UC San Diego
Technical Reports

Title
An Efficient FPGA Implementation of Scalable Matrix Inversion Core using QR Decomposition

Permalink
https://escholarship.org/uc/item/6mj094vk

Authors
Irturk, Ali
Mirzaei, Shahnam
Kastner, Ryan

Publication Date
2009-03-09

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

An Efficient FPGA Implementation of Scalable Matrix Inversion Core using QR Decomposition

Ali Irturk†, Shahnam Mirzaei‡, Ryan Kastner†

†Department of Computer Science and Engineering
University of California, San Diego

‡Department of Electrical and Computer Engineering
University of California, Santa Barbara

Abstract. We present a novel scalable architecture for matrix inversion that uses the modified Gram-Schmidt algorithm based on QR decomposition. Our core achieves a throughput of $0.18M$ updates per second for a 4 x 4 matrix using 19 bits of precision on a Xilinx Virtex4 SX FPGA. We also present two different designs which use longer data lines, 26 and 32 bits, and compare our results with another matrix inversion architecture which is the only scalable approach so far. We show that our core is significantly faster than the other published FPGA implementation as it requires fewer resources due to the usage of fixed point arithmetic and an effective resource utilization. We show that our proposed architecture is scalable by presenting the results for 6 x 6 and 8 x 8 matrices.

1 Introduction

Orthogonal Frequency Division Multiplexing (OFDM) is one of the most promising technologies for high data rate wireless communications due to its robustness to frequency selective fading, high spectral efficiency, and low computational complexity. Multiple Input Multiple Output (MIMO) systems, which improve the capacity and performance of wireless communication by using multiple transmit and receive antennas, are often used in conjunction with OFDM to improve the channel capacity and reduce intersymbol interference (ISI) [1]. MIMO-OFDM systems require equalization at the receiver side to remove the effect of channel on the signal. Matrix inversion is an essential computation for equalization where the size of the matrix depends on the number of transmitter and receiver antennas. Higher data rates can be achieved by using more antennas at the both sides; however this creates more computation, e.g. a bigger matrix must be inverted during equalization.

Matrix inversion with analytic approach using determinants results in non-scalable architectures, thus the usage of decomposition methods need to be introduced for the inversion of bigger matrices. QR decomposition is the traditionally used decomposition method for matrix inversion because of its stability. Furthermore, Cholesky and LU decompositions are generally used for positive definite and non-singular square matrices while QR decomposition can be used with any kind of matrices. We present a scalable architecture for matrix inversion using QR decomposition in this paper.

There are three different QR decomposition methods: Gram-Schmidt orthogonormalization, Givens Rotations (GR) and Householder reflections [2]. GR has been the focus of previously published hardware implementations due to its stability and accuracy, and the fact that it easily lends itself to a systolic array

architecture using CORDIC blocks. However, it was shown that the modified Gram-Schmidt (MGS) method is numerically equivalent to GR method [3]. We choose to use QRD-MGS method for our implementation to provide a comparison with GR method in terms of area and throughput. We achieve better results than previous GR implementations because of the fixed point arithmetic and effective resource utilization.

The choice of a computing platform plays an important role in the overall design of a communication systems. Previously, designers decided between a hardware or software implementation. The hardware implementation consisted of designing an ASIC, which offers exceptional performance, but long time to market and high costs for all but the largest production chips. The software route tended towards the use of DSPs due to the ease of development and fast time to market. However, they lack the performance for high throughput applications. Recently FPGAs have become prevalent for signal processing applications. FPGAs play a middle role between ASICs and DSPs, as they have the programmability of software with performance approaching that of a custom hardware implementation.

We select Xilinx Virtex 4 SX as our implementation platform. Virtex 4 SX provides a rich set of resources, including Block RAMs for on chip memory and FIFOs, DSP blocks for mathematical operations such as Multiply Accumulates (MACs), division, square root, etc., and clocking management features for frequency synthesis. If these resources are utilized correctly, these features can significantly enhance the performance, area and throughput.

In this paper we present a scalable matrix inversion core which uses QRD-MGS with fixed-point arithmetic and map it onto Xilinx Virtex 4SX FPGA. We explore practical hardware design and implementation issues for FPGAs. Our core results in a high performance scalable architecture for matrix inversion. We achieve 3.8 times smaller design in terms of slices with 1.4 times higher throughput for 4 x 4 matrix inversion compared to the GR implementation [13].

The rest of this paper is organized as follows: Section 2 presents related work. Section 3 describes the QRD-MGS algorithm for matrix inversion. Section 4 addresses the advantages of fixed-point arithmetic and subsequently presents error analysis for different amounts of precision. Section 5 explains the architectural design of the core. Section 6 introduces FPGA resources, discusses design decision and challenges, and presents implementation results in terms of area and timing. We conclude in Section 7.

2 Related Work

Several approaches can be used for matrix inversion. These include Cholesky [4], LU [5] and Gauss Jordan [6] [7]. There are some VLSI architectures for matrix inversion using QR decomposition which do not specifically target FPGAs. Dharmarajan et al. [8] present an algorithm to perform matrix inversion of dense square matrices. They use GR method with systolic array VLSI architecture. Singh et al. [9] present a fully parallel architecture for matrix inversion

using QRD-MGS algorithm. Most previous work using FPGA for matrix inversion is concentrated on smaller matrix sizes and use architectures that will not easily extendable to larger matrix sizes. Liu et al. [10] presented a new method by partitioning the matrix into smaller matrices. However, they didn't consider scalability of their architecture; they found a practical solution for matrices which are not larger than 4 x 4. In [11], they extended their work to bigger matrices, 16 x 16, and the precision analysis for this dimension is presented. However FPGA implementation results are reported for 4 x 4 matrices only.

Edman et al. [12] presents a scalable linear array architecture for inverting complex valued matrices. They use squared Givens rotations algorithm for their implementation with 12 bit fixed-point representation. Their architecture for inversion of a 4 x 4 matrix consumes 86% of the Virtex-IIs capacity and requires 175 cycles to invert a matrix. Karkooti et al. [13] presents an architecture for matrix inversion using QR decomposition based recursive least square (RLS) algorithm. They use squared Givens rotations as their algorithm and implement their design as a folded systolic array with 20 bits floating point representation.

The focus of our work is to design a scalable architecture with a small area requirement for matrix sizes suitable for use in communication applications. Therefore, we use QR decomposition approach instead of an analytical approach where complexity increases very quickly with the size of the matrix. We introduce an effective resource utilization methodology that automatically analyzes dependencies between resources and eliminates unneeded routing circuitry. This is extremely effective in reducing area and enables a scalable architecture. This architecture can be used in two different areas. Firstly, because the area is already small for a 4 x 4 matrix inversion core and the architecture is scalable to the bigger matrix sizes, it is possible to use the rest of the FPGA resources for bigger matrices inversion. Secondly, it is possible to reuse the same FPGA device for other wireless ommunication operations such as FFT/IFFT, matrix multiplication and adaptive weight calculation.

3 Matrix Inversion using QR Decomposition

Matrix Inversion has many applications, like equalization/detection algorithms in MIMO-OFDM systems. The inverse of a square matrix A is a matrix which is shown as A^{-1} and satisfies:

$$A \times A^{-1} = I \tag{1}$$

where I is the identity matrix and it's matrix form can be seen as:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

However, matrix inversion becomes a bottleneck because of its complexity, especially for bigger matrix sizes. Applying QR decomposition decreases this complexity using its properties.

QR decomposition is an elementary operation, which decomposes a matrix into an orthogonal and a triangular matrix. QR decomposition of a matrix A is a decomposition of A as $A = QR$, where Q is an orthogonal matrix ($Q^T \times Q = I$) and R is an upper triangular matrix. The decomposition of $m \times n$ matrices (with $m \geq n$) of full rank is the product of an $m \times n$ orthogonal matrix where $Q^T \times Q = I$ and an $n \times n$ upper triangular matrix. The QR decomposition of a given 4 x 4 matrix A can be seen as follows:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12} & Q_{13} & Q_{14} \\ Q_{21} & Q_{22} & Q_{23} & Q_{24} \\ Q_{31} & Q_{32} & Q_{33} & Q_{34} \\ Q_{41} & Q_{42} & Q_{43} & Q_{44} \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ 0 & R_{22} & R_{23} & R_{24} \\ 0 & 0 & R_{33} & R_{34} \\ 0 & 0 & 0 & R_{44} \end{bmatrix}$$

The solution for the inversion of matrix A, A^{-1}, using QR decomposition can be seen as follows:

$$A \times A^{-1} = I \tag{2}$$

$$Q \times R \times A^{-1} = I \tag{3}$$

$$A^{-1} = \frac{I}{R \times Q} \tag{4}$$

$$A^{-1} = R^{-1} \times Q^T \tag{5}$$

The matrix representation of (5) is shown as:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}^{-1} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ 0 & R_{22} & R_{23} & R_{24} \\ 0 & 0 & R_{33} & R_{34} \\ 0 & 0 & 0 & R_{44} \end{bmatrix}^{-1} \begin{bmatrix} Q_{11} & Q_{21} & Q_{31} & Q_{41} \\ Q_{12} & Q_{22} & Q_{32} & Q_{42} \\ Q_{13} & Q_{23} & Q_{33} & Q_{43} \\ Q_{14} & Q_{24} & Q_{34} & Q_{44} \end{bmatrix}$$

This solution consists of three different parts, QR decomposition, matrix inversion for upper triangular matrix and matrix multiplication. Most number of calculations are done while decomposition of the matrix A. The inverse of R matrix, R^{-1}, is a less complex matrix inversion because of the upper triangular matrix structure of R. And it is calculated with simple back-substitution using identity matrix which can be seen as:

$$R \times R^{-1} = I \tag{6}$$

The transpose matrix, Q^T, requires simple register renaming due to usage of scheduling and does not require any calculation. The multiplication of the inverse upper triangular matrix, R^{-1}, and transpose of Q, Q^T, gives the result

A^{-1}. The required calculations for Matrix Inversion are shown in Figure 1.

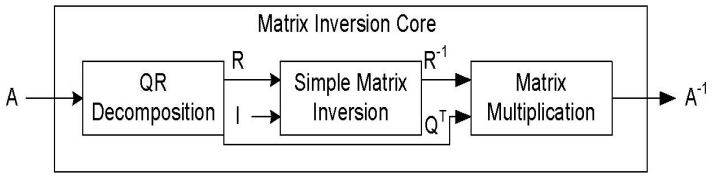

Fig. 1. Matrix Inversion using QR decomposition consists of there different parts to calculate the inverse of A, QR decomposition, the simple inversion of R and matrix multiplication.

3.1 QR decomposition using Modified Gram-Schmidt Algorithm and Matrix Inversion

Applying slight modifications to the Classical Gram-Schmidt (CGS) algorithm gives the modified Gram-Schmidt (MGS) algorithm. QRD-MGS is numerically more accurate and stable than the QRD-CGS. And it is numerically equivalent to the Givens Rotations solution. If the given matrix, A, is well-conditioned, the resulting matrices satisfy their required matrix characteristics. The algorithm for QRD-MGS is shown in Figure 2.

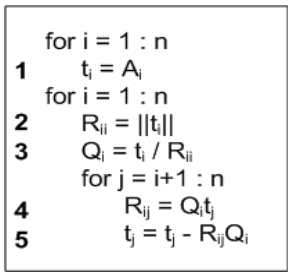

Fig. 2. QRD-MGS Algorithm

We simply start every decomposition by storing the given matrix, A, into the Q matrix entries in the register file. (1) shows this transfer and A matrix is denoted as t matrix which is temporary while creating Q matrix. Every diagonal entry in the R matrix is the euclidean norm of the t matrix columns which is

shown as (2). The Q matrix is created by the division of t with the euclidean norm (3) column by column. Non-diagonal entries of R matrix is computed by multiplication of the real Q matrix column and temporary t matrix columns one by one (4). For example, we first compute the first column of Q matrix and then project this column to the next t matrix columns. Every multiplication gives us the other R matrix entry of the first row. Then, t matrix columns are updated by (5).

Matrix Inversion of an upper triangular matrix requires less calculations compared to full matrix inversion because of its zero entries. The matrix representation is shown as:

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ 0 & R_{22} & R_{23} & R_{24} \\ 0 & 0 & R_{33} & R_{34} \\ 0 & 0 & 0 & R_{44} \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ 0 & R_{22} & R_{23} & R_{24} \\ 0 & 0 & R_{33} & R_{34} \\ 0 & 0 & 0 & R_{44} \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The algorithm for matrix inversion is shown in Figure 3.

```
   for j = 1 : n
      for i = 1 : (j-1)
         for k = 1 : (j-1)
1           R^-1(i,j) = R^-1(i,j) + R^-1(i,k) x R(k,j)
      for k = 1 : (j-1)
2        R^-1(k,j) = -R^-1(k,j) / R(j,j)
3  R^-1(j,j) = 1 / R(j,j)
```

Fig. 3. Matrix Inversion Algorithm for Upper Triangular Matrix, R.

Upper triangular matrix inversion is performed column by column. Calculating the diagonal entries of the R^{-1} matrix is simply dividing 1 by the diagonal entry of R matrix (3) and the rest of the column entries introduces multiplication and addition iteratively (1) which is then divided by R matrix entry (3).

4 Fixed Point Arithmetic and Error Analysis

Fixed point arithmetic is important as it results in faster and smaller functional units. However, it can result in less accurate results if it is not carefully designed. In this section, we discuss these tradeoffs and formulate an appropriate fixed point representation that provides results that are similar to a floating point implementation, yet they are much faster and smaller.

4.1 Fixed Point Arithmetic

Usage of floating point arithmetic is expensive in terms for hardware and leads to inefficient designs especially for FPGA implementation. On the other hand, fixed point arithmetic results in efficient hardware designs. Our design uses two's complement fixed point arithmetic, which is shown in Figure 4. The data lines used in implementation for fixed point arithmetic consist of an integer part, a fractional part and a sign bit.

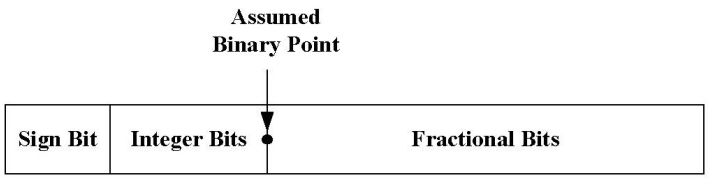

Fig. 4. Two's complement fixed-point representation is used in the calculations of the matrix inversion core. The limited nature of the representation leads round-off and truncation errors. These errors must be investigated carefully for the precision analysis.

Matrix inversion using QR-MGS algorithm requires the usage of addition, subtraction, multiplication, division and square root operations. Number of calculations increases as the matrix dimensions increases. Some of these calculations are straightforward such as addition, subtraction and multiplication; however, division and square root operations are complex and can affect the precision significantly. Furthermore, they are inefficient in terms of FPGA implementation. Fixed-point arithmetic reduces precision and consequently introduces two types of errors: round-off and truncation errors. Round-off error occurs when the result requires more bits than the reserved bit length after a computation. Truncation error occurs due to the limited number of bits to represent numbers. These issues must be handled carefully to prevent overflow which leads to incorrect results. Error analysis is a crucial step and will be considered in the next section.

4.2 Error Analysis

Error analysis is performed for whole matrix inversion core to determine the tradeoffs between precision and area. We investigate the usage of 16 to 64 number of bits to perform matrix inversion. One of the major problems while working with fixed-point arithmetic is preventing overflows. While performing arithmetic calculations, a result may not fit into the reserved bits and if this case is not handled carefully, it causes overflow and incorrect results. To prevent overflow, every entry in the given matrix is normalized as the first step before starting decomposition. Normalizing is performed by dividing every entry by the largest

matrix entry. Therefore, the given matrix entries are always in the [0, 1]. Normalization allows us to calculate the maximum number of bits required for the biggest possible integer assuming that given matrix is well-conditioned. We then reserve this number of bits to ensure that overflow does not occur.

The error analysis is performed by comparing the fixed point arithmetic results with the double precision floating point artihmetic results using Matlab software. We also compare our simulation results which are derived using Modelsim software by simulating our proposed architecture with the actual results. The error analysis is done for 4 x 4 matrices with 16 to 64 bits. We considered mean error for our error analysis which is computed by finding the error for all entries and then dividing the sum of these errors by the total number of entries. This calculation can be seen as:

$$\frac{\sum_{i=1}^{m} |y_i - \hat{y}_i|}{m} \tag{7}$$

We generate 100 random matrices for our error analysis which have entries in the [0, 1] and matrix inversion is performed. The mean error for different number of bits is shown in Figure 5 with log domain scale.

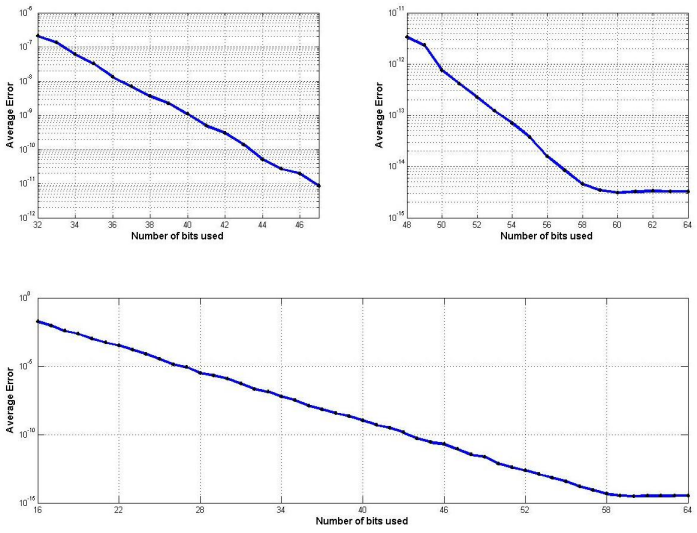

Fig. 5. The mean error is an important metric for error analysis. For better precision, more bits must be used in calculations which introduce the tradeoff between the precision and area of the design. We perform error analysis by comparing the fixed point arithmetic results with the double precision floating point artihmetic results using Matlab software. We also compare our simulation results which are derived using Modelsim software by simulating our proposed architecture with the actual results.

5 Architectural Design of Matrix Inversion Core

The proposed architecture works at the instruction-level where the instructions define the required calculations for the matrix inversion. For better performance results, instruction level parallelism is exploited. The dependencies between the instructions limit the amount of parallelism that exists within a group of computations. Dynamic Scheduling using Tomasulo's approach is a widely known approach to produce instruction level parallelism in presence of dependences [14]. In this approach, controller units track the operands to determine whether they are available and perform register renaming which assigns a free arithmetic unit for the desired calculation. Register renaming is provided by reservation station usage in every arithmetic unit where reservation stations fetch and buffer an operand as soon as the operand is ready. Our proposed design consists of two controller units and three arithmetic units. The arithmetic units are capable of computing decomposition, simple matrix inversion using back-substitution and matrix multiplication. The control units are instruction and timing and operand controller. The arithmetic units are adder/subtractor, multiplier/divider and square root units. The proposed matrix inversion core is shown in Figure 6.

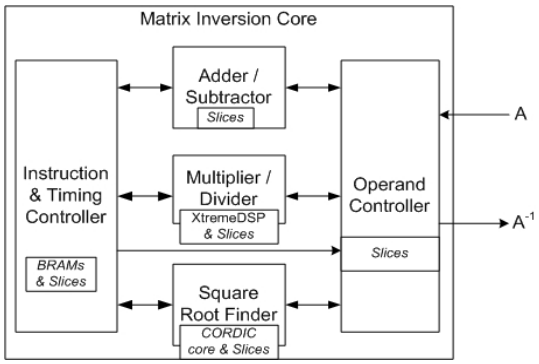

Fig. 6. The proposed Matrix Inversion core is capable of doing QRD, simple matrix inversion and matrix multiplication. Matrix Inversion core consists of 2 controller and 3 arithmetic units.

The Instruction and Timing controller unit keeps the instructions which are required to perform matrix inversion. The control elements supervise the status of the reservation stations of the arithmetic units and the current place of the operands. The Instruction and Timing controller function is as follows: 1) it gets the instructions from the memory element by ensuring the maintenance of correct data flow, 2) decodes the instruction to understand the desired calculation, the required operands and the destination memory entry in operand controller unit, 3) creates scheduled instructions by performing register renaming, 4) checks the current status of the arithmetic units and the current place

of the operands and sends the scheduled instruction to the other units of the design or stall the core if there is no free reservation station. Every calculation starts by storing the given matrix data, A into the memory entries of the Operand Controller. The Operand Controller sends the required operands for calculations if the operand is ready. Otherwise it waits until the calculation of the operand is completed and then sends the data after updating the memory. Arithmetic units are reserved for specific calculations. The arithmetic units are adder/subtractor, multiplier/divider and square root finder. Each unit, except the square root unit, consists of matrix size number of reservation stations to perform desired calculations concurrently. Every arithmetic unit has three different stages: fetching the instruction from Instruction and Timing Controller, fetching the required operands from the outputs of the arithmetic units or from the memory unit of Operand Controller unit and performing the calculation. Addition, subtraction and multiplication use one clock cycle. Division and square root operations use N clock cycles, where N is equal to the number of bit in the input data. If the input matrix is 4×4, our core uses 4 reservation stations in each adder/subtractor and multiplier/divider units due to the fact that they are able to perform most of these calculations in parallel. The square root function unit uses only one reservation station because it is rarely needed during the QR decomposition. We choose maximum parallelism for QR decomposition and the calculations for matrix inversion and multiplication use the same resources with different scheduling.

6 FPGA Implementation Results

The proposed architecture is implemented on a Virtex4-SX FPGA. We use 19, 26 and 32 bits for the data representations and compare their results. The addition and subtraction functional units are implemented by using SLICEs with low delay carry chain network, the multiplications use XtremeDSP blocks which are very efficient for this purpose and finally square root function uses CORDIC core, all provided by Xilinx Coregen toolset. We use Block RAMs available on Xilinx FPGAs as memory storage space for instructions. The Block RAM modules provide flexible 18Kbit dual-port RAM, that are cascadable to form larger memory blocks. Embedded XtremeDSP slices with 18 x 18 bit dedicated multipliers and 48-bit accumulator provide flexible resources to implement multipliers to achieve high performance. Furthermore, Xilinx Coregen toolset implements these cores very efficiently since it uses special mapping and place and route algorithms to implement the above mentioned cores to attain high performance design.

Matrix inversion core which uses 19 bits achieves 116 MHz of speed on the FPGA with a throughput of $0.18M$ updates per second. The latency for matrix inversion is 663 cycles. Matrix inversion core which uses 26 bits achieves 113 MHz of speed on the same FPGA with a throughput of $0.14M$ updates per second. The latency for matrix inversion is 803 cycles. And matrix inversion core which uses 32 bits achieves 96.2 MHz of speed on the same FPGA with a throughput of

$0.11M$ updates per second. The latency for matrix inversion is 923 cycles. The resources used for these designs are shown in Table 1.

Table.1. Area results for 4×4 matrices with 19, 26 and 32 bits of data lines.

19 bits	$LUTs$	FFs	$BRAMs$	$DSP48$	$SLICEs$
Instruction and Timing Controller	172	46	1	0	92
Operand Controller	1,338	268	0	0	866
Arithmetic Units	2,018	1,417	0	12	1,457
Total	3,528	1,731	1	12	2,415
26 bits					
Instruction and Timing Controller	172	46	1	0	92
Operand Controller	4,488	1,136	0	0	2,505
Arithmetic Units	2,826	2,094	0	12	2,059
Total	7,486	3,276	1	12	4,656
32 bits					
Instruction and Timing Controller	172	46	1	0	92
Operand Controller	5,041	1,408	0	0	2,910
Arithmetic Units	3,591	2,754	1	12	2,638
Total	8,804	4,208	1	12	5,640

The comparison between [13] and our architecture can be seen in the Table 2.

Table.2. Area and throughput comparisons with [12]

	$Wordlength(bit)$	$LUTs$	FFs	$BRAMs$	$DSP48$	$SLICEs$	$Throughput$
[13]	20	NR	NR	9	22	9,117	$0.13M$
Our	19	3,528	1,731	1	12	2,415	$0.18M$
Our	26	7,486	3,276	1	12	4,656	$0.14M$
Our	32	8,804	4,208	1	12	5,640	$0.11M$

Due to the usage of scheduling, our design is easily extendable for larger matrix sizes. This can be done by changing the instructions for scheduling which are required for matrix inversion and using more memory elements and control elements. For best timing results n number of reservation stations in arithmetic units are added in the core which is 4 in our design and works for 4×4 matrices. It is important to state that more area efficient designs can be implemented using less reservation stations with the cost of more clock cycles to perform inversion. We also compare our results of 4×4 with 6×6 and 8×8 which use 19 bits of data lines in Table 3.

Table.3. Area and throughput comparisons for bigger matrix sizes.

	$SLICEs$	$BRAMs$	$DSP48$	$Throughput$
4×4	2,415	1	12	$0.18M$
6×6	7,820	2	18	$0.07M$
8×8	11,761	4	24	$0.03M$

7 Conclusion

A matrix inversion core is designed and implemented on Xilinx4-SX FPGA using QRD Modified Gram-Schmidt algorithm with fixed-point arithmetic. The error analysis for matrix inversion is investigated for different data lengths. The proposed design for 4×4 matrix inversion runs with a clock rate of 116 MHz and achieves a throughput of $0.18M$ updates per second using 19 bits of data length. The proposed design has better results than the previous work in terms of area and throughput. We also present the results for 26 and 32 bits of data lengths. Our design is easily extendable to other matrix sizes and the results for 6×6 and 8×8 matrices are also presented.

References

1. Hanzo, L., Keller, T.: OFDM and MC-CDMA: A Primer. Wiley-IEEE Press (2006)
2. Golub, G.H., Van Loan, C. F.: Matrix Computations. 3 rd ed. John Hopkins University Press, Baltimore, MD, (1996)
3. Bjorck, A: Numerics of Gram-Schmidt Orthogonalization. Linear Algebra and Its Applications (1994) 198:297–316
4. Burian, A., Takala, J., Ylinen, M.: A Fixed-Point Implementation of Matrix Inversion Using Cholesky Decomposition. Proceedings of the 46th IEEE International Midwest Symposium on Circuits and Systems. (2003) 1431 - 1434
5. Happonen, A., Piirainen, O., Burian, A.: GSM Channel Estimator Using a Fixed-Point Matrix Inversion Algorithm. International Symposium on Signals, Circuits and Systems. (2005) 119 - 122
6. Salmela, P., Happonen, A., Burian, A., Takala, J.: Several Approaches to Fixed-Point Implementation of Matrix Inversion. International Symposium on Signals, Circuits and Systems. (2005) 497 - 500
7. Matos, G., Neto, H.C.: On Reconfigurable Architectures for Efficient Matrix Inversion. FPL. (2006) 1 - 6
8. El-Amawy, A., Dharmarajan, K.R.: Parallel VLSI Algorithm for Stable Inversion of Dense Matrices. IEE Proceedings-Computers and Digital Techniques, Volume 136, Issue 6, (1989) 575 - 580
9. Singh, C.K., Prasad S.H., Balsara P.T.: VLSI Architecture for Matrix Inversion using Modified Gram-Schmidt based QR Decomposition. 20th International Conference on VLSI Design. (2007) 836 - 841
10. Eilert, J., Wu, D., Liu, D.: Efficient Complex Matrix Inversion for MIMO Software Defined Radio. IEEE International Symposium on Circuits and Systems. (2007) 2610 - 2613
11. Wu, D., Eilert, J., Liu, D., Wang, D., Al-Dhahir, N., Minn, H.: Fast Complex Valued Matrix Inversion for Multi-User STBC-MIMO Decoding. IEEE Computer Society Annual Symposium on VLSI. (2007) 325 - 330
12. Echman, F.; Owall, V.: A Scalable Pipelined Complex Valued Matrix Inversion Architecture. ISCAS. (2005) 4489 - 4492
13. Karkooti, M.; Cavallaro, J.R.; Dick, C.: FPGA Implementation of Matrix Inversion Using QRD-RLS Algorithm. Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers (2005) 1625 - 1629
14. Hennessy, J.L., Patterson D.A.: Computer Architecture: A Quantitative Approach. Morgan Kaufman Publishers, Third Edition. (2003)