AN OPTIMISED SYSTOLIC ARRAY-BASED MATRIX INVERSION FOR RAPID PROTOTYPING OF KALMAN FILTERS IN FPGA'S

Yat Tin Lai, Abbas Bigdeli and Morteza Biglari-Abhari

Department of Electrical and Computer Engineering, The University of Auckland
Private Bag 92019, Auckland, New Zealand
phone: +64 9 373 7599 ext 88156, fax: +64 9 373 7461, email: abbas.bigdeli@ieee.org
web: www.ece.auckland.ac.nz/embedded

ABSTRACT

In this paper we propose a pipelined structure for systolic array-based matrix inversion. The main focus is to optimise the systolic structure for rapid prototyping of Kalman filters in FPGA devices. The design is implemented in VHDL language enabling potential users to effectively customise the structure for different size Kalman filters suitable for different applications. The proposed solution consists of pipeline registers, an innovative logic control unit, and a segmented Look Up Table based division scheme. The new architecture has an advantage of O(2n) resource consumption, compared to the $O(n^2)$ in other systolic array structures. The resulting precision error is within an acceptable range.

1. INTRODUCTION

Over the past forty years, Kalman filters have been widely used in many applications such as target tracking, navigation systems, adaptive control and many other dynamic systems. Kalman filter algorithm is based on minimising the mean-square error recursively. Therefore, it can work well in estimating the unknown states of a noisy dynamic process [1].

The algorithm of an adaptive Kalman filter involves several iterative matrix manipulations such as matrix inversion, multiplication, addition and subtraction. Real-time implementation of Kalman filters is hence limited by the computationally extensive nature of the algorithm. Many attempts have been made to employ various systolic architectures for VLSI implementation of Kalman filters [2], [3]. These methods along with a dozen others presented in the literature are merely for permanent structures and are only suitable for VLSI implementation. Rapid prototyping of a Kalman filter-based system requires the implementation to be parameterisable. Systolic-based architectures should be modified to meet hardware requirements of the Field Programmable Gate Array (FPGA) technology [4]. This paper investigates the design and hardware implementation of a generic Kalman filter in VHDL language where user is able to set the parameters to change the state number of the filter. Result accuracy, time performance, resource utilisation and the flexibility of customisation are the main concerns in the research.

2. KALMAN FILTER

Kalman filter is in the family of recursive least-squares (RLS) filters, which means the solution has to be computed step by step recursively [5]. With its distinctive feature of state-space concept, Kalman filter is a powerful linear estimator for dynamic systems [6]. The main feature is that only the previous state estimate and the new input data are required to generate the new state estimate in each computation cycle, which results in a low memory requirement [7]. Kalman filter algorithm has two basic operations; *prediction* and *filtering*, both executing in a single cycle recursively. *Prediction* is to predict the new states and uncertainty while *filtering* is to correct the prediction with the new measurements.

The Kalman filter equations are a set of matrix operations, including matrix addition, matrix subtraction, matrix multiplication and matrix inversion. Among these matrix operations, matrix inversion is computationally expensive and thus being the bottleneck in processing time of the algorithm [2]. Thus, this paper concentrates on FPGA implementation of matrix inversion, which is in fact the "heart" of Kalman filter.

3. IMPLEMENTATION OF MATRIX INVERSION

Plain matrix inversion is needed in numerous signal processing applications. Consequently, it is no surprise to find extensive literature on various methods for efficient implementation of matrix inversion [8].

3.1 Schur Complement in Systolic Array

According to Fadeev algorithm, Schur complement of $\mathbf{D+CA^{-1}B}$ can be utilised to calculate the matrix operations involved in the Kalman filter equations [9]. Simply setting matrix **D** as zero and both matrix **C** and B as identity matrix can calculate the inverse of matrix **A**. Schur complement can be implemented in a systolic array structure, which is built up by two types of processing element (PE), boundary cell and internal cell [9]. The systolic array structure for a 2×2 matrix is shown in Figure 1, where the operations of PEs are described in Table 1.

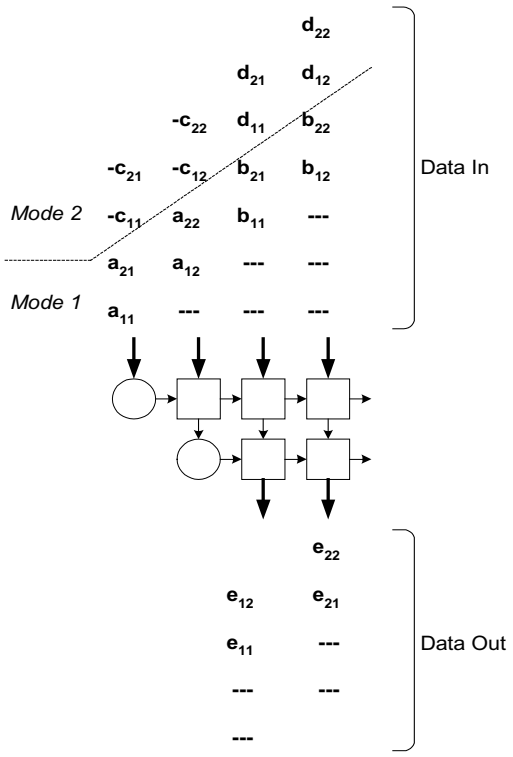

Figure 1: The systolic array structure for a 2×2 matrix.

		Boundary Cell	Internal Cell
Input		X, P	X, S, C, P
Output	Mode 1	if | X | > | P | : 1, - P / X	if S = 1 : X, S, C, P + C*X
		else : 0, - X / P	else : P, S, C, X + C*P
	Mode 2	always : 0, - X / P	always : P, S, C, X + C*P

Table 1: Operations of PEs.

3.2 Modified Systolic Array Structure

A modified systolic array, named as pipelined systolic array (PSA) is proposed in this paper and depicted in Figure 2. The PSA combines the multiple PE layers of the original systolic array into a single PE layer with a set of feedback registers in a pipeline structure.

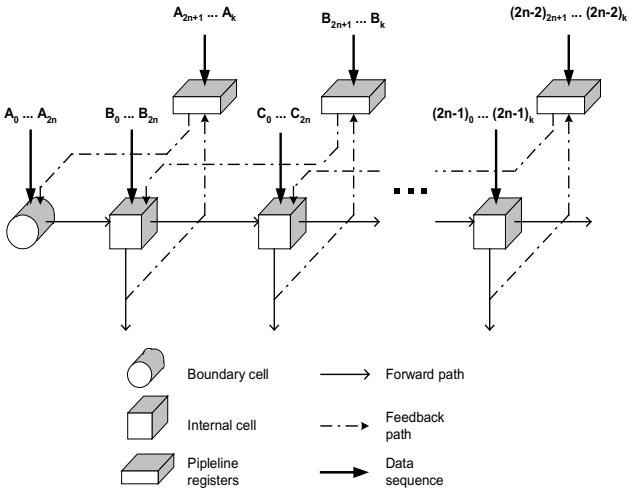

Figure 2: PSA design in 3-D visualisation form.

Data, originally passed to the PEs in the next layer, now enter the pipeline registers and later is fed back to the same PEs layer in PSA. An extra module of control signal logic circuit is required in order to produce control signals for data input selection, cell memory clearance and operation mode control. The main transform from the original systolic array structure, as adopted by many authors in the literature, to the proposed PSA is highlighted in Figure 3.

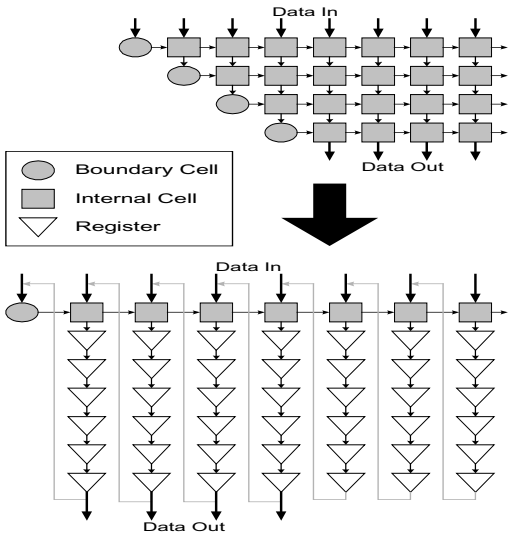

Figure 3: Modification process for PSA structure.

The resource consumption is reduced from $O(n^2)$ to $O(2n)$ where n is the size of n×n matrix [2]. The efficiency of PSA can be further increased to 100% after the feedback data re-enter the PEs. This is because all the PEs in the PSA are utilised at the same time. However, the number of clock cycles required to complete an n×n matrix inversion is in order of $O(n^2)$. The PSA allows the usage of Look-Up-Table (LUT) to replace a fix-point divider to enable a faster clock rate. The comparison between the direct implementation of divider and use of a LUT will be examined in the next section.

4. DIVISION IN HARDWARE

For the arithmetic calculations involved in the each PE, scalar division in digital hardware design is the most resource consuming and time consuming process. An LUT is a good option to approximate the division result, but precision error and memory usage problem can limit this approach. Therefore, it is worth investigating the advantages and disadvantages of using LUT for scalar division calculation.

4.1 Direct Division

Many FPGA design tools such as Altera Quartus II 3.0, provide built-in modules for arithmetic operations. These modules are usually optimised for the target architecture. For the purpose of matrix inversion, to carry out the scalar division in the boundary cell,

$$\mathbf{c} = \mathbf{a} / \mathbf{b}$$

where $0 <= |\mathbf{a}| < |\mathbf{b}| <= 1$ and **a**, **b** and **c** are 16 bit number, firstly, **a** has to be zero-padded to 32 bit long and then passed to a 32-bit divider. This arrangement occupies 1123 Logic Elements (LE) and executes at the maximum clock rate of 6.18 MHz in an Altera APEX device. The 16-bit finite word length induces a negligible precision error of only 0.0015%.

4.2 Division using a Look-Up-Table

Most FPGA devices provide efficient and optimised built-in multipliers. Noting that numerical result of **a** divided by **b** is the same as **a** multiplying by **1 / b**, the 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 only a limited memory, which limits the size of the LUT for this purpose. Due to the fact that the value of **1 / b** falls into a decreasing quadratic curve, while **b** tends to one, 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 better illustrated in Figure 4.

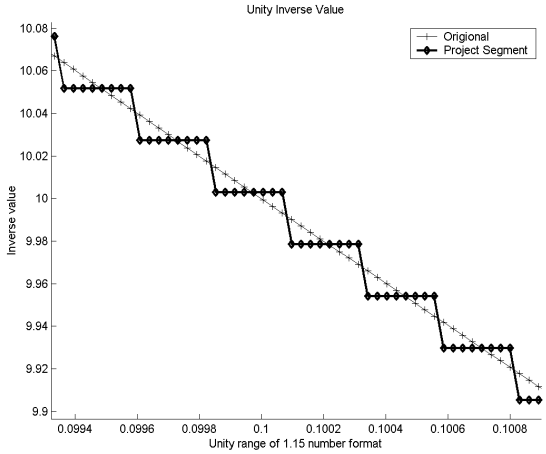

Figure 4: Segmentation for LUT.

By mapping the inverse values for positive numbers only, the size of LUT can be further minimised by 50%. The sign of the division result can be worked out later with a simple XOR gate logic.

4.3 Optimum Segmentation Approach

Since **b** is a 16-bit number in 1.15 format, there are totally $(2^{15} - 1) = 32767$ different values of **1 / b**. Table 2 presents the mapping ratios for five different segmentation methods, namely *Seg-1* to *Seg-5*.

Name	*Segmentation*	*Mapping Ratio*
Seg-1	no segmentation	1 : 8
Seg-2	1 – 1023	1 : 1
	1024 – 2047	1 : 2
	2048 – 4095	1 : 4
	4096 – 8191	1 : 8
	8192 – 32767	1 : 16
Seg-3	1 – 511	1 : 1
	512 – 1023	1 : 2
	1024 – 2047	1 : 4
	2048 – 4095	1 : 8
	4096 – 8191	1 : 16
	8192 – 16383	1 : 32
	16384 – 32767	1 : 64
Seg-4	1 – 511	1 : 1
	512 – 1023	1 : 2
	1024 – 2047	1 : 4
	2048 – 4095	1 : 8
	4096 – 8191	1 : 16
	8192 – 16383	1 : 32
	16384 – 24575	1 : 64
	24576 – 32767	1 : 128
Seg-5	1 – 511	1 : 1
	512 – 1023	1 : 2
	1024 – 2047	1 : 4
	2048 – 4095	1 : 8
	4096 – 8191	1 : 16
	8192 – 9362	map to constant value 1/4
	9363 – 13107	map to constant value 1/3
	13108 – 21845	map to constant value 1/2
	21846 – 32767	map to constant value 1

Table 2: Various Segmentation schemes.

Since the value of **1 / b** retrieved from the LUT is then multiplied by **a**, any precision error will be magnified. Therefore, it is important to consider the worst-case error. Table 3 presents a comparison of the various mapping schemes in Table 2.

	Seg-1	*Seg-2*	*Seg-3*	*Seg-4*	*Seg-5*
Average error	0.13%	0.78%	**0.07%**	0.08%	11.87%
Worst case error	300%	3.03%	**0.21%**	0.26%	49.99%
Resource (bit)	53248	81920	**53248**	61504	47740

Table 3: Precision error and resource comparison

Figures in Table 3 suggest *Seg-3* would be the natural choice for composition of the LUT. The LUT holds 4096 inverse values with a 26-bit word length in 16.10 data format. Figure 5 shows the error analysis over the whole date range.

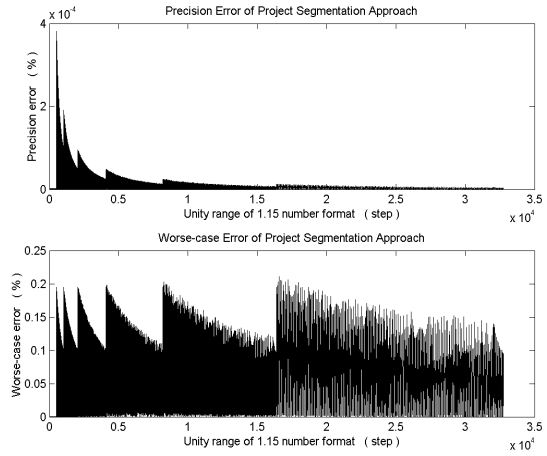

Figure 5: Error analysis of *Seg-3*.

5. PERFORMANCE ANALYSIS

By applying LUT to the PSA structure, a generic hardware design is established for matrix inversion in different sizes. The result of matrix operations, including addition, subtraction, multiplication and inversion, is calculated in 1.15 number format. For an n×n matrix, there is a total of 2n PEs in the PSA, with 1 boundary cell and 2n – 1 internal cells, and 2(n – 1) layers of feedback registers. The processing time requires $2(n^2 - 1)$ clock cycles. This implementation can run with a maximum clock frequency of 16.5 MHz for n<10.

Benchmark is carried out to compare the PSA performance with other matrix inversion systems. For an n×n matrix, the PSA requires 2n PEs while in [2], {n× (3n + 1) / 2} PEs are used. PSA allows the clock speed running at 16MHz, comparing to the maximum clock frequency of 2MHz in VHDL design presented in [4] and 10MHz in Geometric Arithmetic Parallel Processor (GAPP) in [11]. When it comes to resource consumption, the PSA approach is still superior compared to those presented in the literature. When working with a 4×4 matrix, it takes 4784 LE to implement the PSA in Altera APEX device while 8610 LE is required to implement the hardware design as reported in [4].

6. KALMAN FILTER IMPLEMENTATION

Since the PSA is able to calculate the Schur complement $\mathbf{D+CA^{-1}B}$, the Kalman filter equations can be computed with the existing PSA by appropriately substituting **A, B, C** and **D**. However, registers are needed to store the intermediate result of $\mathbf{D + CA^{-1}B}$ generated in each cycle, because Kalman filter equations require several cycles to be implemented using the Schur complement. Since an intermediate result register will not consume significant resource (16 LE, <0.2%), a generic Kalman filter when implemented using the proposed PSA structure can easily fit in a typical FPGA device. Figure 6 shows an estimate of resource consumption (LE) and maximum clock speed (MHz) for Kalman filter with different numbers of state (n).

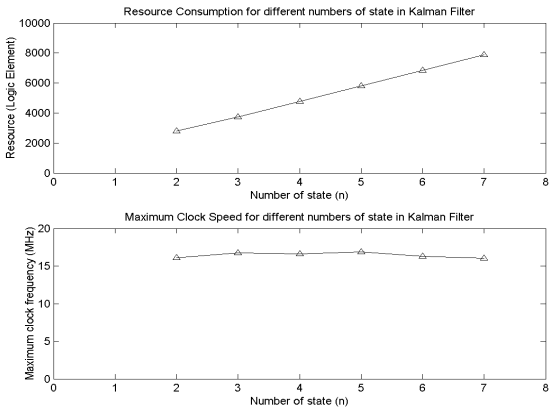

Figure 6: Resource estimate for PSA-based Kalman filters.

7. CONCLUSION

A customisable structure for an n-state Kalman filter has been designed in VHDL and partly implemented in an Altera APEX device with smaller logic resource space. By applying Fadeev's algorithm, matrix manipulations can be performed via the Schur complement. This paper presented a generic Pipeline Systolic Array structure with an advantage of O(2n) resource consumption, compared to the $O(n^2)$ in fixed-size systolic array structures reported in the literature. The resulting precision error is within an acceptable range.

REFERENCES

[1] S. Haykin, *Adaptive Filter Theory*, 4th Edition, Prentice Hall, USA, 2002.

[2] S-G. Chen, J-C. Lee and C-C. Li, "Systolic Implementation of Kalman Filter", *Circuits and Systems, APCCAS '94, IEEE Asia-Pacific Conference*, pp 97-102, 1994.

[3] C.J.B. Fayomi, M. Sawan and S. Bennis, "Parallel VLSI Implementation of A New Simplified Architecture of Kalman Filter", *Electrical and Computer Engineering, 1995. Canadian Conference*, Vol 1, pp 117 – 119, 1995.

[4] Z. Salsic and C.R. Lee, "Scalar-based direct algorithm mapping FPLD implementation of a Kalman filter", *Aerospace and Electronic Systems, IEEE Transactions on*, Volume: 36 Issue: 3, pp 879-888, 2000.

[5] D.C. Swanson, *Signal Processing for Intelligent Sensor Systems*, Marcel Dekker Inc., New York, 2000.

[6] S.V. Vaseghi, *Advanced Digital Signal Processing and Noise Reduction*, 2nd edition, John Wiley & Sons Ltd., 2000.

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

[8] El-Amawy, "A Systolic Architecture for Fast Dense Matric Inversion", *IEEE Transactions on Computers*, Vol. 38, NO. 3, March 1989.

[9] G.W., Irwin, "Parallel algorithms for control", *Control Eng. Practice*, Vol 1, no 4, 1993.

[10] C.R. Lee, "FPLD Implementation and Customisation in Multiple Target Tracking Applications", *Engineering PhD Thesis*, The University of Auckland, 1998.

[11] D. Lawrie and P., Fleming, "Fine-grain parallel processing implementations of Kalman filter algorithms", *Control '91., International Conference* , Vol 2, pp 867 – 870, 1991.