FPGA Implementation of Fast QR Decomposition Based on Givens Rotation

Semih Aslan[1], Sufeng Niu [2], Jafar Saniie[2]

[1] Department of Electrical and Computer Engineering
Texas State University
San Marcos, Texas, 78666, USA

[2] Department of Electrical and Computer Engineering
Illinois Institute of Technology
Chicago, Illinois, 60616, USA

Abstract — **In this paper, an improved fixed-point hardware design of QR decomposition, specifically optimized for Xilinx FPGAs is introduced. A Givens Rotation algorithm is implemented by using a folded systolic array and the CORDIC algorithm, making this very suitable for high-speed FPGAs or ASIC designs. We improve the internal cell structure so that the system can run at 246MHz with nearly 24M updates per second throughout on a Virtex5 FPGA. The matrix size can be easily scaled up.**

Keywords: FPGA, QR decomposition, givens rotation, systolic

I. INTRODUCTION

In this paper, we developed an architecture for QR decomposition [1] using the Givens Rotation algorithm [2][3]. The proposed design, based on CORDIC (Coordinate Rotation Digital Computer) algorithm [4][5] and fixed-point calculations, is optimized for FPGA platforms and the system can run at maximum speed at 246MHz.

The organization of the paper is as follows: The section II provides a background discussion on QR decomposition and the brief mathematic procedure. The section III presents how to map the Givens Rotation to FPGA architecture. In addition, the traditional cell structure is compared with the proposed design and its optimization. In the section IV, the results and error margins are discussed.

II. QR DECOMPOSITION AND GIVENS ROTATION

A. QR Factorization and background

QR decomposition is one of the most significant operations in linear algebra. It decomposes a matrix into an orthogonal and a triangular matrix. It is one of the core operations for Multiple Input Multiple Output (MIMO) Orthogonal Frequency Division Multiplexing (OFDM). It is commonly used to find the matrix inversion and numerous applications in scientific computing like Multi-User Detection, Channel Estimation, the Spatial Filtering Problem and Adaptive Beamforming [6]. For example, a phased array system and adaptive beamforming can overcome many of the interference problems encountered [7]. The method for fast and numerically robust adaptive beamforming is the QR Decomposition Recursive Least Squares (QRD-RLS) algorithm, which has faster convergence than LMS and is numerically stable.

Herdeg, et al. [8], used and evaluated the Givens Rotation on GPU. Egecioglu [9] also compared the Householder transformation on workstations. However, a general purpose CPU including a DSP processor is not capable of handling real time signal processing situations, while Gentleman & Kung [10] first proposed a parallel and pipelined triangular array.

There are different methods which can be used to compute QR decomposition including Householder Reflections, Givens Rotations, Gram-Schmidt Orthonormalization, and so on. Each algorithm has its pros and cons. Due to its properties and mathematic procedure, the Givens Rotation algorithm for VLSI is easy to implement with pipelining and ASIC fabrication. In the following sections, we introduce the Givens Rotation and its high-speed implementation.

B. Givens Rotation Algorithm

Given a matrix A:

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

where R is an upper triangle matrix, Q is orthogonal and satisfies:

$$Q^T Q = I \tag{2}$$

Givens Rotation eliminates one element in a matrix one at a time. After a series of Givens Rotations are applied to zero required elements, it produces the upper triangular matrix R and Q, while Q is composed of a series of rotation components.

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ 0 & r_{22} & r_{23} & r_{24} \\ 0 & 0 & r_{33} & r_{34} \\ 0 & 0 & 0 & r_{44} \end{pmatrix} = Q_6Q_5Q_4Q_3Q_2Q_1 \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = Q^T A \tag{3}$$

Here, an example of a 4 x 4 matrix Givens Rotation step is given:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \quad (4)$$

First, we use a_{11} to eliminate a_{21}. For the first two rows, we generate the Givens rotation matrix:

$$Q_1 = \begin{pmatrix} c_1 & s_1 & 0 & 0 \\ -s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (5)$$

where:

$$c_1 = \frac{a_{11}}{\sqrt{a_{11}^2 + r_i^2}} \qquad s_1 = \frac{r_i}{\sqrt{a_{11}^2 + r_i^2}} \quad (6)$$

and $(r_i = a_{2i}),\ i = 1,2,3,4$. By multiplying the original matrix:

$$\begin{pmatrix} c_1 & s_1 & 0 & 0 \\ -s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \quad (7)$$

The first two rows of the calculation procedure are shown below:

$$\begin{pmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ r_1 & r_2 & r_3 & r_4 \end{pmatrix} \quad (8)$$

$$= \begin{pmatrix} c_1 a_{11} + s_1 r_1 & c_1 a_{12} + s_1 r_2 & c_1 a_{13} + s_1 r_3 & c_1 a_{14} + s_1 r_4 \\ -s_1 a_{11} + c_1 r_1 & -s_1 a_{12} + c_1 r_2 & -s_1 a_{13} + c_1 r_3 & -s_1 a_{14} + c_1 r_4 \end{pmatrix} \quad (9)$$

$$= \begin{pmatrix} \sqrt{a_{11}^2 + r_1^2} & X & X & X \\ 0 & X & X & X \end{pmatrix}, X \ any\, number \quad (10)$$

a_{21} has been zeroed. By following the same procedure, we can obtain the upper triangle matrix. In the next section, we will see how to map the algorithm to FPGA and its improvements.

III. FPGA IMPLEMENTATION

A. Previous work and systolic architecture

From Section II, it can be clearly seen that for each two rows, the first column is calculated for cosine (c) and sine (s) pair, whereas the others are for rotation. Based on this rule, the systolic architecture can be mapped as Figure 1 as 4 by 4 matrix. Each row has a boundary cell to generate cosine and sine value to feed internal cells. Each circuit works in a pipelined fashion, one cell at a time.

Inside the boundary cell, Xilinx Reference design [11] gives a solution as shown in Figure 2.a, while a double CORDIC structure is presented in [5][4] which divides into a master and a slave to generate cosine and sine in Figure2.b. Lattice [12] also provides a CORDIC rotation combined with division and multiplier for implementation.

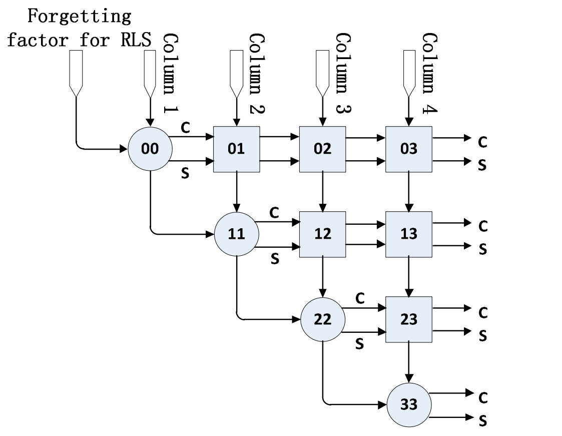

Figure 1. Systolic Array for Givens Rotation

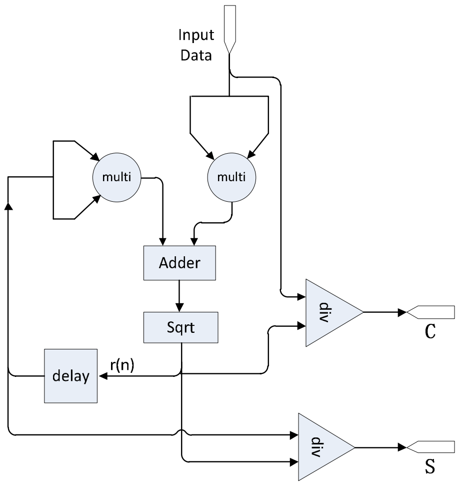

Figure 2 a) Traditional boundary cell structure [11]

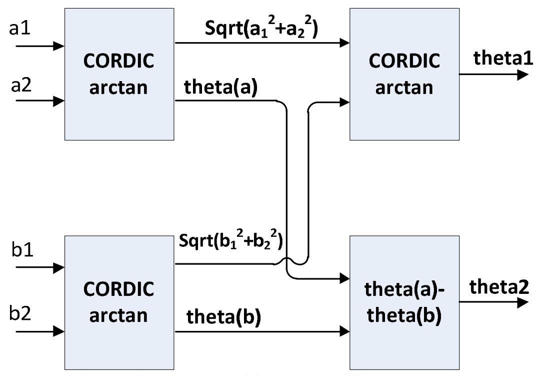

Figure 2 b) Traditional boundary cell structure [4]

B. Proposed QR decomposition design

The Givens Rotation is an iterative algorithm; the next calculation depends on the previous results. This is a critical issue in the hardware design of the systolic triangle array. Previous research showed that there is a feedback loop inside the boundary cell due to the recursive algorithm property. However, for the hardware, pipelining cannot be used in the feedback loop, resulting in a very low clock frequency and low throughput of the system. This is in conflict with the requirements of real time signal processing applications. To solve the problem for the inside feedback loop, we examine the mathematical procedure.

Assuming input matrix as:

$$\begin{pmatrix} x & a_{12} & a_{13} & a_{14} \\ r & a_{22} & a_{23} & a_{24} \end{pmatrix} \quad (11)$$

Where r is a value needs to be zeroed. So,

$$c = \frac{a}{\sqrt{r_{k-1}^2 + a^2}} \qquad s = \frac{r}{\sqrt{r_{k-1}^2 + a^2}} \quad (12)$$

For each iteration, r is updated to r_{k-1}, and in the next iteration, $r_{k-1} \times r_{k-1}$ is used. Therefore, we rewrite the equations as:

$$c = \pm\sqrt{\frac{a^2}{r_{k-1}^2 + a^2}} \qquad s = \sqrt{\frac{r^2}{r_{k-1}^2 + a^2}} \quad (13)$$

$r = r_{k-1}$ for each update, and r_{k-1} is positive as is s.

$$r_k^2 = r_{k-1}^2 + a^2 \quad (14)$$

The proposed boundary cell can be implemented using the above equations, as shown in Figure 3. Following the equation (13), the square root actually was extracted from iteration bound, and iteration structure corresponds to equation (14).Within the cell, the multiplier and square root operations are removed from the feedback loop. An adder and a multiplier can use the MAC hard core called DSP48 available inside the Xilinx Virtex 5 FPGA fabric; this can provide highest resource efficiency. S is positive all the time, whereas the sign digit is determined by input data.

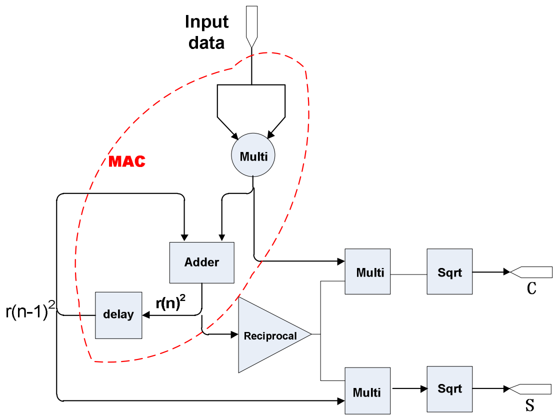

Figure 3. Improved boundary cell structure

For the design, we set the input data as 1 bit sign digit, 1 bit integer digit, and 14 bits fraction digits as an example.

Sign (1 bit)	Integer (1 bit)	Fraction (14 bits)

Figure 4. Number representation

However, for the structure shown in Figure 3, we cannot get the direct result of the boundary cell itself since only r^2 is calculated each time. If we want the value of r, another square root core is necessary for the computation. Assuming input data obey 1 sign digit, 1 integer digit, and 14 bits fractional digits, which range from -1.99999 to 1.99999, the r^2 is between 0 to 3.99999. Therefore, the integer part will be two digits. The square root core can only deal with one integer digit fraction number using CORDIC IP core. Hence, in order to obtain the result, we write *r* as:

$$r = \sqrt{r^2} = \sqrt{2} \times \sqrt{\frac{1}{2} r^2} \quad (15)$$

For $0 < r^2 < 4$, so: $0 < \frac{1}{2} r^2 < 2$

Thus, data needs to be shifted right and then the square root will follow. After the square root operation, we multiply with $\sqrt{2}$. $\sqrt{2}$ can be written as "0101101010000010". Here we use sub-expression "101" for the optimization. So the whole constant multiplier needs 4 adders instead of 5. Equations (16) and (17) are calculated using Figure 5 below.

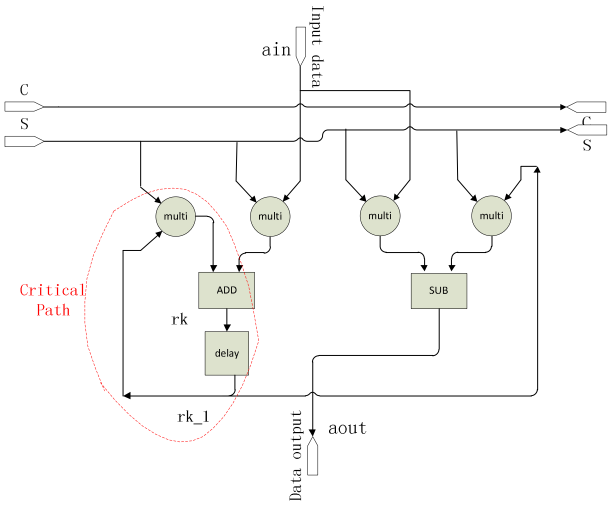

Figure 5. Internal cell structure and critical path

$$a_{out} = -s a_{in} + c r_{k-1} \quad (16)$$

$$r_k = a_{in} c + s r_{k-1} \quad (17)$$

To accelerate speed in the rotator cell, we need to first find the critical path inside the circuit. The latency of the rotator cell is determined from input data to output data, while the feedback loop from output delay to multiplier determine the maximum clock frequency. In order to obtain the best performance with resource optimization, two DSP48 hard cores and two resource-saving multipliers are used to finish the task. Additionally, the cosine and sine inputs also call for delay for the next rotator cell computation to follow the timing.

For a high scalability, a simple interface is necessary for each cell between rows and columns. Data will be still running even at the end of inputs due to the feedback loop of the pipelining system. Therefore, the timing issue becomes very critical in the design. A communication protocol (state machine) between different cells is designed to give a hand shaking signal to other cells. This way, the rotator cell and boundary cell have very strong scalability for different column and row sizes.

We select a Xilinx Virtex 5 XUP [11] device as our implementation platform. Virtex 5 provides a reconfigurable fabric, including Block RAMs for FIFOs and on-chip memory, DSP 48 blocks for Multiply Accumulates (MACs), clock management, etc.

IV. Error Analysis and Simulation Results

In the design, 16-bit fixed-point number representation is used. A testbench is created for testing the results with MATLAB, which generates random numbers for the matrix. The matrix is then imported for test vectors. The Modelsim behavior simulation result show that the total latency is 180 clock cycles, with maximum frequency worked at 246MHz. Every cell works in parallel and the stream latency is 5 clock cycles. A MATLAB generated test vector is given as input:

$$x = \begin{pmatrix} 0.8147 & 0.6323 & 0.9575 & 0.9572 \\ 0.9058 & 0.0975 & 0.9648 & 0.4854 \\ 0.1270 & 0.2784 & 0.1576 & 0.8002 \\ 0.9133 & 0.5469 & 0.9706 & 0.1418 \end{pmatrix} \tag{18}$$

Results as calculated by MATLAB using floating-point representation are:

$$Q = \begin{pmatrix} 0.5332 & 0.4892 & 0.6519 & -0.2268 \\ 0.5928 & -0.7162 & 0.1668 & 0.3283 \\ 0.0831 & 0.4507 & -0.0989 & 0.8833 \\ 0.5978 & 0.2113 & -0.7331 & -0.2461 \end{pmatrix} \tag{19}$$

$$R = \begin{pmatrix} 1.5279 & 0.7450 & 1.6758 & 0.9494 \\ 0 & 0.4805 & 0.0534 & 0.5112 \\ 0 & 0 & 0.0579 & 0.5218 \\ 0 & 0 & 0 & 0.6142 \end{pmatrix} \tag{20}$$

From Modelsim, the hardware results are shown below:

$$R = \begin{pmatrix} 1.5278 & 0.7448 & 1.6755 & 0.9492 \\ 0 & 0.4804 & 0.0535 & 0.5110 \\ 0 & 0 & 0.0579 & 0.5176 \\ 0 & 0 & 0 & 0.6152 \end{pmatrix} \tag{21}$$

It can be seen that R(3,4)=0.5176 while the MATLAB result is 0.5218, which is quite different. The reason is the accumulation error due to the previous iteration. If the word length is doubled, the error will be under 10^{-4}.

Synplify [13] tool is used for FPGA sythesis. The FPGA resource utilization is shown in Table I.

Table I. Synplify FPGA Synthesis Results

Number of Slice Registers	16,929 out of 138,240 12%
Number of Slice LUTs	10,899 out of 138,240 7%
Number used as Logic	10,457 out of 138,240 7%
Number used as Memory	426 out of 36,480 1%
Number of DSP48Es	28 out of 128 21%

The sythesis report also shows that:

- Minimum period: 4.062ns (Maximum Frequency: 246.160MHz)
- Minimum input arrival time before clock: 2.988ns
- Maximum output required time after clock: 2.826ns

V. Conclusion and Future Work

In this project, improved FPGA implementation of a Givens Rotation is presented. The feedback loop problem is address, and a very high throughout is achieved. The results from Modelsim and MATLAB are compared to verify the correct functionality. Further work is expected on large matrix size decomposition as well as using dynamic reconfigurable technology to optimize size and area. Another issue is that the division operation creates a large latency. Future research would focus on fast inverse square roots on FPGA and VLSI implementation to reduce latency.

References

[1] S. Aslan, E. Oruklu, and J. Saniie, "Realization of area efficient QR factorization using unified division, square root, and inverse square root hardware," in *IEEE International Conference on ElectroInformation Technology*, 2009, pp. 245-250.

[2] J. Becker, M. Platzner, and S. Vernalde, Eds., *Field Programmable Logic and Application*, vol. 3203. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1149-1151.

[3] S. Chandrasekaran, M. Gu, J. Xia, and J. Zhu, "A fast QR algorithm for companion matrices," *Operator Theory :Advances and Applications*, vol. 179, Birkhäuser Verlag Basel, Switzerland, 2007, pp. 111-143.

[4] J. R. Cavallaro and A. C. Elster., "A CORDIC Processor Array for the SVD of a Complex Matrix," *Elsevier, SVD AND SIGNAL PROCESSING, II*, 1991, pp. 227-239.

[5] J. R. Cavallaro and F. T. Luk, "CORDIC arithmetic for an SVD processor," in *1987 IEEE 8th Symposium on Computer Arithmetic (ARITH),* 1987, pp. 113-120..

[6] S. O. Haykin, *Adaptive Filter Theory,* 4th Ed., Prentice Hall, New Jersey, 2002.

[7] M. K. Gay, "Real-time FPGA implementation of adaptive beamforming using QR decomposition," in *IEEE High Performance Extreme Computing Conference*, 2005.

[8] M. McGraw-Herdeg, D. P. Enright, and B. S. Michel, "Benchmarking the NVIDIA 8800GTX with the CUDA Development Platform," in *IEEE High Performance Extreme Computing Conference*, 2007.

[9] O. Egecioglu, "Givens and Householder Reductions for Linear Least Squares on a Cluster of Workstations," University of California at Santa Barbara, Santa Barbara, CA, USA, 1995.

[10] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic arrays," in *Proceedings of the Society of PhotoOptical Instrumentation Engineers Conference Series*, 1981, vol. 298, pp. 19-26.

[11] Xilinx, "Xilinx ISE," 2012. [Online]. Available: http://www.xilinx.com/.

[12] L. Semiconductor, "LatticeECP/EC FPGAs: A Systolic Array Processor for Software Defined Radio," 2005.

[13] Synopsys, "Synplify Pro," 2012. [Online]. Available: http://www.synopsys.com.