MSGR-based Low Latency Complex Matrix Inversion Architecture

Lei Ma, Kevin Dickson, John McAllister, John McCanny
Institute of Electronics, Communications and Information Technology, Queen's University, Belfast
{lma04, k.dickson, j.mcallister, j.mccanny}@ecit.qub.ac.uk

Abstract

This paper presents a matrix inversion architecture based on the novel Modified Squared Givens Rotations (MSGR) algorithm, which extends the original SGR method to complex valued data, and also corrects erroneous results in the original SGR method when zeros occur on the diagonal of the matrix either initially or during processing. The MSGR algorithm also avoids complex dividers in the matrix inversion, thus minimising the complexity of potential real-time implementations. A systolic array architecture is implemented and FPGA synthesis results indicate a high-throughput low-latency complex matrix inversion solution.

1. Introduction

Matrix inversion is a critical operation in a wide range of signal processing systems, such as multiple-input multiple-output (MIMO) systems for wireless communications (e.g. WiFi, WiMAX). Real-time implementations of emerging MIMO systems demand high-performance and low-complexity matrix inversion operations on both real and complex valued data. These requirements have prompted the design of suitable hardware architectures for real-time complex matrix inversion.

Methods for computing matrix inversion can be divided into two categories: *iterative* and *direct* [2]. Iterative methods require an initial estimate of the solution and subsequent updates based on calculation of the previous estimate error. Normally, these iterative methods involve high-complexity sequential matrix computations and are not particularly suitable for real-time implementation. Direct methods include Gaussian elimination, Cholesky decomposition, LU decomposition and QR decomposition (QRD). These typically compute the solution in a pre-defined finite number of operations and are, therefore, more suitable for time-constrained applications.

QRD is an attractive approach for matrix inversion due to its well known numerical stability [3]. Several algorithms and architectures have been proposed for the computation of QRD-based matrix inversion. Architectures which employ the Gram-Schmidt [5] and conventional Givens rotations (CGR) [6] algorithms are disadvantaged as they require high-complexity square-root operations. Whilst the shift-and-add processing nature of CORDIC-based matrix inversion [7] offers low-complexity hardware implementation, its inherent latency can preclude it from high-performance applications [8]. Squared Givens rotations (SGR) [9] offer square-root free processing and a number of SGR-based matrix inversion architectures have been proposed [10], [11]. However, the SGR algorithm produces erroneous results when zeros occur on the diagonal elements of the matrix either initially or during processing. This limitation is overcome in the modified squared Givens rotations (MSGR) algorithm [1]. Since this algorithm also avoids complex dividers, MSGR offers a low-complexity technique for matrix inversion operations in high-performance systems such as MIMO.

In this paper, a complex matrix inversion architecture based on the novel MSGR algorithm is presented. In Section 2, the MSGR-based matrix inversion technique is described. Derivation of a high-throughput low-latency systolic array architecture for this algorithm is described in Section 3, and novel optimisations of the resulting architecture, including resource sharing and reduction of divider operations, are presented in Section 4. FPGA synthesis results are presented and discussed in Section 5. Conclusions are outlined in Section 6.

2. Matrix Inversion Method

2.1. CGR-Based Matrix Inversion

QRD of a matrix **A** results in (1).

$$\mathbf{A} = \mathbf{QR} \tag{1}$$

Where **R** is an upper triangular matrix and **Q** is an orthogonal matrix (for a complex valued matrix **A**, **Q** is a unitary matrix). The inversion of the matrix **A** is then given as in (2).

$$\mathbf{A}^{-1} = (\mathbf{QR})^{-1} = \mathbf{R}^{-1}\mathbf{Q}^{-1} = \mathbf{R}^{-1}\mathbf{Q}^{H} \tag{2}$$

As **R** is an upper triangular matrix, its inversion can be found by back substitution as described in Section 2.3. Consider the generation of **R** given **A** as in (2). For two partially reduced rows of complex data as in (3), CGR eliminates element a_k using (4)-(6), where a^* is the complex conjugate of a.

978-1-4244-2179-4/08/$25.00 ©2008 IEEE

$$\begin{bmatrix}\mathbf{r}\\ \mathbf{a}\end{bmatrix}=\begin{bmatrix}0,\ldots,0,r_k,\ldots,r_p\\ 0,\ldots,0,a_k,\ldots,a_p\end{bmatrix} \quad (3)$$

$$q=(r_k^* r_k + a_k^* a_k)^{1/2} \quad (4)$$

$$\bar{\mathbf{r}}=q^{-1}(r_k^*\mathbf{r}+a_k^*\mathbf{a}) \quad (5)$$

$$\bar{\mathbf{a}}=q^{-1}(-a_k\mathbf{r}+r_k\mathbf{a}) \quad (6)$$

2.2. MSGR-Based Matrix Inversion

In [9], Dohler proposed SGR to calculate the upper triangular matrix without the need for square-root operations and with reduced multiplications compared to CGR. Using SGR, the matrix **A** is decomposed as in (7)

$$\mathbf{A}=\mathbf{Q}_A\mathbf{D}_U^{-1}\mathbf{U} \quad (7)$$

where, $\mathbf{Q}_A=\mathbf{Q}\mathbf{D}_R$ is not normally an orthogonal matrix, $\mathbf{D}_R=diag(\mathbf{R})$, $\mathbf{D}_U=diag(\mathbf{U})=diag(\mathbf{D}_R\mathbf{R})=\mathbf{D}_R^2$, and $\mathbf{U}=\mathbf{D}_R\mathbf{R}$ is an upper triangular matrix. The inversion of the matrix **A** is then given as in (8).

$$\mathbf{A}^{-1}=(\mathbf{Q}_A\mathbf{D}_U^{-1}\mathbf{U})^{-1}=\mathbf{U}^{-1}(\mathbf{Q}_A\mathbf{D}_U^{-1})^{-1} \quad (8)$$

U is also an upper triangular matrix and its inversion can be found by back substitution. Methods for this and the generation of $(\mathbf{Q}_A\mathbf{D}_U^{-1})^{-1}$ are described in Section 2.3.

The SGR algorithm outlined in [9] suffers from two key drawbacks: it addresses real valued data only and produces erroneous results when zeros occur on the matrix diagonal. To overcome these problems, MSGR is proposed for complex valued matrices with the potential for zeros on the diagonal [1]. MSGR uses (9)-(10) to translate rows **r** and **a** to **U** and **V**-space respectively.

$$\mathbf{u}=r_k^*\mathbf{r} \quad (9) \qquad \mathbf{a}=w^{1/2}\mathbf{v} \quad (10)$$

where, $w>0$ is a scale factor. Therefore, (4)-(6) can be replaced by the MSGR updating as given in (11)-(13), which does not involve square-root operations.

$$\bar{\mathbf{u}}=\mathbf{u}+wv_k^*\mathbf{v} \quad (11)$$

$$\bar{\mathbf{v}}=\mathbf{v}-(v_k/u_k)\mathbf{u} \quad (12)$$

$$\bar{w}=wu_k/\bar{u}_k \quad (13)$$

The condition of zeros on the diagonal (i.e. $u_k=0$) can occur either in the input matrix or during phases of rotations (10)-(12), and if not dealt with, produces erroneous results. To overcome this problem, a novel solution for the condition $u_k=0$, $v_k\neq 0$ (which permits subsequent processing), as well as a solution for $u_k=v_k=0$ (which has not been previously considered), is proposed in the MSGR algorithm, i.e.

$$for\ u_k=0,\ v_k\neq 0:\ \begin{cases}\bar{\mathbf{u}}=wv_k^*\mathbf{v}\\ \bar{\mathbf{v}}=-\mathbf{u}\\ \bar{w}=w\end{cases} \qquad for\ u_k=0,\ v_k=0:\ \begin{cases}\bar{\mathbf{u}}=w\mathbf{v}\\ \bar{\mathbf{v}}=-\mathbf{u}\\ \bar{w}=w\end{cases}$$

2.3. Back Substitution Algorithm

Recalling (7) and (8), MSGR decomposes the input matrix as $\mathbf{A}=\mathbf{Q}_A\mathbf{D}_U^{-1}\mathbf{U}$, which can be inverted as $\mathbf{A}^{-1}=\mathbf{U}^{-1}(\mathbf{Q}_A\mathbf{D}_U^{-1})^{-1}$. Inversion of the upper triangular matrix **U** can be computed using back substitution, as in (14) where $\mathbf{G}=\mathbf{U}^{-1}$.

$$G_{ij}=\begin{cases}-(\sum_{k=1}^{j-1}G_{ik}u_{kj})/u_{jj} & i<j\\ 1/u_{ij} & i=j\\ 0 & i>j\end{cases} \quad (14)$$

Back substitution is performed as part of the invert-and-multiply (IAM) array (Fig. 1), as described in Section 3.

The $(\mathbf{Q}_A\mathbf{D}_U^{-1})^{-1}$ component must also be found for the inversion of **A**. If we rewrite (8) as $\mathbf{U}=(\mathbf{Q}_A\mathbf{D}_U^{-1})^{-1}\mathbf{A}$, then $(\mathbf{Q}_A\mathbf{D}_U^{-1})^{-1}$ can be considered to be the factor by which **A** is multiplied to produce **U**. This multiplication is actually achieved through rotations in the MSGR algorithm. Therefore, this factor can be isolated by multiplying by the identity matrix **I**, which is equivalent to rotating **I** in the same manner as **A**. Therefore, processing **I** in the MSGR array immediately after **A** produces the output $(\mathbf{Q}_A\mathbf{D}_U^{-1})^{-1}$, which is multiplied by the result of the back substitution ($\mathbf{U}^{-1}$) in the IAM array to produce the desired matrix inversion $\mathbf{A}^{-1}=\mathbf{U}^{-1}(\mathbf{Q}_A\mathbf{D}_U^{-1})^{-1}$.

3. Matrix Inversion Architecture

Here, we propose a high-throughput low-latency systolic array architecture for complex matrix inversion. This MSGR-based architecture is derived from the CGR-based architecture proposed in [6], but significantly does not require square-root computations. The systolic array (Fig. 1) comprises two stages: MSGR and IAM arrays.

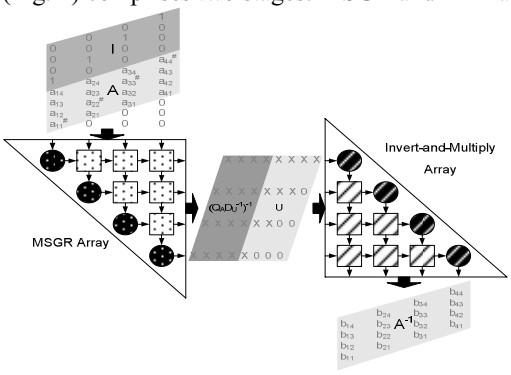
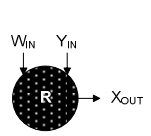


Figure 1. Matrix Inversion Systolic Array

3.1. MSGR Array

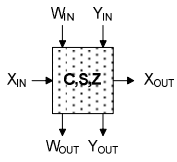
The MSGR array produces an upper triangular matrix **U** given an input matrix **A**. It is also used to produce the

component $(\mathbf{Q}_A\mathbf{D}_U^{-1})^{-1}$ given an input identity matrix **I**. Matrices are input to the array in a skewed manner as shown in Fig. 1. MSGR boundary cells (Fig. 2) are used to translate input vectors to **U**-space. MSGR internal cells (Fig. 3) are used for both the calculation of rotation angles and subsequent rotations. It can be seen that diagonal and off-diagonal elements require different operations in these cells. Therefore, diagonal elements of the input matrix are tagged with a flag as shown in Fig. 1. It can also be seen how the condition of zeros on the diagonal (i.e. u_k=0) is dealt with in these cells according to the MSGR algorithm.




```
if (Y_IN with flag)          //diagonal elements
  X_OUT=Y_IN*×Y_IN×W_IN;
  if (Y_IN==0)               //condition u_k=0
    R=1;
  else                       //condition u_k≠0
    R=Y_IN*;
else                         //off-diagonal elements
  X_OUT=R×Y_IN×W_IN;
  R=R;
```

Figure 2. MSGR Boundary Cell

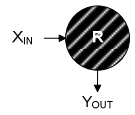



```
if (X_IN with flag)          //diagonal elements
  X_OUT=X_IN+(W_IN×Y_IN*×Y_IN);
  Y_OUT=0;
  if (X_IN==0)               //condition u_k=0
    C=1; Z=0; W_OUT=1;
    if (Y_IN==0)             //condition v_k=0
      S=1;
    else                     //condition v_k≠0
      S=Y_IN*;
  else                       //condition u_k≠0
    C=Y_IN/X_IN; S=Y_IN*; Z=1;
    W_OUT=W_IN×(X_IN/X_OUT);
else                         //off-diagonal elements
  X_OUT=(Z×X_IN)+(W_IN×S×Y_IN);
  Y_OUT=(Z×Y_IN)-(C×X_IN);
  C=C; S=S; Z=Z; W_OUT=W_OUT;
```

Figure 3. MSGR Internal Cell

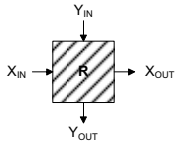
3.2. Invert-and-Multiply Array

The IAM array comprises boundary cells (Fig. 4) and internal cells (Fig. 5) which carry out both the inversion of **U** by back substitution and subsequent multiplication by $(\mathbf{Q}_A\mathbf{D}_U^{-1})^{-1}$ to complete the inversion.




```
if (X_IN with flag)    //diagonal elements
  R=1/X_IN;
  Y_OUT=0;
else                   //off-diagonal elements
  R=R;
  Y_OUT=-R×X_IN;
```

Figure 4. Invert-and-Multiply Boundary Cell




```
if (X_IN with flag)    //diagonal elements
  Y_OUT=0;
  X_OUT=X_IN;
  R=Y_IN/X_IN;
else                   //off-diagonal elements
  Y_OUT=Y_IN-(R×X_IN);
  X_OUT=X_IN;
  R=R;
```

Figure 5. Invert-and-Multiply Internal Cell

4. Architecture Optimisations

If a potential application tolerates longer computation times, the systolic array architecture can be mapped to a linear array of time-shared resources. The application of suitable mapping methodologies is a well researched area [8]. Here, the focus is on exploitation of techniques to minimise the complexity of the array architecture. These include resource sharing and reduced number of divider operations.

4.1. Divider Share in MSGR internal cell

In the MSGR internal cell (Fig. 3), two divides are required for inputted diagonal elements, i.e. $C=Y_{IN}/X_{IN}$ and $W_{OUT}=W_{IN}\times(X_{IN}/X_{OUT})$. As dividers are complicated and expensive hardware resources, it is beneficial to share the same divider to carry out both of these operations. This is achieved by moving the latter divide into the off-diagonal computation mode, i.e. $C=Y_{IN}/X_{IN}$ and $W_{OUT}=W_{IN}\times X_{IN}$ are computed during the diagonal element mode and $W_{OUT}=W_{OUT}/X_{OUT}$ is updated during the off-diagonal mode. It is possible to postpone the update of W_{OUT} since (1) a new W_{OUT} is only passed to an adjacent cell after computation on a diagonal element, and (2) W_{OUT} is not used in the next sample as $Y_{OUT}=0$ for a diagonal element. This gives a suitable timeslot for W_{OUT} to be updated before being used in further computation in the adjacent cell. As a result of this resource sharing, the number of dividers required in the MSGR internal cell is halved with no adverse effect on computation time.

4.2. Divider Share in IAM Array

It is also possible to reduce divider requirements in the IAM array. Both the boundary and internal cells (Fig. 4 and Fig. 5) use a common divisor, i.e. X_{IN}. As this value is constant along each row, the divide operations can be removed from each cell and replaced with a single divide operation at the beginning of the row as shown in Fig. 6. For an $N\times N$ matrix, this modification removes $N(N+1)/2$ dividers from the IAM array and replaces them with just N dividers in the divider array. This modification comes at no extra cost as the resultant multiplication in the internal cell, i.e. $R=Y_{IN}\times(1/X_{IN})$, can be time-shared with the multiplication in off-diagonal mode.

It is worth noting that the MSGR algorithm ensures that no computationally expensive complex divides are required on the diagonal elements during back substitution.

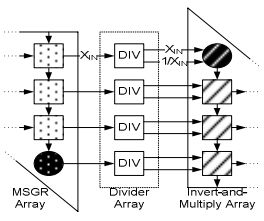


Figure 6. Divider Array

5. Synthesis Results and Discussion

The MSGR-based matrix inversion systolic array has

been synthesised on Xilinx Virtex 4 FPGA technology (xc4vlx200-ff1513). The architecture has been captured and simulated using generic HDL, synthesised using Synplicity Synplify Pro 8.6.2, and mapped, placed and routed using Xilinx ISE 9.1.03i.

In this design study, floating-point arithmetic has been used to allow greater range and precision. Synthesis results are presented for both a reduced-precision floating-point format (exponent bits, E=6, mantissa bits, M=14) commonly used in QRD-based RADAR implementations [4] and IEEE single-precision format (E=8, M=24). Floating-point operators (v3.0) were generated using Xilinx CORE Generator 9.1.03i. The overall architecture can be readily mapped to other technologies by using non-target-specific fixed/floating-point cores.

Real-time MIMO systems have a limited period to decode and recover transmitted data. Therefore, high-throughput and low-latency operations such as decoding, matrix inversion and multiplication are crucial. The systolic array architecture presented in this paper offers highly parallel matrix inversion operations with throughput $f/2n_r$ and latency $(3K+2)n_r$ cycles, where f is clock frequency and K is latency (pipeline depth) of each cell.

Table 1. FPGA synthesis comparison

Architecture	Throughput (MInversions/s)	Latency		Occupied Slices	DSP block
		µs	cycle		
MSGR	1.6*	1.5	20	34,674	0
[10]	0.13	8.1	933	9,117	22

*Worst-case throughput for MSGR-based matrix inversion architecture for minimal pipelining K=1. Throughput rates up to 17.5 MInversions/s possible (K=16).

Alternatively, cells can be further pipelined (increasing K) to increase throughput up to 17.5 MInversions/s (f=157 MHz) but at the cost of increased resource usage, as illustrated in Fig. 7.

Other implementations are not compared here due to the fairness problem. In [11] a systolic array is implemented which consumes 8321 slices and 92 DSP blocks, but they only solve the matrix QRD part not the whole matrix inversion algorithm. In [12], the matrix inversion is implemented by application specific processor and the synthesize result exclude the memory used in the design and the algorithm they used is only for small size matrix.

6. Conclusions

A complex matrix inversion architecture based on a novel MSGR algorithm is presented, which extends the original SGR method for complex valued data, and also corrects erroneous results in the original SGR method when zeros occur on the diagonal of the matrix either initially or during processing. The MSGR algorithm also avoids complex dividers in the matrix inversion, thus minimising the complexity of potential real-time implementations. A systolic array architecture is presented with generic modular processing cells optimised for reduced complexity. FPGA synthesis results indicate a high-throughput (1.6-19.6 MInversions/s) low-latency (1.5 µs) complex matrix inversion solution.

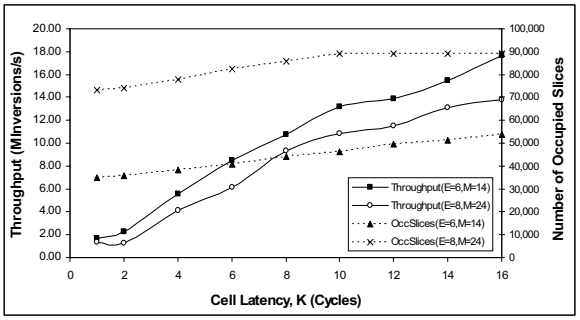


Figure 7. Throughput and Occupied Slices versus Cell Latency

7. References

[1] L. Ma, K. Dickson, J. McAllister and J. McCanny, "Modified Givens rotations and their application to matrix inversion", in *ICASSP,* 08, PP. 1437-1440, March 2008.

[2] M. Ylinen, A. Burian and J. Takala, "Updating matrix inverse in fixed-point representation: Direct versus iterative methods," in *International Symposium on System-on-Chip,* pp. 45-8, 2003.

[3] S. Haykin, *Adaptive Filter Theory.*, 2nd ed.Prentice Hall, 1991,

[4] G. Lightbody, R. Walke, R. Woods and J. McCanny, "Linear QR architecture for a single chip adaptive beamformer," *Journal of VLSI Signal Processing Systems,* vol. 24, pp. 67-81, 2000.

[5] C. K. Singh, Sushma Honnavara Prasad and P. T. Balsara, "VLSI architecture for matrix inversion using modified gram-schmidt based QR decomposition," in *20th International Conference on VLSI Design,* pp. 836-841, 2007.

[6] A. El-Amawy and K. R. Dharmarajan, "Parallel VLSI algorithm for stable inversion of dense matrices," *IEE Proceedings, Part E: Computers and Digital Techniques,* vol. 136, pp. 575-580, 1989.

[7] A. Maltsev, V. Pestretsov, R. Maslennikov and A. Khoryaev, "Triangular systolic array with reduced latency for QR-decomposition of complex matrices," in *ISCAS 06,* pp.21-24, May 2006.

[8] R. L. Walke, R. W. M. Smith and G. Lightbody, "Architectures for adaptive weight calculation on ASIC and FPGA," *Conference Record of the Asilomar Conference on Signals, Systems and Computers,* vol. 2, pp. 1375-1380, 1999.

[9] R. Dohler, "Squared Givens rotation," *IMA Journal of Numerical Analysis,* vol. 11, pp. 1-5, 01. 1991.

[10] M. Karkooti, J. R. Cavallaro and C. Dick, "FPGA implementation of matrix inversion using QRD-RLS algorithm," in *39th Asilomar Conference on Signals, Systems and Computers,* 2005, pp. 1625-1629.

[11] Barbara Cerato, Guido Masera and Peter Nilsson, "Hardware architecture for matrix factorization in MIMO receivers," in *GLSVLSI'07,* pp. 196-199, March 2007.

[12] Johan Eilert, Di Wu and Dake Liu "Efficient Complex Matrix Inversion for MIMO Software Define Radio". ISCAS 07, PP 2610-2613, May 2007.