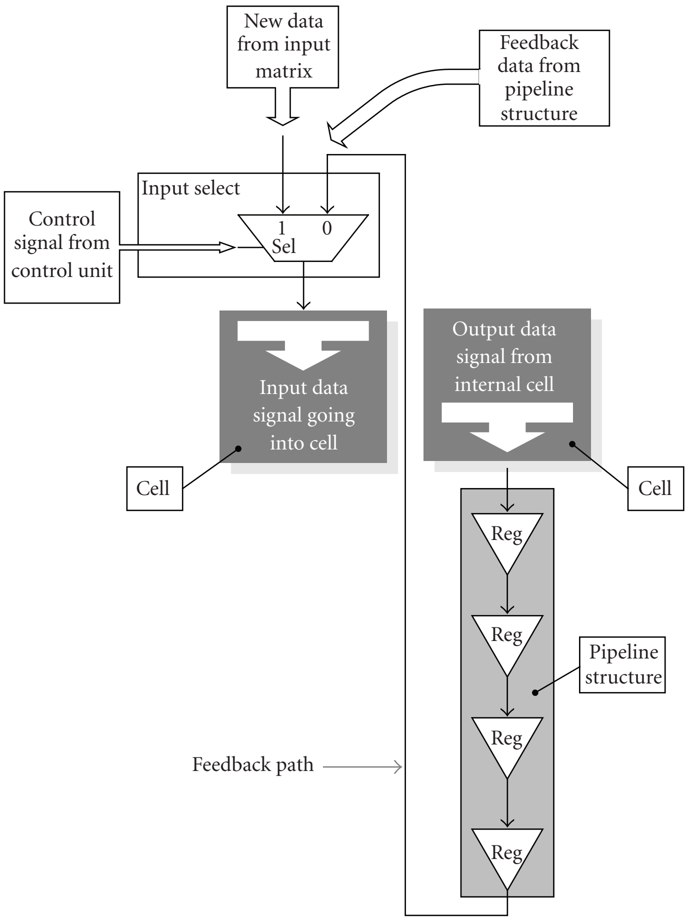

New data from input matrix
Feedback data from pipeline structure
Input select
Control signal from control unit
1
0
Sel
Input data signal going into cell
Output data signal from internal cell
Cell
Cell
Reg
Reg
Reg
Reg
Pipeline structure
Feedback path

FIGURE 10: Data-path architecture.

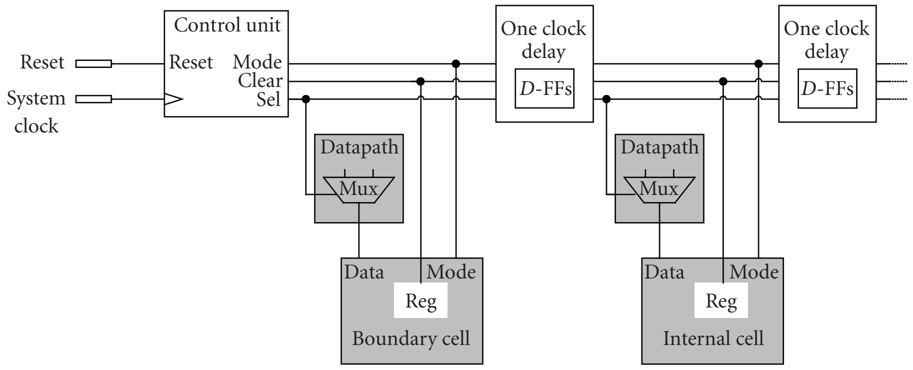


FIGURE 11: Control unit interfacing with other modules in PSA.

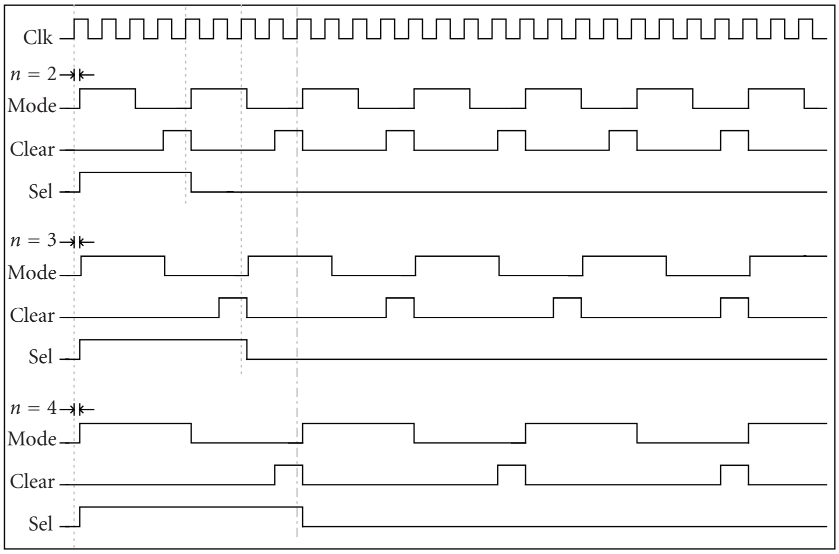


FIGURE 12: Timing diagram of control signals for different PSA sizes.

module as shown below for example:

$$\begin{aligned} \text{PSA size} &= \sum \text{size (boundary cell + internal cell} \\ &\qquad + \text{data path + control unit)} \\ &= \underbrace{(976)}_{\text{BoundryCell}} + \underbrace{(495I)}_{\text{InternalCell}} + \underbrace{(16R + 16M)}_{\text{DataPath}} \\ &\quad + \underbrace{(131 + 3D)}_{\text{ControlUnit}}[\text{LEs}], \end{aligned} \tag{8}$$

where I, R, M, and D represent the number of internal cells, 16-bit pipelining registers, 16-bit input select multiplexers, and 3-bit signal delay D-FFs, respectively. It should be noted that the actual size of the synthesised PSA on FPGA device will be affected by the architecture and routing resources of the FPGA.

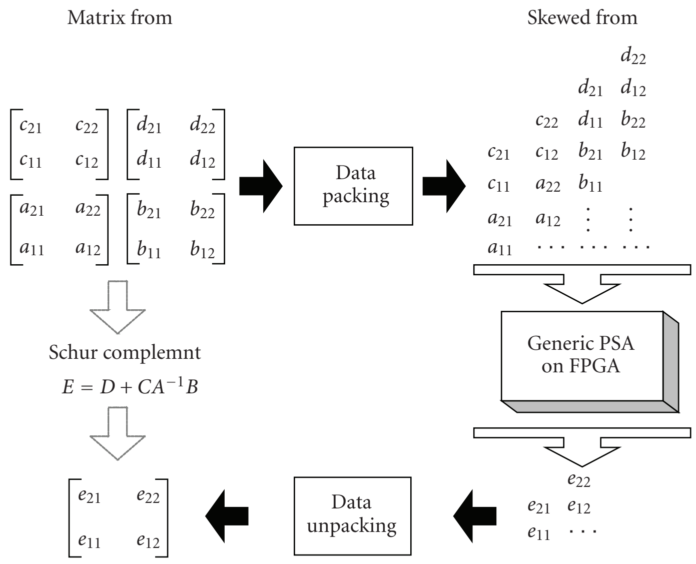
The processing time for the $n \times n$ matrix inversion in PSA is $2(n^2 - 1)$ clock cycles at a maximum clock frequency running at 16.5 MHz for $n < 10$ in our implementation (Altera APEX EP20K200EFC484-2). When a larger PSA is synthesised, the system clock period decreases as the critical path extends.

5.2. Comparisons with other implementations

The PSA performance has been compared with some other matrix inversion structures based on systolic arrays in terms of number of processing elements (or cells), number of cell types, logic element consumption, maximum clock frequency, and design flexibility.

For an $n \times n$ matrix inversion, the PSA requires $2n$ cells while $[n(3n + 1)/2]$ cells are used in the systolic array based on the Gauss-Jordan elimination algorithm [10]. In the PSA, cells are classified as either boundary or internal cells, while the processing elements in the matrix inversion array structure in [5] are divided into three different functional groups.

When working with a 4×4 matrix, it takes 4 784 LEs to implement the PSA on an Altera APEX device, while 8 610 LEs are used to implement the same in a matrix-based systolic algorithm engineering (MBSAE) Kalman filter [11].



Matrix from
Skewed from
c_{21} c_{22} c_{11} c_{12}
d_{21} d_{22} d_{11} d_{12}
a_{21} a_{22} a_{11} a_{12}
b_{21} b_{22} b_{11} b_{12}
Data packing
Generic PSA on FPGA
Schur complemnt
$E = D + CA^{-1}B$
Data unpacking
e_{21} e_{22} e_{11} e_{12}

FIGURE 13: Procedures for input data packing and output data unpacking.

When synthesised on an Altera APEX device (EP20K-200EFC484-2), PSA allows a maximum throughput of 16 MHz, compared to only 2 MHz in the design presented in the systolic array based design reported in [11] and 10 MHz in geometric arithmetic parallel processor (GAPP) in [12]. The PSA is designed to be customisable and parameterisable, but other systolic arrays in the literature were all fixed-size structures.

5.3. Limitations

In our design several built-in modules from the vendor library were used for basic dataflow control and arithmetic calculations. Therefore, the results reported in this paper are valid only for specific FPGA devices. However, as libraries provided by other FPGA vendors have equivalent functionalities readily available, the proposed design can be easily modified and ported to other FPGA device families.

One disadvantage of the PSA design is that input data has to be in skewed form before entering the array. When the PSA interfaces with other processors, a data wrapping preprocessing stage may be required to pack the data in the specific skewed form shown in Figure 13. Output data from the PSA are unpacked to rearrange the results back to regular matrix form.

5.4. Effects of the finite word length

The finite word length performance of the PSA structure was analysed. All quantities in the structure are represented using fixed-point numbers. It should be noted that only multiplication and division, which itself is computed by multiplication, will introduce round-off error [13]. Addition and subtraction do not produce any round-off noise. The approach used here was to follow the arithmetic operations in the different variables update equations and keep track of the errors which arise due to finite-precision quantisation. As described earlier in the paper, all the multiplication operations are performed using 26-bit long data. Computation results, as well as the data in the LUT, are of 26-bit long. To a large extent, this eliminates the possibility of overflow occurring with matrices of small size regardless of the actual data values. Simulation shows that the inverse of a matrix of size up to 10×10, and data represented with 26 bits, which is sufficient for most practical applications, can be computed with minimal error. Obviously, as the size of the matrix increases, the error also increases. However, as the proposed design is fully parameterised, the word length used in the computation can be accordingly increased, but it will result in higher FPGA resource usage.

6. KALMAN FILTER IMPLEMENTED USING PSA

6.1. Kalman filter

Since its introduction in the early 60s [14], Kalman filter has been used in a wide range of applications and as such it falls in the category of recursive least square (RLS) filters. As a powerful linear estimator for dynamic systems, Kalman filter invokes the concept of state space [15]. The main feature of the state-space concept allows Kalman filters to compute a new state estimate from the previous state estimate and new input data [16]. Kalman filter algorithms consist of six equations in a recursive loop. This means that results are continuously calculated step by step. To derive the Kalman filter equations, a mathematical model is built to describe the dynamics and the measurement system in form of linear equations (9) and (10).

(i) Process equation:

$$x(n+1) = \mathbf{A}x(n) + w(n). \tag{9}$$

(ii) Measurement equation:

$$s(n) = \mathbf{B}x(n) + v(n), \tag{10}$$

where $x(n)$ is the state at time instance n, $s(n)$ is the measurement at time instance n, $\mathbf{A}$ is the processing matrix, $\mathbf{B}$ is the measurement matrix, $w(n)$ is the system processing noise, and finally $v(n)$ is the measurement noise. In (9), $\mathbf{A}$ describes the plant and the changes of state vector $x(n)$ over time, while $w(n)$ is a plant disturbance vector of a zero-mean Gaussian white noise. In (10), $\mathbf{B}$ linearly relates the system states to the measurements, where $v(n)$ is a measurement noise vector of a zero-mean Gaussian white noise.

The Kalman filter equations can be grouped into two basic operations: prediction and filtering. Prediction, sometimes referred to as time update, estimates the new state and the uncertainty. An estimated state vector is denoted as $\hat{x}(n)$. When an estimate of $\hat{x}(n)$ is computed before the current measurement data $s(n)$ become available, such estimate is classified as an a priori estimate and denoted as $\hat{x}(n)$. When the estimate is made after the measurement $s(n)$ arrives, it is called a posteriori estimate [16]. On the other hand, filtering, usually referred to as measurement update, is to correct the previous estimation with the arrival of new measurement data. The prediction error can be computed from the difference between the value of actual measurements and the estimated value. It is used to refine the parameters in a prediction algorithm immediately in order to generate a more accurate estimate in the future. The full set of Kalman filter equations can be found in [17].

It is evident from the Kalman filter equations that its algorithm comprises a set of matrix operations, including matrix addition, matrix subtraction, matrix multiplication, and matrix inversion. Among these matrix operations, matrix inversion is the most computationally expensive and thus being the bottleneck in the processing time of the algorithm such that the overall system processing time mainly depends on matrix inversion speed [10]. In Section 2, a new implementation of matrix inversion, which is in fact the "heart" of Kalman filter, was presented. Hardware implementation of another critical operation, division, was presented in Section 3.

Table 2: Matrix substitutions for Kalman filter algorithms.

		Schur complement	Result
Step 1	A	$\mathbf{I}$	$\hat{x}^{-}(n \mid n-1)$
	B	$\hat{x}(n-1 \mid n-1)$	
	C	$\mathbf{A}$	
	D	$\mathbf{0}$	
Step 2	A	$\mathbf{I}$	$\mathbf{AP}(n-1 \mid n-1)$
	B	$\mathbf{P}(n-1 \mid n-1)$	
	C	$\mathbf{A}$	
	D	$\mathbf{0}$	
Step 3	A	$\mathbf{I}$	$\mathbf{P}^{-}(n \mid n-1)$
	B	$\mathbf{A}^T$	
	C	$\mathbf{AP}(n-1 \mid n-1)$	
	D	$\mathbf{Q}(n-1)$	
Step 4	A	$\mathbf{I}$	$\mathbf{P}^{-}(n \mid n-1)\mathbf{B}^T$
	B	$\mathbf{B}^T$	
	C	$\mathbf{P}^{-}(n \mid n-1)$	
	D	$\mathbf{0}$	
Step 5	A	$\mathbf{I}$	$\mathbf{BP}(n \mid n-1)\mathbf{B}^T + \mathbf{R}(n)$
	B	$\mathbf{P}^{-}(n \mid n-1)\mathbf{B}^T$	
	C	$\mathbf{B}$	
	D	$\mathbf{R}(n)$	
Step 6	A	$\mathbf{BP}(n \mid n-1)\mathbf{B}^T + \mathbf{R}(n)$	$\mathbf{K}(n)$
	B	$\mathbf{I}$	
	C	$\mathbf{P}^{-}(n \mid n-1)\mathbf{B}^T$	
	D	$\mathbf{0}$	
Step 7	A	$\mathbf{I}$	$\mathbf{P}(n \mid n)$
	B	$[\mathbf{P}^{-}(n \mid n-1)\mathbf{B}^T]^T$	
	C	$-\mathbf{K}(n)$	
	D	$\mathbf{P}^{-}(n \mid n-1)$	
Step 8	A	$\mathbf{I}$	$s(n) - \mathbf{B}\hat{x}^{-}(n \mid n-1)$
	B	$\hat{x}^{-}(n \mid n-1)$	
	C	$-\mathbf{B}$	
	D	$s(n)$	
Step 9	A	$\mathbf{I}$	$\hat{x}(n \mid n)$
	B	$s(n) - \mathbf{B}\hat{x}^{-}(n \mid n-1)$	
	C	$\mathbf{K}(n)$	
	D	$\hat{x}^{-}(n \mid n-1)$	

6.2. Kalman filter in PSA-based structure

As a case study to verify the performance of the proposed PSA, a Kalman-filter-based echo cancellation application was implemented. By appropriate substitutions of matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ (Table 2), matrix-form Kalman filter equations can be computed by the PSA in 9 steps. A complete execution of the 9 steps produces state estimates in the next time instance and constitutes one recursion in the Kalman filter algorithm.

The components of the four input matrices are queued in a skewed package entering the PSA cells row by row. It can be noted from Table 2 that some Schur complement results will be used as input data in later steps. Thus, extra registers are required to store the intermediate results. To ensure that the intermediate results are reloaded to specific cells at the correct time instances, a new data path and control unit is created. In the existing PSA structure, data in $\mathbf{A}$ and $\mathbf{C}$ are aligned in the same column entering to the cells in left-half group, while $\mathbf{B}$ and $\mathbf{D}$ are in another column toward the right-half cells group. Along the feedback paths, the result, $\mathbf{E} = \mathbf{D} + \mathbf{CA}^{-1}\mathbf{B}$, is connected to the same columns of $\mathbf{A}$ and $\mathbf{C}$ as shown in Figure 14. In this case, the intermediate result cannot be used as the input data for $\mathbf{B}$ and $\mathbf{D}$. Therefore, a new data path with an input multiplexer is added to allow $\mathbf{E}$ passing to cells in right-half group. A control unit is required to switch the multiplexer input sources between intermediate result $\mathbf{E}$ and new data from $\mathbf{B}$ and $\mathbf{D}$. The modified design is presented with thick lines in Figure 15.

The results obtained from the echo cancellation application using the PSA-based Kalman filter closely match the

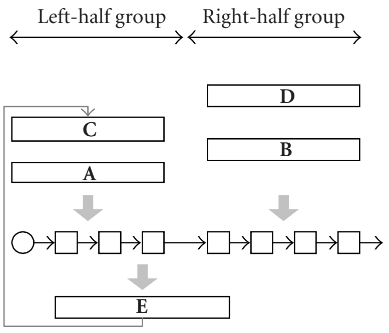


Figure 14: The original data paths of PSA.

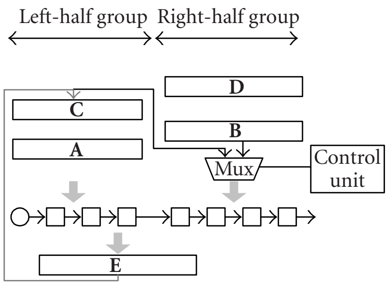


Figure 15: The new data paths of a PSA-based Kalman filter.

theoretical values. The small residual error observed in the resulting data, is contributed to the finite word length effect typical of fixed-point structure of the proposed design.

6.3. Comparison with other implementations

There are several hardware implementations for Kalman filter in the literature. For a 4-state Kalman filter, all the Kalman filter equations can be expressed as 30 scalar equations. Similar to the PSA, direct operation of matrix inversion is also avoided in the matrix decomposition method (MDM) and the Kalman gain calculation turns into a set of 4 scalar equations with scalar division and addition. With the high processing speed of 169.4 nanoseconds reported in [18], MDM seems to have a better speed over the PSA (280 nanoseconds) for the same target APEX device. However, the PSA structure still enjoys the following advantages.

Flexibility

When the number of states in a Kalman filter changes, all the scalar equations in MDM become invalid as matrix dimensions in the algorithm depend on the size of the state vector. Considerable design time is required to decompose the matrix-form equations again. However, in the PSA, a Kalman filter with different number of states can be generated by modifying one parameter (number of states, i.e., the matrix size) in the heading of the VHDL code. The PSA serves as an IP block for a generic Kalman filter in VHDL, while MDM is a hard-wired implementation for a fixed Kalman filter.

Clock speed

The advantages and the conditions of using LUT with multiplier to perform scalar division has been discussed in Section 3.2. This approach enables PSA to have a system clock frequency 3.5 times faster than using scalar dividers only.

Resource usage

In the MDM method, 32 operations of addition/subtraction, 22 multiplications, and 4 divisions are involved in scalar operations. The overall logic element usage of the PSA is 40% lower than an equivalent MDM-based design for a 4-state Kalman filter implementation.

7. CONCLUSIONS

In this paper, an optimised systolic-array-based matrix inversion for implementation in FPGA was proposed and used for rapid prototyping of a Kalman filter. Matrix inversion is the computational bottleneck and the most complex operation in Kalman filtering. The PSA matrix inversion results in a simple, yet fast, implementation of the operation. It is scalable to matrices of various sizes and is implemented as a parameterised design. This allows its direct customisation and instantiation for application-specific problems. Resource utilisation is low and linearly depends on the matrix size.

Modified from the Schur complement systolic array, the PSA simplifies recursive matrix-form equations in Kalman filters to scalar operations and inherits the design advantages of parallelism and pipelining. In the proposed PSA design, a new approach for implementation of scalar division has also been proposed, which speeds up the division operation 3.5 times over traditional dividers and yet uses less logic elements and resources to implement.

REFERENCES

[1] G. W. Irwin, "Parallel algorithms for control," *Control Engineering Practice*, vol. 1, no. 4, pp. 635–643, 1993.

[2] M. Ceschia, M. Bellato, A. Paccagnella, and A. Kaminski, "Ion beam testing of ALTERA APEX FPGAs," in *Proceedings of IEEE Radiation Effects Data Workshop*, pp. 45–50, Phoenix, Ariz, USA, July 2002.

[3] A. El-Amawy, "A systolic architecture for fast dense matrix inversion," *IEEE Transactions on Computers*, vol. 38, no. 3, pp. 449–455, 1989.

[4] A. K. Ghosh and P. Paparao, "Performance of modified Faddeev algorithm on optical processors," *IEE Proceedings. J: Optoelectronics*, vol. 139, no. 5, pp. 325–330, 1992.

[5] M. Zajc, R. Sernec, and J. Tasic, "An efficient linear algebra SoC design: implementation considerations," in *Proceedings of 11th Mediterranean Electrotechnical Conference (MELECON '02)*, pp. 322–326, Cairo, Egypt, May 2002.

[6] F. M. F. Gaston and G. W. Irwin, "Systolic Kalman filtering: an overview," *IEE Proceedings. D: Control Theory & Applications*, vol. 137, no. 4, pp. 235–244, 1990.

[7] F. M. F. Gaston, D. W. Brown, and J. Kadlec, "A parallel predictive controller," in *Proceedings of UKACC International*

Conference on Control, vol. 2, pp. 1070–1075, Exeter, UK, September 1996.

[8] A. El-Amawy and K. R. Dharmarajan, “Parallel VLSI algorithm for stable inversion of dense matrices,” *IEE Proceedings. E: Computers and Digital Techniques*, vol. 136, no. 6, pp. 575–580, 1989.

[9] N. Faroughi and M. A. Shanblatt, “An improved systematic method for constructing systolic arrays from algorithms,” in *Proceedings of 24th ACM/IEEE Design Automation Conference (DAC ’87)*, pp. 26–34, Miami Beach, Fla, USA, June–July 1987.

[10] S.-G. Chen, J.-C. Lee, and C.-C. Li, “Systolic implementation of Kalman filter,” in *Proceedings of IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS ’94)*, pp. 97–102, Taipei, Taiwan, December 1994.

[11] Z. Salcic and C.-R. Lee, “Scalar-based direct algorithm mapping FPLD implementation of a Kalman filter,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 3, part 1, pp. 879–888, 2000.

[12] D. Lawrie and P. Fleming, “Fine-grain parallel processing implementations of Kalman filter algorithms,” in *Proceedings of International Conference on Control*, vol. 2, pp. 867–870, Edinburgh, Scotland, UK, March 1991.

[13] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, McGraw-Hill/Irwin, Boston, Mass, USA, 2nd edition, 2001.

[14] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Transaction of the ASME, Series D, Journal of Basic Engineering*, vol. 82, pp. 35–45, March 1960.

[15] S. V. Vaseghi, *Advanced Digital Signal Processing and Noise Reduction*, John Wiley & Sons, New York, NY, USA, 2nd edition, 2000.

[16] E. W. Kamen and J. K. Su, *Introduction to Optimal Estimation*, Springer, London, UK, 1999.

[17] D. C. Swanson, *Signal Processing for Intelligent Sensor Systems*, Marcel Dekker, New York, NY, USA, 2000.

[18] C.-R. Lee, *FPLD implementation and customisation in multiple target tracking applications*, Engineering Ph.D. thesis, the University of Auckland, Auckland, New Zealand, 1998.

Abbas Bigdeli was born in Ahvaz, Iran in 1973. He received a Bachelor in electronics engineering in 1995 from the Department of Electrical Engineering, Amir Kabir University of Technology, Tehran, Iran. He started his postgraduate studies at James Cook University, Australia, in 1996. He concluded his Ph.D. research in 2000 and moved to Auckland, New Zealand, to join the Faculty of Engineering at The University of Auckland. His current research interests are in the area of reconfigurable embedded and network processors, security solutions for wireless networks, hardware/software implementation of image and video processing and design, and fabrication of intelligent implantable medical devices. He has published over 35 scientific and technical papers in international journals and conferences. He has recently patented an invention on securing legacy 802.11 wireless LAN systems. He is the Program Leader for electronics projects at Polymer Electronic Research Centre at The University of Auckland. He has been on the Executive Committee of IEEE New Zealand North Section since 2001. He has been acting as a Technical Reviewer for several journals and conferences. These include the Journal of Microprocessors and Microsystems, Australian Journal of Research and Practice in Information Technology, as well as FPL, IEEE VLSI, and EUSIPCO conferences.

Morteza Biglari-Abhari received the B.S. degree from Iran University of Science and Technology, M.S. degree from Sharif University of Technology in Tehran, and Ph.D. degree from The University of Adelaide in Australia. Currently, he is Senior Lecturer in the Department of Electrical and Computer Engineering at The University of Auckland in New Zealand. His main research interests are computer architecture, multiprocessor system-on-chips, compiler optimisations, and hardware/software codesign for low-power embedded systems. He is also a Member of the Steering Committee of Polymer Electronic Research Centre (PERC) at The University of Auckland. He has been Chair of the IEEE Computer Chapter (New Zealand North Section) since 2004 and Reviewer of some technical journals and conferences such as Journal of Microprocessors and MicroSystems, EURASIP Journal on Applied Signal Processing, and FPL, VLSI, ISSPA, and EUSIPCO conferences.

Zoran Salcic is a Professor of computer systems engineering at The University of Auckland, New Zealand. He holds the B.E., M.E. and Ph.D. degrees in electrical and computer engineering from the University of Sarajevo received in 1972, 1974, and 1976, respectively. He did most of the Ph.D. research at the City University of New York in 1974 and 1975. He has been with the academia since 1972, with the exception of years 1985–1990 when he took the posts in the industrial establishment, leading a major industrial enterprise institute in the area of computer engineering. His expertise spans the whole range of disciplines within computer systems engineering: complex digital systems design, custom computing machines, reconfigurable systems, field programmable gate arrays, processor and computer systems architecture, embedded systems and their implementation, design automation tools for embedded systems, hardware/software codesign, new computing architectures, and models of computation for heterogeneous embedded systems and related areas. He has published more than 170 refereed journal and conference papers and numerous technical reports. He has supervised six Ph.Ds and more than 40 M.E. thesis completions and took part in numerous Ph.D. and M.E. examinations. He is the Founding Editor-in-Chief of the new EURASIP Journal on Embedded Systems.

Yat Tin Lai received his B.E. and M.E. degrees in computer systems engineering from The University of Auckland in 2002 and 2004, respectively.