An edited version of this work was publiched in IEEE TRANS. ON CIRCUITS AND SYSTEMS–II, VOL. 62, NO. 9, SEPT 2015
DOI:10.1109/TCSII.2015.2435753

High-Throughput FPGA Implementation of QR Decomposition

Sergio D. Muñoz and Javier Hormigo

***Abstract*—This brief presents a hardware design to achieve high-throughput QR decomposition, using Givens Rotation Method. It utilizes a new two-dimensional systolic array architecture with pipelined processing elements, which are based on the COordinate Rotation DIgital Computer (CORDIC) algorithm. CORDIC computes vector rotations through shifts and additions. This approach allows a continuous computation of QR factorizations with simple hardware. A fixed-point FPGA architecture for 4×4 matrices has been optimized by balancing the number of CORDIC iterations with the final error. As a result, compared to other previous proposals for FPGA, our design achieves at least 50% more throughput, and much less resource utilization.**

***Index Terms*—QR Decomposition, systolic array, pipelined, FPGA, high-throughput, CORDIC**

I. Introduction

MOST of the advanced signal processing algorithms are based on algebraic matrix operations. Many examples of this are found in wireless communication, such as multiple-input-multiple-output (MIMO), beam-forming, multi-user detection and cancellation, etc [1]. One useful operator for these matrix operations is QR factorization, especially for MIMO technologies [2] [3] and adaptive filtering [4]. Some of this applications require high-throughput QR decomposition but are for small matrix sizes. Thus, many works have addressed the parallel hardware implementation of this operation for either ASIC or FPGA technologies. In this work, we focus on high-throughput computation for small matrices on FPGAs.

The Givens Rotation Method (and its variations) is probably the most widely used to implement QR decomposition by hardware due to its robust numerical properties and its easy parallelization [5]. In the literature, there are several papers in which QR factorization has been implemented on FPGA by using this method. Although, serial approaches or linear systolic arrays may be used [6], to achieve high throughput, the most common hardware implementation is through two-dimension (2D) systolic arrays, such as in [7], [8], [2], [9], [10], [11]. A 2D systolic array is a parallel grid structure where processing elements (PEs) works in parallel and are locally interconnected. This systolic architecture allows the exploitation of different grades of parallelism inherent to the the Given Rotation algorithm. Thus, these approaches have high-throughput and relatively low latency, at the cost of considerable area consumption.

In this work, through combining several ideas, we have designed a new architecture which improves previous high-throughput FPGA implementations. It is based on the CORDIC algorithm to simplify hardware, pipelining the PEs to obtain better throughput, along with a different schedule for performing the Given Rotations to reduce latency. As a result, the proposed architecture has very high-throughput and low latency, with a relatively reduced area consumption. They also have a very simple control and communication logic.

The next sections of this brief are organized as followed: Section II reviews some important aspects of the QR decomposition using Givens Rotations, along with a brief review of some previous works proposed in the literature. Section III presents the proposed architecture to achieve high-throughput. In Section IV the results of the FPGA implementation are studied and compared with other previous works. Finally, Section V provides the conclusions of this work.

II. Givens Algorithm and previous FPGA implementations

Given a matrix $A^{m\times n}$, this is equivalent to the product of two factors, i. e. $A = Q \cdot R$, in which matrix $Q^{m\times m}$ is orthogonal and $R^{m\times n}$ is an upper triangular matrix [5]. The computation of these two factors is called QR decomposition or factorization.

The Givens Method achieves a QR factorization through unitary transformations, called Givens Rotations, which selectively allow the introducing of a zero element [5]. Givens rotation matrix has rank-two corrections about identity matrix, where the rank (i, j) is replaced by orthogonal values based on sines and cosines.

$$\begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix} \times \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} a'_1 \\ 0 \end{bmatrix} \quad (1)$$

As an example, a Givens rotation is represented in Eq. 1 for a 2×1 matrix, where the resultant matrix has a new inserted zero; this can be extrapolated to any other matrix size. The rotation angle θ must be computed beforehand by the formula $\arctan(\frac{a_2}{a_1})$. Alternatively, these values can also be calculated by Eq. 2 and Eq. 3.

$$\cos(\theta) = \frac{a_{i,k}}{\sqrt{a_{i,k}^2 + a_{j,k}^2}} \quad (2) \qquad \sin(\theta) = \frac{-a_{j,k}}{\sqrt{a_{i,k}^2 + a_{j,k}^2}} \quad (3)$$

Accordingly, Givens Method algorithm starts zeroing the lower elements, from the first column to the last one, and, on each column, starting from the bottommost element to the

This work was supported in part by the Ministry of Education and Science of Spain and Junta of Andalucía under contracts TIN2013-42253-P and P07-TIC-02630, respectively.

The authors are with the Department of Computer Architecture, Universidad de Málaga, Málaga E-29071 Spain (e-mail: smunoz@uma.es; fjhormigo@uma.es).

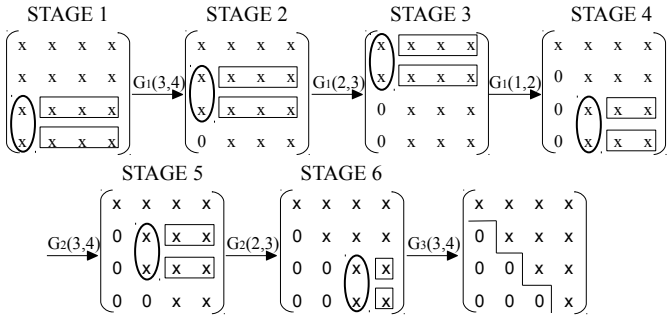

Fig. 1. Usual Givens rotation schedule for 4×4 matrices.

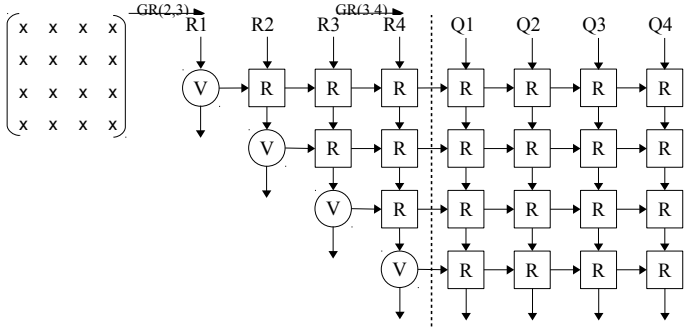

Fig. 2. Column-based 2D-systolic array for 4×4 matrices.

diagonal element. The upper triangular matrix R is achieved by accumulating the Givens Rotations on the initial matrix. Similarly, Q is obtained when the same rotations are applied to the identity matrix.

As an example, Fig. 1 illustrates the application of the Givens Method on a 4×4 matrix to achieve an upper triangular matrix R in 6 stages. Each arrow represents a Givens Rotation, where $G_k(i,j)$ specifies the involved rows (i,j) and the column k where a zero will be inserted. The circular areas indicate the elements selected to calculate the rotation angle, whereas the squared areas delimit those elements that will be rotated using said angle.

It is clearly seen that this algorithm has different levels of parallelism that could be exploited depending on the selected architecture. Several works, such as in [10] and [7], have proposed a 2D systolic array similar to the one showed in Fig. 2. In this architecture, each PE always works over elements on the same column. On each row of the architecture, the Givens rotations may be performed in parallel using as many PEs as non-zero elements are within the row of the matrix.

Besides this parallel computation, this configuration has the advantage of the two different types of PEs used, one (V) to compute the rotation angle which, at first, requires much more complicated operations, and another (R) to perform the effective rotations, which is much simpler. Each row of PEs only needs one PE type V and the rest as type R. Thus, although they need more PEs, the number of PEs type V (much more complex) are reduced and, then, the overall area may be also reduced.

This architecture is used in [7], where standard arithmetic

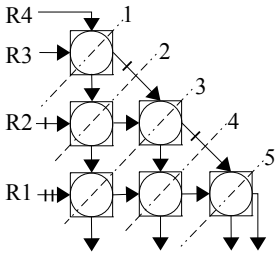

Fig. 3. Row-based 2D-systolic array for 4×4 matrices.

operations are utilized to implement the PEs. In [10], iterative CORDIC circuits are used instead, which reduces area consumption. However, in these approaches, due to data dependencies between consecutive rotations, the PEs of the last rows are idle most of the time, which means an important waste of resources. Besides this, the same data dependency prevents the use of pipelining internally in the PEs, which limits the achievable throughput.

A different and older approach is the one used in [8] and [12] which is shown in Fig. 3. On this scheme, a PE completely performs a Givens rotation for all elements of the two rows. Thus, the two operations involved in a Givens rotation have to be combined in one PE, making it more complex, although much fewer PEs are required. The main advantage of this approach is the fact that the only data dependency, which prevents the pipelining within the PEs, is the one between the computation of the rotation angle and the rotation itself. In [12], they propose to interleave columns of different input matrices to overcome this dependency, but this is unpractical for many applications, especially for deep pipelines. On the other hand, Square Root and Division Free Givens Rotations (SDFG) [13] are utilized in [8], where this dependency is eliminated by means of a pre- and post-processing which allows pipelining of the PEs. Thus, this architecture achieves a high-throughput, but the complexity of the operations involved also requires a high utilization of resource.

III. Proposed architecture

Similarly to the work in [8], we propose to use a 2D-array architecture where each PE works with all the elements of the same row and these PEs are pipelining, to achieve high throughput. Yet, at the same time, we propose different connections for the PEs within the 2D array to reduce latency. Moreover, the PEs are designed based on the CORDIC algorithm to implement this pipeline in a simpler way, which produces a system with lower area and higher throughput. Next, we present some details of this architecture.

A. Givens Rotations Schedule

The classic schedule to implement the Givens algorithm, as it is previously described in Fig. 1, starts zeroing the bottommost element of the first column, and serially continues up in the same column until this column is finished. Then, the same procedure is performed over the next column and so on, until the matrix is triangular. The 2D systolic array

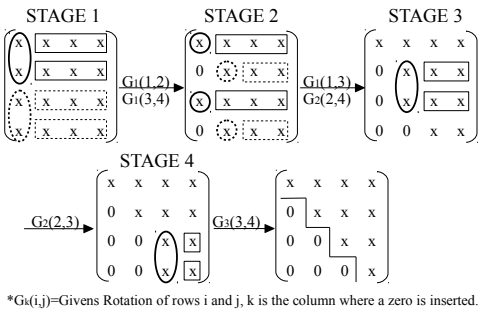

Fig. 4. Givens rotations scheduled for increasing parallelism.

improves this schedule by performing several rotations at the same time. Concretely, as it is indicated in Fig 3, all PEs in the same diagonal work in parallel, significantly reducing the number of steps required for one matrix computation. However, this schedule can be performed with a higher level of parallelism, if as many rows as possible were rotated simultaneously. Thereby the algorithm decreases latency by reducing its amount of steps. We should note that the number of Givens rotations remains the same but the number of Givens rotations by step is increased.

To reduce the number of steps as much as possible, on each step all suitable pairs of rows (i.e., two unselected rows that contain the same number of zeros to the left) are selected and they are rotated in parallel. This is repeated until an upper triangular matrix is achieved. Fig. 4 illustrates how a 4×4 matrix is factorized by using this schedule. In this figure, two types of lines are described, dotted and continuous; each one represents a Givens rotation made simultaneously. In the first stage, two Givens rotations are performed concurrently, it takes the adjacent rows $(1, 2)$ and $(3, 4)$. This means inserting two zeros in rows 2 and 4 as shown on the second stage. Following, rotations $G_1(1, 3)$ and $G_2(2, 4)$ are calculated, finishing the computation on the first column. Then, the first row will not be used again, restricting the algorithm to only one rotation by stage for the two last stages. Therefore, only four stages are required using this schedule.

B. CORDIC-based processing elements

CORDIC (COordinate Rotation DIgital Computer) is an iterative algorithm based on shifts and additions which allows calculating many elementary functions with a very simple hardware [14]. The same CORDIC circuit may operate in two modes, vectoring or rotation mode. The former rotates an input vector (X, Y) until its Y coordinate reaches a zero value, returning angle θ and the X coordinate has rotated. The latter rotates an input vector (X, Y) with a determined angle θ. Therefore, a CORDIC unit could be used to compute the angle for a Givens Rotation (vectoring mode) and, then, performs the rotation through the rest of the row using said angle (rotation mode).

Many different circuits based on CORDIC have been proposed in the literature to perform Givens Rotations. To achieve our goal, we have selected the one used to implement a linear systolic array in [15]. It is a pipeline architecture

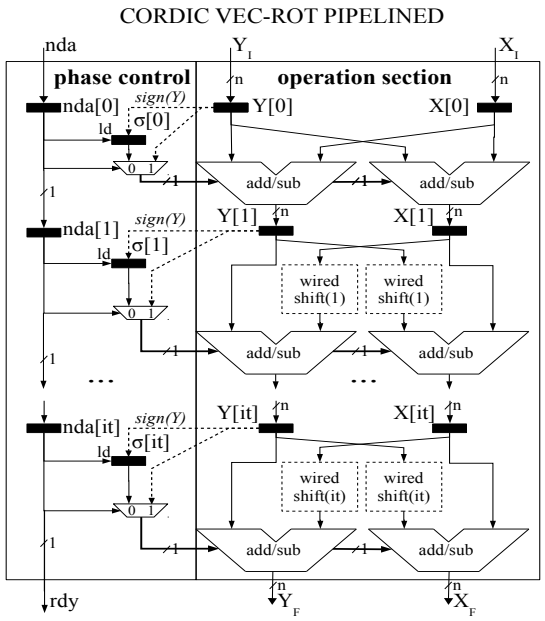

Fig. 5. CORDIC-based processing element.

which performs both vectoring and rotation mode. Due to data dependency of the linear array, the circuit presented in [15] needs matrix interleaving to take advantage of the pipeline design. However, within our row-based 2D systolic array, the pipeline is used in a natural way. This CORDIC approach replaces the computation of the rotation angle θ by the direction of each micro-rotation. This direction is indicated by the sign of Y on each CORDIC stage and it is stored in a register (σ) to be used in subsequent rotations. Thus, this circuit allows overlapping the computation of the angle with the rotation of the corresponding rows.

Fig. 5 shows this circuit divided into two sections. The right section is the operation section that contains the typical CORDIC x-y data-path. The left section represents the control hardware which, in vectoring mode, selects the rotation direction and updates the σ registers. These registers configure the $adds/subs$ in the rotation mode. An active nda signal indicates a new angle computation (vectoring mode) on this stage. Then, $sign(Y)$ is used to control the add/sub and it is stored in the σ register. While the active nda signal goes through the pipeline, the rest of the elements of the corresponding rows are introduced into the circuit to be rotated using said stored directions (rotation mode). Therefore, both computations are overlapped and, furthermore, it is clearly seen that a new Givens rotation may be introduced in the pipeline before the actual one was completely finished.

The constant scale factor introduced by the CORDIC algorithm [14] is compensated by multiplying the output values by its inverse. As we will see in the next section, constant multipliers or embedded multipliers could be utilized for this operation.

C. Proposed circuit

Using the schedule described in section III-A, the proposed architecture is derived by assigning a PE to each Givens

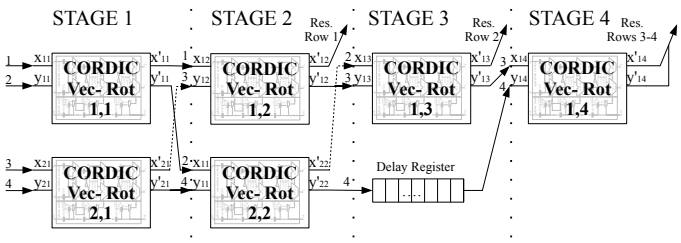

Fig. 6. CORDIC base architecture implemented to factorize 4×4 size matrices.

rotation. Fig. 6 illustrates the architecture for 4×4 matrices. There are four stages, the two first ones with two PEs, since two Givens rotations are performed in parallel, and only one in the two others. Input and output buses are connected directly from one stage to the next. Only a FIFO register is required on stage 3, since one of the rows computed in stage 2 is used in stage 4. Not much logic is required for synchronization of this architecture, due to its pipelined structure. The nda signals of the PEs on the first stage may be set externally, or using a counter if the flow of input matrices is regular. In the next stages, the nda inputs are connected to the nda outputs of the previous stage. In some PEs, the nda signal has to be delayed one extra cycle to compensate for the zero elements. In the first stage, all rows are introduced simultaneously, element by element (input matrix followed by identity matrix). Furthermore, thanks to its fully pipelined architecture, a new matrix computation could start right after the last element is introduced. Therefore, a very high throughput is achieved (for this example, one matrix computation each 8 cycles).

IV. Performance analysis and comparison

Using the proposed architecture, a VHDL fixed-point QR decomposition core for 4×4 matrices has been designed. Said core allows us to configure both bit-width and number of CORDIC iterations. This core has been simulated and synthesized using Xilinx ISE 14.3 software, and implemented and evaluated using a hardware Virtex-6 XV6VLX240T speed -2 FPGA platform. To confirm the correctness of the proposed core, first, it has been tested with a wide range of random matrices and the results have been checked using Matlab.

Secondly, to improve the area and the latency of the proposed circuit, we have experimentally studied how the number of CORDIC iterations influences the error of its results. To do this, different circuits, using three word-lengths (16, 24 and 32 bits), have been implemented on our hardware platform for several numbers of CORDIC iterations. Using each one of these circuits, the QR decomposition has been calculated for 50,000 random matrices whose results have been checked by computing $Q^t * R$ and comparing it with the original matrix A. In Table I, the maximum error detected on these comparisons is presented for each tested configuration. It is clearly observed that, at first, the maximum error decreases when the number of iterations increases, due to the better approximation achieved for the rotation angle. However, at a certain point, the error starts to slightly increase due to the accumulated rounding error. Thus, to obtain minimum error while reducing the area and the latency, the best configurations for 16, 24 and 32 bit-widths are 10, 18, 26 CORDIC iterations, respectively.

TABLE I
Maximum error of QR factorization for 16, 24 and 32 bit word-lengths depending on the number of CORDIC iterations.

Word-Lenght	**16 bits**				
CORDIC-Iter.	8	10	12	14	16
Latency (cycles)	44	52	60	68	76
Max. Error	1.4e-3	5.8e-4	6.9e-4	7.3e-4	8.9e-4
Word-Lenght	**24 bits**				
CORDIC-Iter.	12	16	18	20	24
Latency (cycles)	60	76	84	92	108
Max. Error	8.06e-5	6.01e-6	3.5e-6	3.7e-6	4.7e-6
Word-Lenght	**32 bits**				
CORDIC-Iter.	20	24	26	28	32
Latency (cycles)	92	108	116	124	140
Max. Error	3.3e-7	2.4e-8	9.4e-9	1.6e-8	2.3e-8

Table II shows the implementation results for the three analyzed word-lengths, each one with three different approaches for the scale factor compensation required by the CORDIC algorithm (see Section III-B). All designs use the optimum number of iterations previously computed. Two approaches use the embedded multipliers (DSP48E) which typically exist in FPGAs, either non-pipelined or pipelined (Multiplier A and Multiplier B, respectively). While, the third approach uses pipelined constant-coefficient (Multiplier C) designed with Xilinx Core Generator. There are not great area differences between the approaches using DSP48, but the one pipelined allows much higher clock frequency and, consequently, much better throughput at the cost of a moderate latency increase. On the other hand, the number of slices used to implement the constant-coefficient multipliers is relatively high compared to the rest of the circuit. Then, although this approach achieves the same throughput as the pipelined-DSP48 one, and even less latency, this approach may be only selected if it is required to save DSP48 for different computations.

Finally, to study the effectiveness of our proposal, from the literature, we have selected some representative works which provide enough data to perform a reasonable comparison. Tab. III shows the mean results of these works along with the ones for our 16-bit circuit using Multiplier A. To provide a fair comparison, we have synthesized our architecture on equivalent FPGAs as those works, concretely Virtex4 (XC4VFX60-11) and Virtex5 (XC5VTX150T-2).

Regarding the performance, the only design with a throughput relatively close to ours is the one in [8]. Similarly to our proposal, it uses pipelined PEs and has practically the same latency in clock cycles. However, its lower maximum frequency provides that our proposal presents about 35% less latency (seconds) and 50% more throughput than the design in [8]. The better critical path of our design is explained mainly by the simplicity of the CORDIC architecture. On the other hand, the circuits presented in [10] and [7] have a throughput which is one order of magnitude lower than ours, since their PEs are iterative.

Regarding the area, our design clearly requires several

TABLE II
FPGA IMPLEMENTATION RESULTS FOR 16, 24 AND 32 BITS WORD-LENGTHS.

Device	**Xilinx Virtex 6 XC6VLX240T -2**		
Word-Lenght	**16 bits**	**24 bits**	**32 bits**
CORDIC-Iterations	10	18	26
Cycles/Matrix	8	8	8
Multiplier A	**Dedicated Non-Pipelined DSP48E**		
Latency (cycles)	52	84	116
DSP48E	12(1%)	24(3%)	48(6%)
Max. Freq. (Mhz)	234.5	170.3	117.1
Slice Registers	2,402(1%)	5,888(1%)	10,844(3%)
Slice LUTs	2,498(1%)	6,109(4%)	11,337(7%)
Multiplier B	**Dedicated Pipelined DSP48E**		
Latency (cycles)	60	100	140
DSP48E	12(1%)	24(3%)	48(6%)
Stages Pipe. Mult.	2	4	6
Max. Freq. (Mhz)	421.1	398	377.6
Slice Registers	2,457(1%)	6,046(1%)	11,520(3%)
Slice LUTs	2,587(1%)	6,220(4%)	11,225(7%)
Multiplier C	**Const. Coef. Pipelined (without DSP48E)**		
Latency (cycles)	60	96	132
Stages Pipe. Mult.	2	3	4
Max. Freq. (Mhz)	421.1	398	377.6
Slice Registers	3,262(1%)	8,400(2%)	16,259(5%)
Slice LUTs	3,641(2%)	8,596(5%)	16,100(10%)

TABLE III
COMPARATIVE WITH OTHER FPGA IMPLEMENTATIONS

Work	[7]	[8]	[10]	This Work (Multiplier A)	
DEVICE	**Virtex5**	**Virtex4**	**Virtex5**	**Virtex4**	**Virtex5**
W-Length	**16 bits**	**16 bits**	**18 bits**	**16 bits**	
Latency(cl)	180	51	80	52	
Latency(μs)	0.73	0.35	0.72	0.23	0.20
Max. Freq. (Mhz.)	246	144	111	222	254
Throughput MMatrices/sec	1.36	18.05	2.13	27.70	31.70
Slice Reg.	16,929	5,891	7,811	2,311	2,085
Slice LUTs	10,899	9,810	2,609	2,085	2,671
DSP48	28	41	—	12	
Max. Error	4.2e-3*	—	6.32e-3	5.8e-4	

*Note. Maximum error for [7] is obtained from a factored matrix sample, a complete error study has not been done.

times less resources than the others, considering all kinds of resources. The closest one is the circuit in [10], which also uses a CORDIC-based architecture. Although it requires practically the same number of LUTs and no multipliers, the number of registers is more than three times greater. This is mainly explained by the much greater number of PEs presented in the architecture and the use of carry-save arithmetic.

Taking all these results into account, we could conclude that the architecture proposed herein presents much better throughput, and much lower resource utilization, than previously proposed works. Moreover, this throughput could be doubled by using the pipelined version of DSP48 at the cost of a moderate latency increase.

V. CONCLUSION

This brief presents a fixed-point systolic architecture to achieve high-throughput QR Decomposition for small matrices. This is achieved by performing as many Givens rotations as possible in parallel in an unrolled architecture, and using a pipelined CORDIC circuit which allows completely overlapping the angle computation and the rows rotation. Thus, this highly pipelined circuit performs a $n \times n$ matrix decomposition each $2n$ clock cycles. The FPGA implementation of this architecture for 4×4 matrices has been optimized for different word-lengths by selecting the appropriate number of CORDIC iterations. Comparing with previous FPGA approaches, our proposal highly improves both performance and resources utilization.

REFERENCES

[1] K. Sarrigeorgidis and J. Rabaey, "A scalable configurable architecture for advanced wireless communication algorithms," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 45, no. 3, pp. 127–151, 2006.

[2] Z.-Y. Huang and P.-Y. Tsai, "Efficient implementation of QR decomposition for gigabit MIMO-OFDM systems," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 58, no. 10, pp. 2531–2542, Oct 2011.

[3] Y. Wu, J. McAllister, and P. Wang, "High performance real-time preprocessing for fixed-complexity sphere decoder," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, Dec 2013, pp. 1250–1253.

[4] S. Chan and X. Yang, "Improved approximate QR-LS algorithms for adaptive filtering," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 51, no. 1, pp. 29–39, Jan 2004.

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[6] K. Boonyi, J. Tagapanij, and A. Boonpoonga, "FPGA-based hardware/software implementation for MIMO wireless communications," in *Electrical Engineering Congress (iEECON), 2014 International*, March 2014, pp. 1–4.

[7] S. Aslan, S. Niu, and J. Saniie, "FPGA implementation of fast QR decomposition based on Givens rotation," in *Circuits and Systems, 2012 IEEE 55th International Midwest Symposium on*, 2012, pp. 470–473.

[8] M. Abels, T. Wiegand, and S. Paul, "Efficient FPGA implementation of a high throughput systolic array QR-decomposition algorithm," in *Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on*, Nov 2011, pp. 904–908.

[9] R.-H. Chang, C.-H. Lin, K.-H. Lin, C.-L. Huang, and F.-C. Chen, "Iterative QR decomposition architecture using the modified Gram-Schmidt algorithm for MIMO systems," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 57, no. 5, pp. 1095–1102, May 2010.

[10] D. Chen and M. Sima, "Fixed-point CORDIC-based QR decomposition by Givens rotations on FPGA," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, Nov 2011, pp. 327–332.

[11] G. Prabhu, B. Johnson, and J. Rani, "FPGA based scalable fixed point QRD core using dynamic partial reconfiguration," in *VLSI Design (VLSID), 2015 28th International Conference on*, Jan 2015, pp. 345–350.

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

[13] J. Gotze and U. Schwiegelshohn, "A square root and division free Givens rotation for solving least squares problems on systolic arrays," *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 4, pp. 800–807, Jul 1991.

[14] M. D. Ercegovac and T. Lang, *Digital arithmetic*. Elsevier, 2003.

[15] J. Luo and C. Jong, "Scalable linear array architectures for matrix inversion using Bi-z CORDIC," *Microelectronics Journal*, vol. 43, no. 2, pp. 141 – 153, 2012.