CORDIC-BASED GIVENS QR DECOMPOSITION FOR MIMO DETECTORS

A Thesis
Presented to
The Academic Faculty

by

Minzhen Ren

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2013

CORDIC-BASED GIVENS QR DECOMPOSITION FOR MIMO DETECTORS

Approved by:

Professor Xiaoli Ma, Committee Chair
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor David V. Anderson, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor James H. McClellan
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Matthieu R. Bloch
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: 19 August 2013

To my parents

ACKNOWLEDGEMENTS

I would like to thank my advisors and my reading committee for their strong support over the past two years! Especially, I want to thank Dr. David V. Anderson, Dr. Brian J. Gestner (Sandia National Labs), Dr. Linghua Kong (formerly with CATEA), Dr. Xiaoli Ma, and Dr. James H. McClellan for the research opportunities they have provided over the past five years, and for their guidance and encouragement, helping me build a strong academic background and find my research interests! I would also like to thank my supervisor and colleagues in Texas Instruments Inc. (Greenville, SC) for providing me with the valuable experience, which helps me build critical skills for finishing my research! Finally, I want to thank my friends and my family for making an unforgettable journey through my academic years!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Lower-case, bold-face letters	column vectors, or complex-valued numbers.
Upper-case, bold-face letters	matrices.
Superscript $\{\cdot\}^{\mathcal{H}}$	Hermitian.
Superscript $\{\cdot\}^{\mathcal{T}}$	transpose.
Superscript $\{\cdot\}^{-1}$	inverse.
$\lvert\cdot\rvert$	absolute value of a scaler.
$\lVert\cdot\rVert_{L_1}$	L_1-norm approximation.
$\lVert\cdot\rVert_{\mathcal{F}}$	Frobenius norm.
$\lceil\cdot\rceil$	integer ceilling.
$\Re\{\cdot\}$	the real part.
$\Im\{\cdot\}$	the imaginary part.
$\{i.f\}$	fixed-point numbers precision: i-ineger bits and f-fraction bits.

GLOSSARY

ASIC application-specific integrated circuit, p. 44.

CORDIC cordinate rotation digital computer, p. 2.

flops floating-point operations, p. 8.

FPGA field-programmable gate array, p. 2.

FSM finite-state machine, p. 29.

MIMO multiple-input multiple-output, p. 1.

PAR place-and-route, p. 28.

PE processing element, p. 19.

QRD QR decomposition, p. 1.

RAM random-access memory, p. 26.

RTL register-transfer level, p. 28.

TACR three angle complex rotation, p. 9.

SUMMARY

The object of the thesis research is to realize a complex-valued QR decomposition (QRD) algorithm on FPGAs for MIMO communication systems. The challenge is to implement a QRD processor that efficiently utilizes hardware resources to meet throughput requirements in MIMO systems. By studying the basic QRD algorithm using Givens rotations and the CORDIC algorithm, the thesis develops a master-slave structure to more efficiently implement CORDIC-based Givens rotations compared to traditional methods. Based on the master-slave structure, an processing-element array architecture is proposed to further improve result precision and to achieve near-theoretical latency with parallelized normalization and rotations. The proposed architecture also demonstrates flexible scalability through implementations for different sizes of QRDs. The QRD implementations can process 7.41M, 1.90M and 0.209M matrices per second for 2×2, 4×4 and 8×8 QRDs respectively. This study has built the foundation to develop QRD processors that can fulfill high throughput requirements for MIMO systems.

CHAPTER I

INTRODUCTION

The QR decomposition (QRD) is a mathematical practise to factorize an $m \times n$ matrix $\boldsymbol{A}$ into an upper-triangular matrix $\boldsymbol{R}$ and an orthogonal matrix $\boldsymbol{Q}$ so that $\boldsymbol{A} = \boldsymbol{QR}$. It is one of the most commonly-used approaches to solve a system of linear equations with applications in many different areas such as digital signal processing, control systems and telecommunication systems. Symbol detection in multiple-input multiple-output (MIMO) communication systems is one of many important QRD applications. In order to achieve higher data rate and reliability, MIMO systems utilize multiple antennas at both sides of a communication link, so multiple data streams can be transmitted in parallel [8, 9]. Consequently, current generation wireless standards, such as IEEE 802.11n, IEEE 802.16e, and 3GPP-LTE, have employed MIMO technology, and MIMO technology will likely be used more intensively in future generation standards. A challenge for MIMO technology is that the computational complexity associated with optimally decoding the received signal vectors can be very high. This complexity is exacerbated because MIMO technology tends to be used in very high-throughput systems.

Many symbol detection algorithms have been proposed to face the challenge of building low-complexity, high-throughput MIMO receivers. Suboptimal or non-linear detectors can achieve lower complexity at the cost of lower performance in terms of bit-error rate. Maximum-likelihood detection algorithms that achieve near-optimal performance, such as tree-search-based algorithms, suffer from a complexity that grows exponentially with the signal constellation size and number of antennas. These

algorithms require the results of the QRD of a channel matrix $\boldsymbol{H}$ to proceed subsequent processing. The importance of QRDs in MIMO systems is twofold. First, the precision of QRD results directly affect the detection performance. Low-precision results can substantially degrade the symbol detection. Second, the delay introduced by QRD may be critical to the entire system. A long delay may require a redesign or simplification in other components of the system in order to meet throughput requirements.

The object of this thesis is to realize complex-valued QRD from a theoretical algorithm to a hardware implementation. By studying the fundamentals of QRD algorithms, this thesis reveals the basic building blocks for constructing a QRD algorithm based on Givens rotations and the coordinate rotation digital computer (CORDIC) algorithm. Using CORDIC-based Givens rotations, a processing-array type architecture is proposed and implemented on Xilinx field-programmable gate array (FPGA) platforms. The contribution of the thesis is not only to meet throughput requirements using minimum hardware, but to also understand the trade-off between throughput and cost. This study also builds the foundation that can lead to more sophisticated designs achieving better throughput in the future.

The remaining chapters of the thesis are organized as follows. Chapter 2 first establishes the evaluation criteria for QRD algorithms and implementations. Then, the fundamentals of real-valued QRD algorithms are introduced and extended into complex-valued cases as well as a sorted-QRD algorithm. Many existing QRD solutions in the literature are reviewed at the end of Chapter 2 to provide comparison material for this study.

Chapter 3 explains the CORDIC algorithm and the use of CORDIC as a routine to solve QRD problems. In the end, a latency-and-precision-improved algorithm is proposed. In Chapter 4, the proposed architecture is constructed based on the algorithm with practical hardware considerations. A trade-off between throughput and

hardware cost is analyzed to provide flexible solutions to meet potential throughput requirements. The proposed architecture is further implemented and evaluated in Chapter 5. Then, comparisons are made against several existing implementations reviewed in Chapter 2. Finally, Chapter 6 summarizes thesis contributions and projects future research opportunities.

CHAPTER II

FUNDAMENTALS AND EXISTING SOLUTIONS

QRD is a classic numerical linear algebra problem and has been thoroughly studied and applied to solving systems of linear equations. In this chapter, an analytical model is first established for evaluating QRD algorithms and implementations in MIMO communication systems. The model further constrains the problem and clarify the goal of this study. Next, a real-valued QRD algorithm using Givens rotations is introduced and extended to complex-valued and sorted QRD cases. Existing QRD algorithms and implementations are reviewed at the end of this chapter.

2.1 Establishing an Analytical Model and Goals

Consider a flat-fading channel transmission for a MIMO system, with n transmission antennas and an equal number of n receiving antennas. The resulting channel matrix $\boldsymbol{H}$ is an $n \times n$ complex-valued matrix. The transmitted data stream is a vector of dimension n, denoted by $\boldsymbol{s} = [s_1, s_2, \ldots, s_n]^{\mathcal{T}}$. Hence, the received data stream, $\boldsymbol{r} = [r_1, r_2, \ldots, r_n]^{\mathcal{T}}$ can be expressed as follows:

$$\boldsymbol{r} = \boldsymbol{H}\boldsymbol{s} + \boldsymbol{w}. \tag{1}$$

In flat-fading channels, the additive noise vector $\boldsymbol{w} = [w_1, w_2, \ldots, w_n]^{\mathcal{T}}$ can be characterized by a zero-mean and unit-variance white Gaussian distribution. Moreover, it is assumed that the channel matrices are perfectly known at the receiver side.

Many symbol detection algorithms, such as V-BLAST, K-Best, etc., start by decomposing a channel matrix into an upper-triangular matrix $\boldsymbol{R}$ as well as transforming the received signal vector into a different domain as $\boldsymbol{y} = \boldsymbol{Q}^{\mathcal{H}} \cdot \boldsymbol{r}$ [21, 22, 23]. Other detection schemes like lattice-reduction aided algorithms, studied in [4, 25], require

both $\boldsymbol{R}$ and $\boldsymbol{Q}$ matrices to proceed. This study focuses on computing both $\boldsymbol{R}$ and $\boldsymbol{Q}$ matrices for square channel matrices, which can be used in most of detection algorithms.

In order to evaluate QRD results numerically, two measurements are defined as follows. In the ideal case, both of these measurements should be zero.

Definition 1 (Dissimilarity)

$$Dissimilarity = \|\boldsymbol{H} - \boldsymbol{QR}\|_{\mathcal{F}}. \tag{2}$$

Definition 2 (Orthogonality) *Given that $\boldsymbol{I}_n$ is an $n \times n$ identity matrix,*

$$Orthogonality = \|\boldsymbol{I}_n - \boldsymbol{QQ}^{\mathcal{H}}\|_{\mathcal{F}}. \tag{3}$$

A QRD algorithm should achieve high numerical performance and should have manageable complexity. Meanwhile, the algorithm should be straightforward to implement in hardware. In order to evaluate the performance of QRD hardware implementations, latency and throughput are defined as follows.

Definition 3 (Latency) *The latency of a QRD hardware implementation measures the time difference between the beginning and the end of a QRD process on a single matrix. Given that the clock frequency of an implementation is known, latency can be alternatively expressed as the number of clock cycles instead of common time units.*

Definition 4 (Throughput) *Given the dimension of matrices and number precision information, the throughput measures the number of matrices that can be processed by a QRD implementation in one second.*

Long latency does not necessarily imply poor throughput performance because many implementations with deeply-pipelined datapath can have multiple matrices

being processed simultaneously. These systems can thus achieve high throughput at the expense of increased latency.

The goal of this study is to efficiently realize a QRD algorithm on FPGAs. Based on the algorithm, the hardware architecture should have flexible scalability so it can solve different sizes of QRD problems with manageable complexity. The implementation should achieve near-theoretical latency in order to fulfill the high throughput requirements in MIMO systems.

2.2 Fundamentals of QRD Algorithm

2.2.1 Real-valued QRD using Givens Rotations

There are three commonly-used algorithm families to solve QRD problems for a real-valued $m \times n$ matrix, denoted by $\boldsymbol{A}$. They are the Gram-Schmidt algorithm, Householder reflections, and Givens rotations. In general, the key to conduct a QRD is to efficiently annihilate the elements located in the lower-triangular part of the matrix. Givens rotations provide a way to selectively eliminate these elements by applying Givens rotation matrices, defined as follows:

$$\boldsymbol{G} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}. \tag{4}$$

In order to normalize a vector, the rotation angle θ is determined by the vector departure angle from the positive x-axis. For instance, to normalize a vector $\boldsymbol{v} = [v_1, v_2]^{\mathcal{T}}$, θ has to be $-\arctan(v_2/v_1)$ so that

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \sqrt{v_1^2 + v_2^2} \\ 0 \end{bmatrix}. \tag{5}$$

Alternatively, one can directly calculate $\cos(\theta)$ and $\sin(\theta)$ using trigonometric identities [5]:

$$\cos(\theta) = \frac{v_1}{\sqrt{v_1^2 + v_2^2}}, \;\; \sin(\theta) = \frac{-v_2}{\sqrt{v_1^2 + v_2^2}}. \tag{6}$$

Applying Givens rotations to $\boldsymbol{A}$ requires (4) to be generalized as follows [5]:

$$\boldsymbol{G}(i,k,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos(\theta) & \cdots & -\sin(\theta) & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \sin(\theta) & \cdots & \cos(\theta) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \\ \\ i \\ \\ k \\ \\ \\ \end{matrix} \quad . \tag{7}$$

$$\begin{matrix} & & i & & k & & \end{matrix}$$

$\boldsymbol{G}(i,k,\theta)$ annihilates the (k,i) entry of $\boldsymbol{A}$, while rotating the rest of the entries on the row i and k. Furthermore, the annihilation has to proceed systematically so that $\boldsymbol{G}(i,k,\theta)$ will not reintroduce non-zero values in previously-eliminated zero positions. Fig. 1 illustrates a systematic approach of applying QRD column-wise from top to bottom for a 3×3 matrix. While triangularizing $\boldsymbol{A}$ to obtain $\boldsymbol{R}$, the Givens QRD

$$\begin{bmatrix} \mathbb{R} & \mathbb{R} & \mathbb{R} \\ \mathbb{R} & \mathbb{R} & \mathbb{R} \\ \mathbb{R} & \mathbb{R} & \mathbb{R} \end{bmatrix} \xrightarrow{\boldsymbol{G}(1,2,\theta_{(1,2)})} \begin{bmatrix} \mathbb{R} & \mathbb{R} & \mathbb{R} \\ 0 & \mathbb{R} & \mathbb{R} \\ \mathbb{R} & \mathbb{R} & \mathbb{R} \end{bmatrix} \xrightarrow{\boldsymbol{G}(1,3,\theta_{(1,3)})} \begin{bmatrix} \mathbb{R} & \mathbb{R} & \mathbb{R} \\ 0 & \mathbb{R} & \mathbb{R} \\ 0 & \mathbb{R} & \mathbb{R} \end{bmatrix} \xrightarrow{\boldsymbol{G}(2,3,\theta_{(2,3)})} \begin{bmatrix} \mathbb{R} & \mathbb{R} & \mathbb{R} \\ 0 & \mathbb{R} & \mathbb{R} \\ 0 & 0 & \mathbb{R} \end{bmatrix}$$

Figure 1: Illustration of QRD steps by applying QRD on a 3×3 matrix column-wise from top to bottom. $\boldsymbol{G}(i,k,\theta_{(i,k)})$ represents a 3×3 Givens rotation matrix defined in (7).

algorithm accumulates a matrix $\boldsymbol{Q}$ with the product of all Givens rotation matrices, and the accumulation can be mathematically expressed as, $\boldsymbol{R} = \boldsymbol{G}_N \cdots \boldsymbol{G}_2 \boldsymbol{G}_1 \boldsymbol{A} = \boldsymbol{Q}^{-1}\boldsymbol{A}$. Since $\boldsymbol{Q}$ is an orthogonal matrix, i.e., $\boldsymbol{Q}^{-1} = \boldsymbol{Q}^{\mathcal{T}}$, it can be rewritten as

$$\boldsymbol{Q} = (\boldsymbol{G}_N \cdots \boldsymbol{G}_2 \boldsymbol{G}_1)^{\mathcal{T}} = (\boldsymbol{G}_1^{\mathcal{T}} \boldsymbol{G}_2^{\mathcal{T}} \cdots \boldsymbol{G}_N^{\mathcal{T}}). \tag{8}$$

Hence, the Givens QRD algorithm calculates $\boldsymbol{R}$ by applying (7) to $\boldsymbol{A}$, meanwhile it accumulates $\boldsymbol{Q}$ using (8).

Two critical observations can further simplify the established Givens QRD algorithm. First, it is important to realize that applying generalized Givens rotation

matrices preserves all entries in $\boldsymbol{A}$ except for the ones in the row i and k. In other words, the algorithm can apply (4) to only rows i and k, isolated from $\boldsymbol{A}$, while it leaves the rest of $\boldsymbol{A}$ intact. This simplification is also applicable to the computation of $\boldsymbol{Q}$. Second, by applying Givens rotations in a column-wise and top-to-bottom manner, the dimension of a QRD problem is diminishing during the process. For instance, in the process of triangularizing a 3×3 matrix $\boldsymbol{A}$, once the first column is completely normalized, the remaining QRD problem becomes the triangularization of the 2×2 lower-right sub-matrix of $\boldsymbol{A}$. Both simplifications significantly reduce the number of processing operations compared to using the generalized Givens rotation matrices, and the improved Givens QRD algorithm is formalized in Table 1.

Table 1: QRD Algorithm using Givens Rotations

(1)	$\boldsymbol{Q} = \boldsymbol{I}_m$
(2)	for $i = 1 : n - 1$
(3)	for $k = i + 1 : m$
(4)	$c = \boldsymbol{A}(i,i)/\sqrt{\boldsymbol{A}(k,i)^2 + \boldsymbol{A}(i,i)^2}$
(5)	$s = \boldsymbol{A}(k,i)/\sqrt{\boldsymbol{A}(k,i)^2 + \boldsymbol{A}(i,i)^2}$
(6)	$\boldsymbol{A}([i,k], i:n) = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \boldsymbol{A}([i,k], i:n)$
(7)	$\boldsymbol{Q}([i,k], 1:n) = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}^T \boldsymbol{Q}([i,k], 1:n)$
(8)	end
(9)	end
(10)	$\boldsymbol{R} = \boldsymbol{A}$

According to [5], the complexity of the algorithm is $3n^2(m - n/3)$ floating-point operations (flops) for an $m \times n$ matrix without accumulating $\boldsymbol{Q}$. Since only square matrices are considered in this study, the complexity can be calculated as $2n^3$ flops without $\boldsymbol{Q}$, and $5n^3$ with $\boldsymbol{Q}$.

2.2.2 Complex-valued QRD Algorithm

A complex-valued QRD algorithm is an extension of the real-valued QRD algorithm using complex Givens rotation matrices. For a 2×2 complex-valued matrix $\boldsymbol{H}$:

$$\boldsymbol{H} = \begin{bmatrix} Ae^{j\theta_a} & Ce^{j\theta_c} \\ Be^{j\theta_b} & De^{j\theta_d} \end{bmatrix}, \tag{9}$$

the complex Givens rotation matrix is defined as:

$$\boldsymbol{G}_c = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1)e^{j\theta_2} \\ \sin(\theta_1)e^{-j\theta_2} & \cos(\theta_1) \end{bmatrix}, \tag{10}$$

where $\theta_1 = -\arctan(B/A)$, and $\theta_2 = \theta_b - \theta_a$. Applying $\boldsymbol{G}_c$ to $\boldsymbol{H}$ will annihilate $\boldsymbol{H}(2,1)$, but the diagonal entries remain complex after the rotation, shown as follows:

$$\begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1)e^{-j\theta_2} \\ \sin(\theta_1)e^{j\theta_2} & \cos(\theta_1) \end{bmatrix} \cdot \begin{bmatrix} Ae^{\theta_a} & Ce^{j\theta_c} \\ Be^{\theta_b} & De^{j\theta_d} \end{bmatrix} = \begin{bmatrix} A'e^{j\theta'_a} & C'e^{j\theta'_c} \\ 0 & D'e^{j\theta'_d} \end{bmatrix}. \tag{11}$$

Therefore, for a complex-valued QRD algorithm using (10), additional rotations are required to make each complex-valued diagonal entry real-valued. The processing overhead increases linearly with the dimension of matrices.

In order to minimize the overhead, [10] proposed an alternative rotation matrix that equivalently annihilates one sub-diagonal entry as well as making the top-diagonal entry real-valued. It is accomplished by introducing a third angle into the rotation matrix, namely a three angle complex rotation (TACR). Again, considering the matrix in (9), a TACR can be constructed as follows:

$$\boldsymbol{G}_{TACR} = \begin{bmatrix} \cos(\theta_1)e^{-j\theta_a} & -\sin(\theta_1)e^{-j\theta_b} \\ \sin(\theta_1)e^{-j\theta_a} & \cos(\theta_1)e^{-j\theta_b} \end{bmatrix} \tag{12}$$

$$= \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) \end{bmatrix} \begin{bmatrix} e^{-j\theta_a} & 0 \\ 0 & e^{-j\theta_b} \end{bmatrix}, \tag{13}$$

where $\theta_1 = -\arctan(B/A)$. The matrix-product form in (13) reveals that a TACR is a two-stage process. The first-stage purpose is to convert leading-column entries into real-valued elements as:

$$\begin{bmatrix} e^{-j\theta_a} & 0 \\ 0 & e^{-j\theta_b} \end{bmatrix} \cdot \begin{bmatrix} Ae^{\theta_a} & Ce^{j\theta_c} \\ Be^{\theta_b} & De^{j\theta_d} \end{bmatrix} = \begin{bmatrix} A & Ce^{j(\theta_c-\theta_a)} \\ B & De^{j(\theta_d-\theta_b)} \end{bmatrix}. \tag{14}$$

Then, a real-valued Givens rotation matrix is applied in the second stage to introduce a zero to the sub-diagonal position:

$$\begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) \end{bmatrix} \cdot \begin{bmatrix} A & Ce^{j(\theta_c-\theta_a)} \\ B & De^{j(\theta_d-\theta_b)} \end{bmatrix} = \begin{bmatrix} \sqrt{A^2+B^2} & C'e^{j\theta'_c} \\ 0 & D'e^{j\theta'_d} \end{bmatrix}. \tag{15}$$

As evidently seen in (15), the top-diagonal entry becomes real while the bottom-diagonal entry remains complex. It implies that for a complex-valued QRD algorithm using TACRs, one more rotation is required to make the bottom diagonal entry become real-valued. Although the overhead still exists, the number of additional operations remain constant, which is a major improvement in terms of complexity compared to the algorithm using (10).

The way to compute $\boldsymbol{Q}$ using TACRs remains similar to using Givens rotation matrices. The difference is that TACRs are complex-valued and unitary, i.e., $\boldsymbol{G}_{TACR}^{-1} = \boldsymbol{G}_{TACR}^{\mathcal{H}}$, so the accumulation of $\boldsymbol{Q}$ requires Hermitian operations in the complex-valued case:

$$\boldsymbol{Q} = (\boldsymbol{G}_N^{TACR} \cdots \boldsymbol{G}_2^{TACR} \boldsymbol{G}_1^{TACR})^{\mathcal{H}} = (\boldsymbol{G}_1^{TACR^{\mathcal{H}}} \boldsymbol{G}_2^{TACR^{\mathcal{H}}} \cdots \boldsymbol{G}_N^{TACR^{\mathcal{H}}}). \tag{16}$$

The QRD algorithm using TACRs is formalized in Table 2. The complexity of the algorithm for an $n \times n$ complex matrix increases to approximately $9n^3$ flops without computing $\boldsymbol{Q}$, and approximately $18n^3$ flops with $\boldsymbol{Q}$.

2.2.3 Sorted-QRD Algorithm

A sorted-QRD algorithm is an extension of any classic QRD algorithm created by inserting a column ordering process before each column is normalized. Depending on

Table 2: Complex-valued QRD Algorithm using TACRs

(1)	$\boldsymbol{Q} = \boldsymbol{I}_n$
(2)	for $i = 1 : n-1$
(3)	for $k = i+1 : n$
(4)	$c = \boldsymbol{H}(i,i)/\sqrt{\lvert\boldsymbol{H}(k,i)\rvert^2 + \lvert\boldsymbol{H}(i,i)\rvert^2}$
(5)	$s = \boldsymbol{H}(k,i)/\sqrt{\lvert\boldsymbol{H}(k,i)\rvert^2 + \lvert\boldsymbol{H}(i,i)\rvert^2}$
(6)	$\boldsymbol{G}_{TACR} = \begin{bmatrix} c \cdot e^{-j\theta_{\boldsymbol{H}(i,i)}} & -s \cdot e^{-j\theta_{\boldsymbol{H}(k,i)}} \\ s \cdot e^{-j\theta_{\boldsymbol{H}(i,i)}} & c \cdot e^{-j\theta_{\boldsymbol{H}(k,i)}} \end{bmatrix}$
(7)	$\boldsymbol{H}([i,k], i:n) = \boldsymbol{G}_{TACR} \cdot \boldsymbol{H}([i,k], i:n)$
(8)	$\boldsymbol{Q}([i,k], 1:n) = \boldsymbol{G}^{\mathcal{H}}_{TACR} \cdot \boldsymbol{Q}([i,k], 1:n)$
(9)	end
(10)	end
(11)	$\boldsymbol{H}(n,n) = e^{-j\theta_{\boldsymbol{H}(n,n)}} \cdot \boldsymbol{H}(n,n)$
(12)	$\boldsymbol{Q}(n, 1:n) = e^{j\theta_{\boldsymbol{H}(n,n)}} \cdot \boldsymbol{Q}(n, 1:n)$
(13)	$\boldsymbol{R} = \boldsymbol{H}$

the need of applications, the ordering process searches for either a maximum-norm column among the remaining columns in the algorithm, or a minimum-norm column, and then it permutes such a column to the leading-column position. Intuitively, the process can be also described as pushing the potentially-largest "mass" or "signal power" into either corner of the matrix diagonal.

There have been different motivations to apply sorted-QRD algorithms. In numerical linear algebra, the sorted-QRD algorithm can be adapted for preventing the QRD algorithm from failing when processing rank deficient matrices. In this case, ordering processes target the maximum-norm columns. In contrast, [22], which is among one of the earliest recognition to use sorted-QRD for MIMO communication channel decoding, shows that by minimizing the diagonal entry at each decomposition step, a sorted-QRD algorithm can effectively combat error propagation and improve the efficiency of channel decoding. Hence, the sorted-QRD algorithm using TACRs is formalized based on conclusions in [22, 23], and it is shown in Table 3.

The only differences between Table 2 and Table 3 are Line 3-4, which are the column ordering step and the permutation step respectively. The ordering processes using squared Frobenius norm values add approximately $3n^3$ flops into the algorithm

Table 3: Complex-valued Sorted-QRD Algorithm using TACRs

(1)	$\boldsymbol{Q} = \boldsymbol{I}_n$, $\boldsymbol{p} = [1, 2, \ldots, n]$
(2)	for $i = 1 : n - 1$
(3)	$m_i = \text{argmin}_{c=\{i,i+1,\ldots,n\}} \|\boldsymbol{H}(:, c)\|_{\mathcal{F}}^2$
(4)	exchange column i with column m_i in $\boldsymbol{H}$, $\boldsymbol{Q}$ and $\boldsymbol{p}$
(5)	for $k = i + 1 : n$
(6)	$c = \boldsymbol{H}(i, i)/\sqrt{\|\boldsymbol{H}(k, i)\|^2 + \|\boldsymbol{H}(i, i)\|^2}$
(7)	$s = \boldsymbol{H}(k, i)/\sqrt{\|\boldsymbol{H}(k, i)\|^2 + \|\boldsymbol{H}(i, i)\|^2}$
(8)	$\boldsymbol{G}_{TACR} = \begin{bmatrix} c \cdot e^{-j\theta_{\boldsymbol{H}(i,i)}} & -s \cdot e^{-j\theta_{\boldsymbol{H}(k,i)}} \\ s \cdot e^{-j\theta_{\boldsymbol{H}(i,i)}} & c \cdot e^{-j\theta_{\boldsymbol{H}(k,i)}} \end{bmatrix}$
(9)	$\boldsymbol{H}([i, k], i : n) = \boldsymbol{G}_{TACR} \cdot \boldsymbol{H}([i, k], i : n)$
(10)	$\boldsymbol{Q}([i, k], 1 : n) = \boldsymbol{G}_{TACR}^{\mathcal{H}} \cdot \boldsymbol{Q}([i, k], 1 : n)$
(11)	end
(12)	end
(13)	$\boldsymbol{H}(n, n) = e^{-j\theta_{\boldsymbol{H}(n,n)}} \cdot \boldsymbol{H}(n, n)$
(14)	$\boldsymbol{Q}(n, 1 : n) = e^{j\theta_{\boldsymbol{H}(n,n)}} \cdot \boldsymbol{Q}(n, 1 : n)$
(15)	$\boldsymbol{R} = \boldsymbol{H}$

complexity. Hence, the total number of the operations for the sorted-QRD algorithm is approximately $21n^3$ flops. In addition, sorting processes can hinder the subsequent QRD computations, which will have a negative impact on the latency of implementations.

2.3 Existing QRD Implementations

A considerable amount of research has developed many complex-valued QRD implementations for MIMO systems. The different approaches that have been investigated can be roughly summarized into three different categories.

- The first category is to solve QRD problems using Givens rotation matrices, which have been explained in the previous sections. The CORDIC algorithm is often collaboratively used in this approach to realize rotations in fixed-point arithmetic [2, 6, 7, 8, 10, 15, 20]. Some studies further elaborate this approach by using multi-dimensional rotations [7, 12].

- In the second category, algorithms first rearrange $n \times n$ complex-valued matrices

to obtain $2n \times 2n$ real-valued matrices. Meanwhile, they transform real numbers into the logarithmic domain so that multiplications and divisions can be equivalently replaced by additions and subtractions [14]. The transformation ensures the numerical stability of fixed-point arithmetic.

- The last category directly deploys the modified Gram-Schmidt algorithm. The challenge in this approach is to maintain fixed-point arithmetic stability in inverse and square-root operations. In [13], dedicated inverse and square-root functional units solve the problem, while inverse and square-root operations utilize a reduced floating-point number representation in [9] to obtain precise results.

Besides these three categories, a few recursive or interpolation-based algorithms were used in [3, 16], but they might suffer from intolerantly-high latency for high-dimension matrices.

Regarding different types of hardware architecture, a triangular systolic array is a standard choice for implementing QRD processors in order to meet the high throughput requirements of MIMO systems [2, 10, 14]. However, the hardware cost of a triangular systolic array grows quadratically with the dimension of matrices. The high hardware cost has motivated researchers to find alternatives in order to reduce the cost without significantly degrading the throughput of implementations. In [15, 20], researchers used an array of processing units. In theory, the hardware cost in the array type architecture has a linearly-growing relation with the matrices dimension, which is an improvement from using a triangular systolic array. However, the throughput performance in [15, 20] is worse than the ones reported in [6, 7, 12], all of which use triangular systolic arrays. On the other hand, some studies designed a single-core architecture and successfully fulfill the throughput requirements with minimum hardware cost [8, 9].

CHAPTER III

COMPLEX-VALUED QRD WITH FIXED-POINT ARITHMETIC

Chapter 2 has explained the basic techniques and algorithms using TACRs to solve complex-valued QRD problems. The next challenge is to adapt these techniques for implementation using finite-precision, fixed-point techniques. This chapter introduces the classic CORDIC algorithm that allows one to conduct Givens rotations using fixed-point arithmetic. The proposed algorithm is established based on modified CORDIC-based Givens rotations.

3.1 Classic CORDIC Algorithm

CORDIC arithmetic was first described in 1959 by Jack E. Volder [18, 19]. It is a shifting-and-adding type of algorithm that allows one to iteratively compute a wide range of elementary functions, such as multiplications, divisions, logarithms, exponentials, and many others using fixed-point arithmetic operations [11]. Many applications have employed the CORDIC algorithm, not only because it is cost-friendly for hardware implementations compared to the ones using multipliers and look-up tables, but also because the numerical stability of the algorithm is crucial for fixed-point applications. However, the iterative nature of the CORDIC algorithm introduces a data-dependent latency.

This study uses the CORDIC algorithm to compute Givens rotations for real-valued vectors, denoted by $[x, y]^{\mathcal{T}}$. First, it is important to realize that a Givens

rotation can be broken into micro-rotations to achieve vector normalization:

$$\boldsymbol{G}(\theta) \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \boldsymbol{G}_N(\theta_N) \cdot \boldsymbol{G}_{N-1}(\theta_{N-1}) \cdots \boldsymbol{G}_1(\theta_0) \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sqrt{x^2+y^2} \\ 0 \end{bmatrix}, \tag{17}$$

where $\theta = \sum_{i=0}^{N} \theta_i = -\arctan(y/x)$. Meanwhile, by taking a common factor out of the Givens rotation matrix in (4), as follows:

$$\boldsymbol{G} = \cos(\theta) \begin{bmatrix} 1 & -\tan(\theta) \\ \tan(\theta) & 1 \end{bmatrix} = K \cdot \boldsymbol{R}, \tag{18}$$

(17) can be further rearranged into the following form:

$$\boldsymbol{G}(\theta) = \prod_{i=0}^{N} \cos(\theta_i) \cdot \prod_{i=0}^{N} \begin{bmatrix} 1 & -\tan(\theta_i) \\ \tan(\theta_i) & 1 \end{bmatrix} = K \cdot \prod_{i=0}^{N} \boldsymbol{R}_i. \tag{19}$$

Bassed on (19), the CORDIC algorithm further constrains the micro-rotation angles by a set of pre-defined angles: $\theta_i = \{\arctan(2^{-i}) | 0 \leq i \leq N\}$, called arc tangent radix (ATR) [18]. The definition of ATR, which makes $\tan(\theta_i) = 2^{-i}$, conveniently leads to a significant hardware simplification, which shifting operations equivalently replace multiplications in fixed-point arithmetic. Hence, the realization of micro-rotations in hardware is simply a fixed-point shifting function followed by either an addition or a subtraction, shown as follows, where $[x_i, y_i]^{\mathcal{T}}$ is the vector after i micro-rotations:

$$\boldsymbol{R}_i \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} 1 & -\sigma_i \cdot 2^{-i} \\ \sigma_i \cdot 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} x_i - \sigma_i \cdot 2^{-i} y_i \\ y_i + \sigma_i \cdot 2^{-i} x_i \end{bmatrix}. \tag{20}$$

The variable σ_i in (20), controls the directions of micro-rotations, and it also ensures that the sum of micro-rotation angles converges to the original rotation angle after $N+1$ iterations:

$$\theta = \sum_{i=0}^{N} \sigma_i \theta_i, \;\; \sigma_i = \begin{cases} -1 & \text{if } y_{i-1} > 0, \\ +1 & \text{if } y_{i-1} \leq 0. \end{cases} \tag{21}$$

It is proved that the sum in (21) indeed converges and it has a convergence range between -1.74329 and 1.74329 radians [11]. Thereby, the rotation angle θ can be calculated by (21) while the vector is being normalized. However, the convergence range covers only the first and the fourth quadrant. For vectors that lie within the second or the third quadrant, one extra rotation has to be performed before micro-rotations take place. The extra rotation, also referred as a correction rotation, simply moves these vectors into convergence range by turning $\pm\pi/2$ radians:

$$\begin{aligned} x_0 &= -\sigma_{-1} \cdot y \\ y_0 &= \sigma_{-1} \cdot x \\ \theta_0 &= -\sigma_{-1} \cdot \pi/2. \end{aligned} \tag{22}$$

On the other hand, the scale-factor K in (19) can be rewritten as the following form using trigonometric identities:

$$K = \prod_{i=0}^{N} \cos(\theta_i) = \prod_{i=0}^{N} 1/\sqrt{1 + 2^{-2i}}. \tag{23}$$

Taking advantage of the fact that K does not depend on σ_i, the value of K can be pre-determined as long as the number of micro-rotations is known, which fortunately is the case for most CORDIC applications. Therefore, K becomes an input constant to the algorithm, and it scales the outputs of micro-rotations at the end of the algorithm.

Finally, vector normalization using the CORDIC algorithm, also referred as the CORDIC vectoring mode [18], is formalized in Table 4. Moreover, it is not hard to anticipate that with some modifications, the CORDIC algorithm can be also used for vector rotations by given angles, also known as the CORDIC rotation mode [18]. The only significant change to switch from the vectoring mode to the rotation mode is that the sign of θ_{i-1} determines the value of σ_i instead of y_{i-1}, as follows:

$$\sigma_i = \begin{cases} +1 & \text{if } \theta_{i-1} \geq 0, \\ -1 & \text{if } \theta_{i-1} < 0. \end{cases} \tag{24}$$

The algorithm of the CORDIC rotation mode is formalized in Table 5. In [11] it is also suggested that $(N + 1)$ CORDIC iterations are necessary to obtain N-bit output precision. It implies that if only one CORDIC iteration is performed in one clock cycle, a CORDIC-based vectoring or rotation will have a latency of $(N + 1)$ cycles without considering the correction rotation at the beginning and scale factor compensation at the end. Hence, the worst CORDIC algorithm latency is $(N + 3)$ clock cycles.

Table 4: The Algorithm of CORDIC Vectoring Mode

(1)	$[\ z,\ \theta\ ]$ = CORDIC_vectoring_mode(x, y, N, K)
(2)	$\sigma_{-1} = -sign(y)$
(3)	$x_0 = -\sigma_{-1} \cdot y$
(4)	$y_0 = \sigma_{-1} \cdot x$
(5)	$\theta_0 = \sigma_{-1} \cdot (\pi/2)$
(6)	for $i = 0 : N$
(7)	$\quad \sigma_i = -sign(y_0)$
(8)	$\quad x_{i+1} = x_i - \sigma_i \cdot 2^{-i} \cdot y_i$
(9)	$\quad y_{i+1} = y_i + \sigma_i \cdot 2^{-i} \cdot x_i$
(10)	$\quad \theta_{i+1} = \theta_i - \sigma_i \cdot \arctan(2^{-i})$
(11)	end
(12)	$z = x_N \cdot K$
(13)	$\theta = \theta_N$

Table 5: The Algorithm of CORDIC Rotation Mode

(1)	$[\ x',\ y']$ = CORDIC_rotation_mode(x, y, θ, N, K)
(2)	$\sigma_{-1} = sign(\theta)$
(3)	$x_0 = -\sigma_{-1} \cdot y$
(4)	$y_0 = \sigma_{-1} \cdot x$
(5)	$\theta_0 = \theta - \sigma_{-1} \cdot (\pi/2)$
(6)	for $i = 0 : N$
(7)	$\quad \sigma_i = sign(\theta)$
(8)	$\quad x_{i+1} = x_i - \sigma_i \cdot 2^{-i} \cdot y_i$
(9)	$\quad y_{i+1} = y_i + \sigma_i \cdot 2^{-i} \cdot x_i$
(10)	$\quad \theta_{i+1} = \theta_i - \sigma_i \cdot \arctan(2^{-i})$
(11)	end
(12)	$x' = x_N \cdot K$
(13)	$y' = y_N \cdot K$

3.2 Application of the CORDIC Algorithm to Complex-valued QRDs

In order to use the CORDIC algorithm for solving QRD problems, it is important to understand that applying a Givens rotation to a 2×2 real-valued matrix, showed in (25), is a combination of first-column normalization and second-column rotation by the angle $\theta = -\arctan(b/a)$:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} a & c \\ b & d \end{bmatrix} = \begin{bmatrix} \sqrt{a^2+b^2} & c' \\ 0 & d' \end{bmatrix}. \tag{25}$$

Hence, (25) can be realized by a CORDIC vectoring-and-rotation sequence. First, the CORDIC vectoring mode normalizes the first column and calculates the rotation angle. Second, the CORDIC rotation mode uses the angle to rotate the second column. The vectoring-and-rotation sequence, depicted in Fig. 2, servers as a fundamental building block for a CORDIC-based QRD algorithm.

Such a building block can also implement the two-stage TACR operation, shown again as follows, where bold lowercase letters represent complex numbers:

$$\boldsymbol{G}_{TACR} \cdot \boldsymbol{H} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) \end{bmatrix} \begin{bmatrix} e^{-j\theta_a} & 0 \\ 0 & e^{-j\theta_b} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{a} & \boldsymbol{c} \\ \boldsymbol{b} & \boldsymbol{d} \end{bmatrix} \tag{26}$$

$$= \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) \end{bmatrix} \begin{bmatrix} |\boldsymbol{a}| & \dot{\boldsymbol{c}} \\ |\boldsymbol{b}| & \dot{\boldsymbol{d}} \end{bmatrix} \tag{27}$$

$$= \begin{bmatrix} \sqrt{|\boldsymbol{a}|^2+|\boldsymbol{b}|^2} & \ddot{\boldsymbol{c}} \\ 0 & \ddot{\boldsymbol{d}} \end{bmatrix}. \tag{28}$$

Two vectoring-and-rotation sequences, operating in parallel, can realize TACR operations from (26) to (28), illustrated in Fig. 3. Notice that two TACR stages can share the two sequences to maximize resource utilization.

Since the QRD algorithm in Table 2 is essentially the repetitive utilization of TACRs, the structure in Fig. 3 becomes the prototype of a QRD processor. One

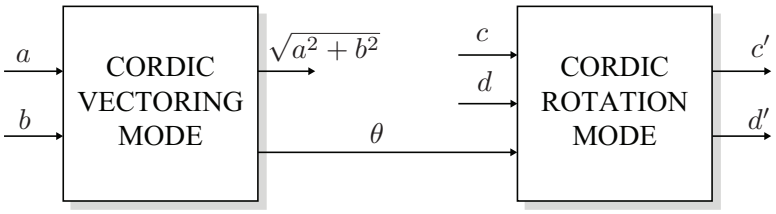
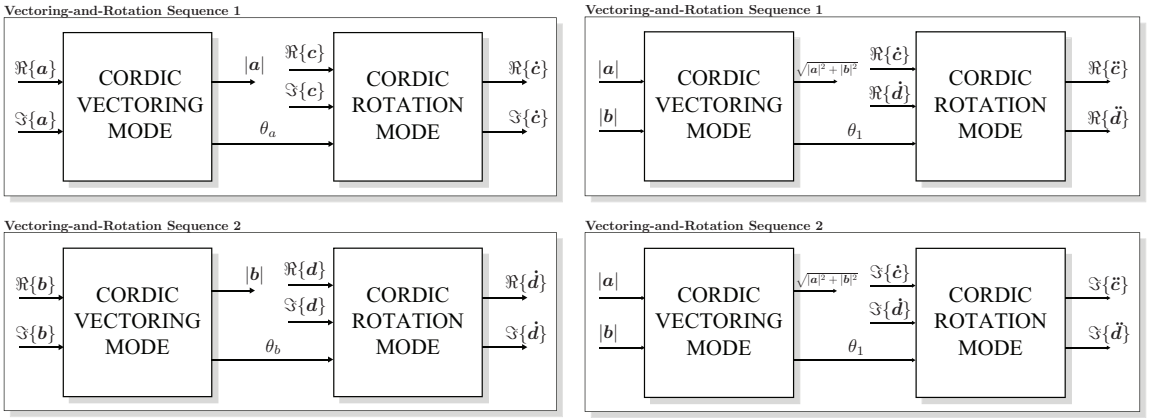


Figure 2: Data flow diagram of a CORDIC-based vectoring-and-rotation sequence for triangularizing a real-valued matrix $\left[\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right]$, where $\theta = -\arctan(b/a)$.

can simply use more CORDIC vectoring-and-rotation sequences to conduct QRDs in higher dimension. In order to maximize resource utilization, two vectoring processing-elements (PEs), each of which carries two CORDIC vectoring modes, are sufficient for any size of matrices, because TACRs are 2-dimensional and vector normalization only takes place in the leading column. On the other hand, the number of rotation PEs, which execute the CORDIC rotation mode, has to linearly increase with the matrix dimension so that they can rotate the rest of matrix columns in parallel. The linear relation can be better illustrated by Fig. 4. Complex-valued $n \times n$ matrices need at most $n-1$ rotation PEs for computing $\boldsymbol{R}$, and they need another n rotation PEs for $\boldsymbol{Q}$ accumulation. Due to the dimension diminishing effect of input matrices, explained



(a) Stage 1 **(b)** Stage 2

Figure 3: Data flow diagram of CORDIC-based vectoring and rotation sequences for triangularizing complex-valued matrix $\left[\begin{smallmatrix} \boldsymbol{a} & \boldsymbol{c} \\ \boldsymbol{b} & \boldsymbol{d} \end{smallmatrix}\right]$, where $\theta_1 = -\arctan(|\boldsymbol{b}|/|\boldsymbol{a}|)$.

in Sec. 2.2.1, the utilization rate of rotation PEs for $\boldsymbol{R}$ is declining as the QRD algorithm progresses. Whereas, rotation PEs for $\boldsymbol{Q}$ stay fully-engaged throughout

the algorithm.

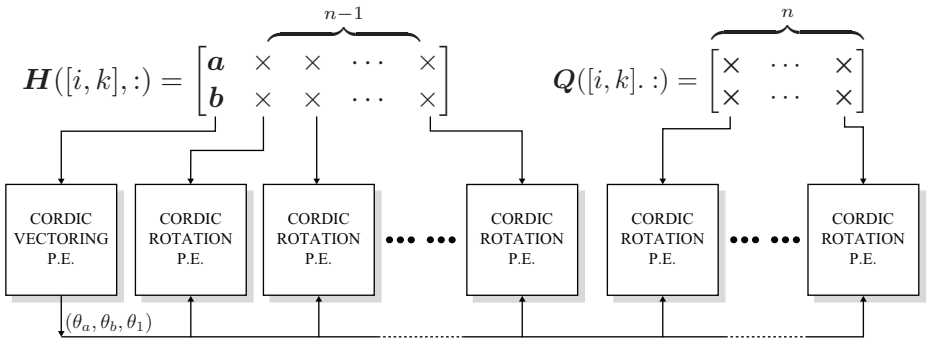


Figure 4: A processing element (PE) array constructed by vectoring and rotation PEs. The $1 \times 2n$ PE array is capable of performing TACRs for row i and row k from complex-valued matrices $\boldsymbol{H}$, where $1 \leq i \leq n$, $1 \leq k \leq n$ and $i \neq k$. Meanwhile, the PE array rotates the corresponding rows in $\boldsymbol{Q}$. Each PE contains two CORDIC vectoring/rotation units.

Re-examining Table 2, one should realize that a PE array depicted in Fig. 4 can execute all computations in the algorithm, including Line 4-6 and Line 11-12. Assuming computation data is instantly available to the PEs, the complex-valued QRD algorithm for $n \times n$ matrices requires $n(n-1)/2$ TACR operations and an additional Givens rotation at the end. With the worst CORDIC latency, the latency of one TACR operation is $3(N+3)$ clock cycles, while a Givens rotation takes $2(N+3)$ cycles. Hence, for the CORDIC-based complex-valued QRD algorithm, the theoretical latency can be calculated as:

$$latency = \frac{3n(n-1)+4}{2} \cdot (N+3). \tag{29}$$

3.3 Proposed Algorithm

The proposed algorithm offers three important modifications of the CORDIC algorithm that improves the latency in (29), and the dissimilarity of CORDIC-based QRD algorithm. The first key observation regarding the vectoring-and-rotation sequence, depicted in Fig. 2, is the data dependency between the vectoring mode and rotation mode. The rotation mode cannot start rotations until the vectoring mode finishes

calculating the rotation angle, θ. The dependency can potentially lead to a latency of $3(N+3)$ clock cycles for one TACR at worst. However, it is critical to realize that the rotation angle calculated in vectoring mode is a linear combination of a basis. The basis, or ATR is pre-defined as $\theta_i = \{\arctan(2^{-i})|0 \leq i \leq N\}$, and the coefficients are control variables, σ_i, previously-defined in (21). The properties of linearity imply that an one-to-one mapping exists between an angle and a set of coefficients. In other words, a set of control variable values is sufficient to represent a unique angle without explicit computations. Since the rotation mode decomposes the angle with the same set of basis, it is concluded that a set of control signals $\{\sigma_i|0 \leq i \leq N\}$ can replace the angle θ to be transferred from the vectoring mode to the rotation mode. The replacement not only is equivalent, but it also eliminates the hardware cost to have a third addition-and-subtraction unit to calculate θ with memories for ATR constants in both modes.

Moreover, within the rotation mode, the decomposition order of an angle is the same as the order with which the angle is calculated in the vectoring mode, i.e., from $i = 0$ to $i = N$. Hence, the vectoring mode can pass a single-digit control signal, σ_i, to the rotation mode as soon as it is determined. This modification minimizes the delay introduced by the data dependency in a vectoring-and-rotation sequence, and allows vector normalization and vector rotations to operate in parallel. This parallelized vectoring-and-rotation sequence is illustrated in Fig. 5. The rotation mode, in this case, works as a slave to the vectoring mode, because it cannot operate without the control signal produced by the vectoring mode. Hence, a CORDIC master-slave architecture more-appropriately describes the relation between the two modes, also labeled in Fig. 5. The new architecture can complete a Givens rotation with the worst latency of $(N + 3)$ clock cycles, which halves what vectoring-and-rotation sequence consumes. It also improves the latency of a TACR operation to $2(N+3)$ clock cycles.

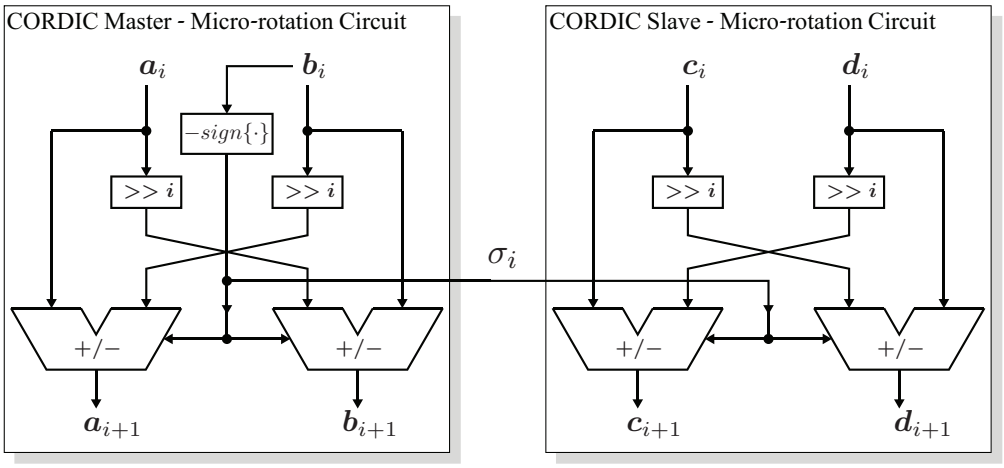


Figure 5: The micro-rotation circuit of the CORDIC vectoring-and-rotation sequence using master-slave architecture. The control signal σ_i is determined by the sign of b_i in the CORDIC master, and it is connected with the slave to control micro-rotation directions.

Eventually, the overall latency for a complex-valued QRD can be re-formulated as

$$latency = (n(n-1)+1) \cdot (N+3), \tag{30}$$

which is less than two third of the latency in (29).

The second observation is that two clock cycles out of the worst TACR latency are used for correction rotations, which are not mandatory because input vectors in the first or second quadrant do not require correction rotations. In other words, if a design conditionally conducts correction rotations, it can save two clock cycles when encountering those vectors. However, the conditional correction rotations are likely to require a sequential placement of a multiplexer and a fixed-point number complement circuit followed by the CORDIC iteration circuit in a single clock cycle. It slows down the clock frequency of the design without significantly reducing the latency. Instead, the proposed algorithm forces the correction rotation in the first TACR stage to be mandatory. In the second stage, the resulting vectors produced by the first stage are expected to be normalized, i.e., vectors with only positive x-components. Hence, the second stage no longer requires correction rotations, and resulting vectors from the first stage can immediately start second-stage CORDIC

iterations. The iteration circuit can remains intact by making the correction rotation a single-cycle operation. With this modification, each TACR operation takes one less clock cycle, and the overall latency is reduced by $n(n-1)/2$ clock cycles from (30), reformulated as follows:

$$latency = (n(n-1)+1) \cdot (N+3) - n(n-1)/2. \tag{31}$$

The third observation is that vector normalization has been unconditional for every input vector so far. It is possible that some of them do not require any normalization, because they could happen to only have positive x-axis components, or because they might have been normalized in the previous steps. Probabilistically, the first case is unlikely to occur; therefore, the focus is to exploit the second case. By revisiting QRD algorithms in Table 2, one should realize that while a full column is being normalized, only the first TACR requires normalization for both vector entries in the first stage. The observation can be better illustrated using a 3×2 input matrix as follows:

$$\boldsymbol{G}_2^{TACR}\boldsymbol{G}_1^{TACR}\boldsymbol{H} = \boldsymbol{G}_2^{TACR}\boldsymbol{G}_1^{TACR}\begin{bmatrix}\mathbb{C} & \mathbb{C}\\ \mathbb{C} & \mathbb{C}\\ \mathbb{C} & \mathbb{C}\end{bmatrix} = \boldsymbol{G}_2^{TACR}\begin{bmatrix}\mathbb{R} & \mathbb{C}\\ 0 & \mathbb{C}\\ \mathbb{C} & \mathbb{C}\end{bmatrix} = \begin{bmatrix}\mathbb{R} & \mathbb{C}\\ 0 & \mathbb{C}\\ 0 & \mathbb{C}\end{bmatrix}. \tag{32}$$

(32) clearly shows that when $\boldsymbol{G}_2^{TACR}$ is ready to apply, the top diagonal entry has become a real-valued number. Therefore, no CORDIC iteration is necessary for that particular entry during the first stage of $\boldsymbol{G}_2^{TACR}$. Bypassing the first stage iterations, unfortunately, does not reduce the overall latency because the bottom row of $\boldsymbol{H}$ is still complex-valued. However, the bypass feature improves the numerical dissimilarity of the CORDIC-based QRD algorithm. Fig. 6 shows the performance comparison between the algorithm in Table 2 and the proposed algorithm featured bypass for 4×4 complex-valued matrices. It is clear that the proposed algorithm has better numerical dissimilarity regardless of the number of fraction bits being used. The

numerical improvement is significant because more precise QRD results can contribute to better MIMO symbol detection performance.

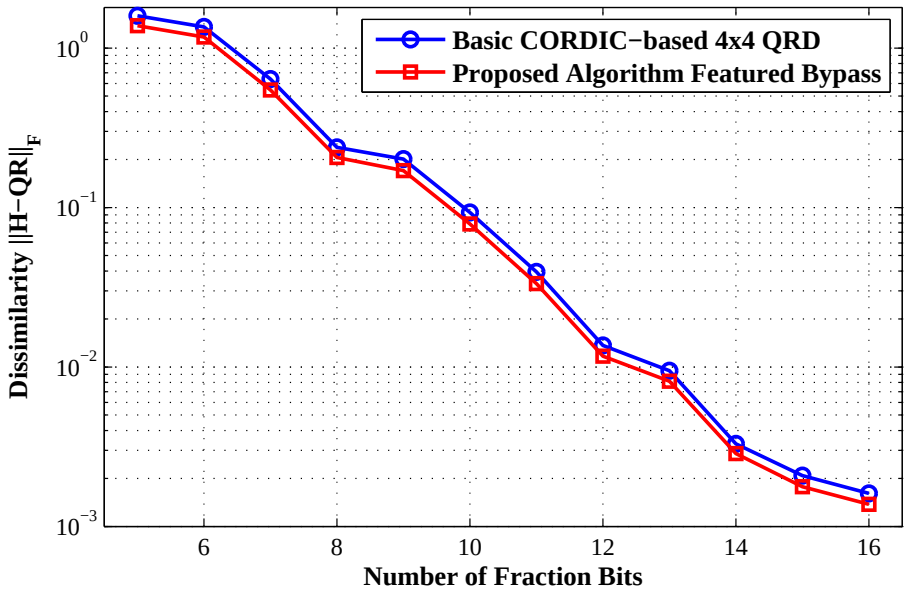


Figure 6: Numerical performance of the proposed algorithm featured bypass comparing to the basic CORDIC-based QRD algorithm for 4×4 matrices. For each point, $10,000$ QRD simulations were conducted with randomly-generated matrices. The simulations used 5-integer bits and a range of fraction bits, which spans from 5 to 16.

With the last improvement, the proposed algorithm is finalized in Table 6. For clarification purposes, Line 4-12 and Line 15-16 are written in mathematical representations. They can be fully replaced by the CORDIC algorithm with the proposed master-slave architecture, and the theoretical latency of the algorithm is derived in (31).

Table 6: Proposed Complex-valued QRD Algorithm using TACRs

(1)	$\boldsymbol{Q} = \boldsymbol{I}_n$
(2)	for $i = 1 : n-1$
(3)	for $k = i+1 : n$
(4)	$c = \boldsymbol{H}(i,i)/\sqrt{\lvert\boldsymbol{H}(k,i)\rvert^2 + \lvert\boldsymbol{H}(i,i)\rvert^2}$
(5)	$s = \boldsymbol{H}(k,i)/\sqrt{\lvert\boldsymbol{H}(k,i)\rvert^2 + \lvert\boldsymbol{H}(i,i)\rvert^2}$
(6)	if($k - i == 1$)
(7)	$\boldsymbol{G}_{TACR} = \begin{bmatrix} c \cdot e^{-j\theta_{\boldsymbol{H}(i,i)}} & -s \cdot e^{-j\theta_{\boldsymbol{H}(k,i)}} \\ s \cdot e^{-j\theta_{\boldsymbol{H}(i,i)}} & c \cdot e^{-j\theta_{\boldsymbol{H}(k,i)}} \end{bmatrix}$
(8)	else
(9)	$\boldsymbol{G}_{TACR} = \begin{bmatrix} c & -s \cdot e^{-j\theta_{\boldsymbol{H}(k,i)}} \\ s & c \cdot e^{-j\theta_{\boldsymbol{H}(k,i)}} \end{bmatrix}$
(10)	end
(11)	$\boldsymbol{H}([i,k], i:n) = \boldsymbol{G}_{TACR}\boldsymbol{H}([i,k], i:n)$
(12)	$\boldsymbol{Q}([i,k], 1:n) = \boldsymbol{G}^{\mathcal{H}}_{TACR}\boldsymbol{Q}([i,k], 1:n)$
(13)	end
(14)	end
(15)	$\boldsymbol{H}(n,n) = e^{-j\theta_{\boldsymbol{H}(n,n)}} \cdot \boldsymbol{H}(n,n)$
(16)	$\boldsymbol{Q}(n, 1:n) = e^{j\theta_{\boldsymbol{H}(n,n)}} \cdot \boldsymbol{Q}(n, 1:n)$
(17)	$\boldsymbol{R} = \boldsymbol{H}$

CHAPTER IV

HARDWARE REALIZATION

The previous chapter proposed a latency-and-precision-improved CORDIC-based QRD algorithm. In this chapter, the algorithm is realized by a scalable complex-valued QRD architecture, designed for Xilinx FPGA platforms. The overall structure is first introduced, followed by a discussion of design scalability. Next, the functionality of key components, including PE master and slaves, data transceivers and a novel sorting circuit, is explained. Finally, a design trade-off is introduced to potentially improve system throughput with higher hardware cost.

4.1 Proposed Architecture and Design Scalability

The challenge to design a fully-functional QRD processor on FPGAs is to efficiently utilize available hardware resources and to also achieve theoretical latency in (31). Without knowing the platform limitations, it is possible that the architecture could produce poor timing performance and large area utilization. A crucial limitation to QRD designs on Xilinx FPGAs comes from on-chip memories. One of the most efficient ways to have data stored and retrieved on Xilinx FPGAs is to use on-ship block random-access memories (RAMs). Most of Xilinx FPGA platforms support block RAMs that can handle only a single data transaction per clock cycle. A single PE, however, processes two complex-valued entries from input matrices in parallel, previously shown in Fig. 3. In other words, one PE requires two data transactions in one clock cycle. Furthermore, an array of PEs requires even greater data transfer bandwidth that linearly grows with the dimension of the array. The high bandwidth demand can be restricted by pipelining, illustrated in Fig. 7. Even though the pipelined structure constrains the bandwidth requirement to two complex-valued entries per

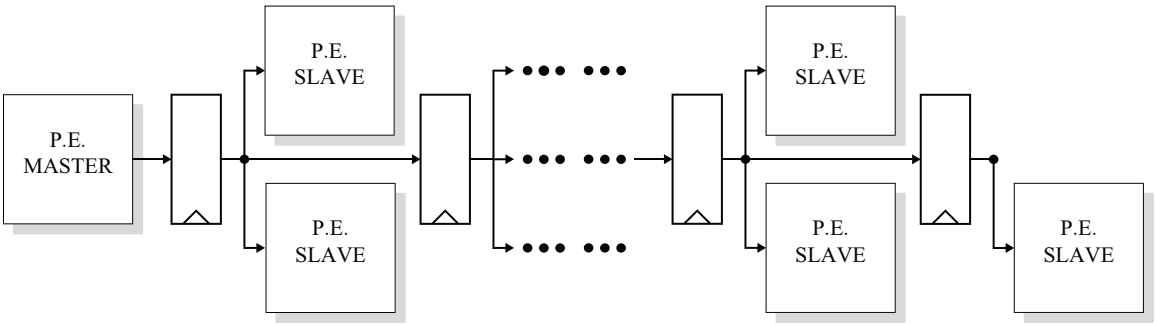


Figure 7: A pipelined PE array constructed by master-slave architecture. The control signals are generated in the PE master and registered before passed to PE slaves. The PE slaves in the top row are used for processing $\boldsymbol{R}$, while the ones at the bottom are for $\boldsymbol{Q}$.

clock cycle for any size of matrices, the 2-to-1 bandwidth ratio between pipelined PE array and block RAMs implies that computations may have to be stalled occasionally until all requested data arrives. In addition, the proposed architecture uses separate block RAMs for input and output matrices. The reason for using separate input and output memories is twofold: first, data storing bandwidth can be doubled; second, a new matrix can be introduced while the previously-processed matrix is being retrieved from the processor. Hence, four separate block RAMs are required in the system level, and they are used for storing an input matrix $\boldsymbol{H}$, an identity matrix $\boldsymbol{I}$, and output matrices $\boldsymbol{R}$ and $\boldsymbol{Q}$ respectively.

The proposed system-level architecture of a 4×4 complex-valued QRD processor is depicted in Fig. 8. Besides a PE array and memories, the data transceivers for $\boldsymbol{R}$ and $\boldsymbol{Q}$ are another two crucial components of the system. Data transceivers transfer the data from memories to PEs and minimize the computation stalls caused by the bandwidth gap between the PE array and memories. Moreover, the sorting circuit, embedded in the matrix $\boldsymbol{R}$ transceiver, and a block RAM $\boldsymbol{p}$ containing column indices are intended only for a sorted-QRD system.

The scalability of the proposed architecture is highly flexible due to two factors. First, the architecture can be fully parameterized from the top level to the bottom level based on matrices dimension, fixed-point data precision and number of CORDIC

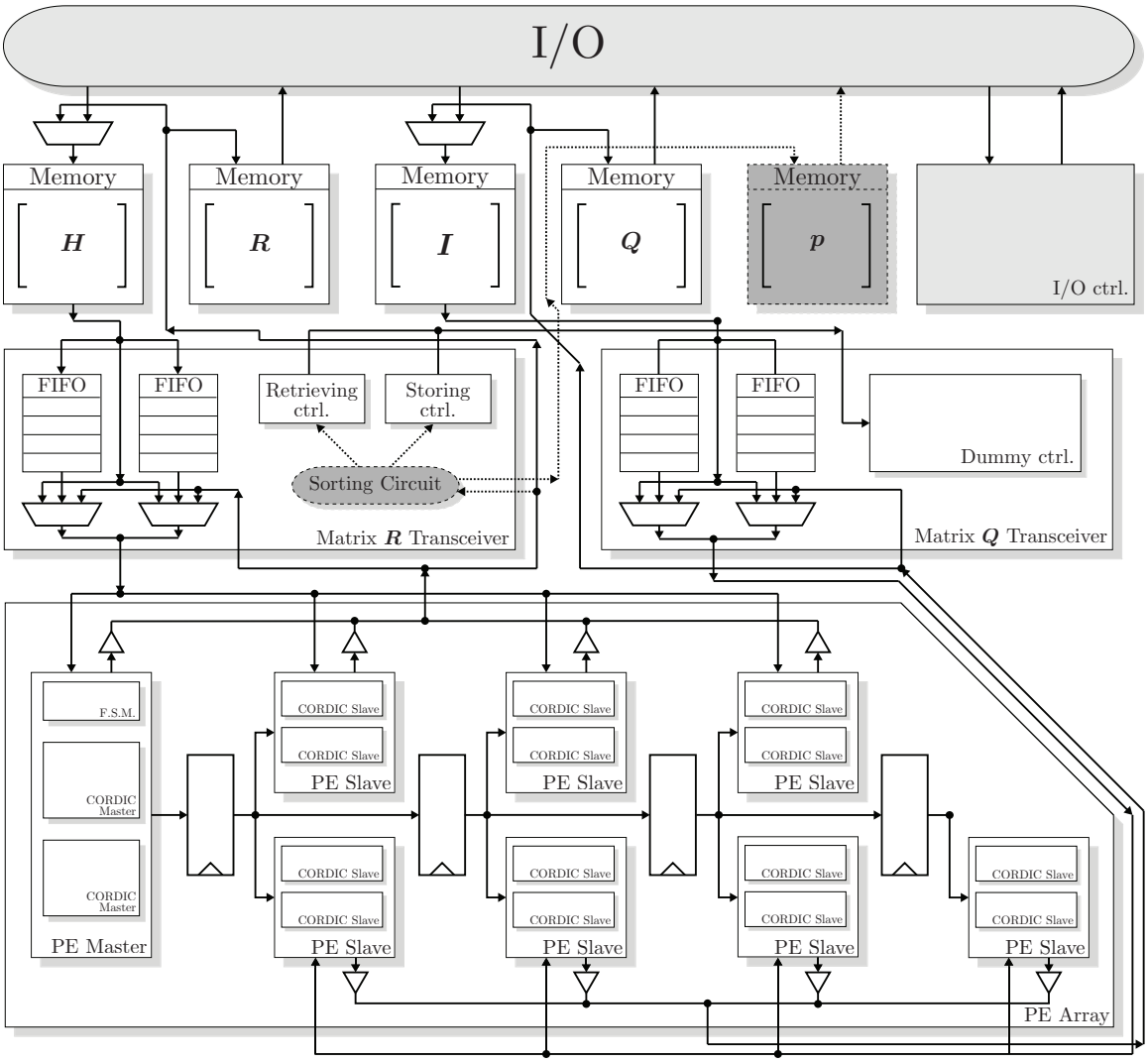


Figure 8: Proposed system level architecture for 4×4 complex-valued QRD and sorted-QRD. The sorting circuit and RAM $\boldsymbol{p}$ are only instantiated for sorted-QRD architecture only, while the rest of components are shared for both.

micro-rotations. More importantly, modern synthesis tools support the synthesis of a parametric register-transfer level (RTL) design, so the tools can correctly translate the proposed architecture into a lower level abstraction for place-and-route (PAR). By only changing parameters and constant values in the RTL design, the system can be scaled to solve different sizes of QRD problems after synthesis and PAR processes. Second, the proposed architecture calculates memories addresses in runtime. Column and row pointers are used to compute addresses for both retrieving and storing data.

It may not be the most efficient to have addresses calculated in runtime, but it avoids manual adjustments for addresses when the architecture is scaled.

4.2 PE Master and Slaves

PE master and slaves are computation cores in the proposed QRD architecture. The PE master consists of two CORDIC masters and it can normalize 2-dimension complex-valued vectors. Meanwhile, the PE master sends the control signals, including σ_i, data shifting control signals and the input multiplexers select signals, to PE slaves. A finite state machine (FSM) is designed for these purposes, and the state transition diagram is shown in Fig. 9. With the PE master FSM, the CORDIC iteration circuit, in Fig. 5, can be extended to a complete CORDIC master, depicted in Fig. 10.

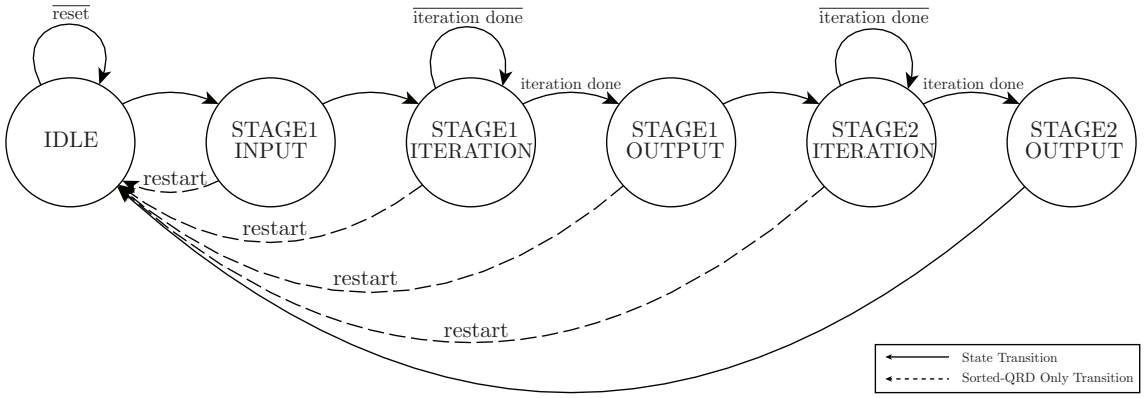


Figure 9: State transition diagram of PE master FSM. For simplicity, only state transition signals are annotated on the diagram, and state inputs and outputs are omitted. State "STAGE1 INPUT" implements the mandatory correction rotation to all input vectors, and no "STAGE2 INPUT" is implemented because no correction rotation is needed in the second stage. The "OUTPUT" states are used for CORDIC scale-factor K compensations.

The design of a CORDIC slave is nearly-identical to the design of a CORDIC master, except that control signals are inputs to slaves instead of being generated internally. PE slaves contain two CORDIC slaves and they can rotate 2-dimension complex-valued vectors based on control signals from CORDIC masters. Compared to standard systolic array architecture in which every PE requires dedicated control logic,

the advantage of the master-slave architecture is that the control logic is centralized in a single PE master, and only control signals need to be stored and distributed to slaves.

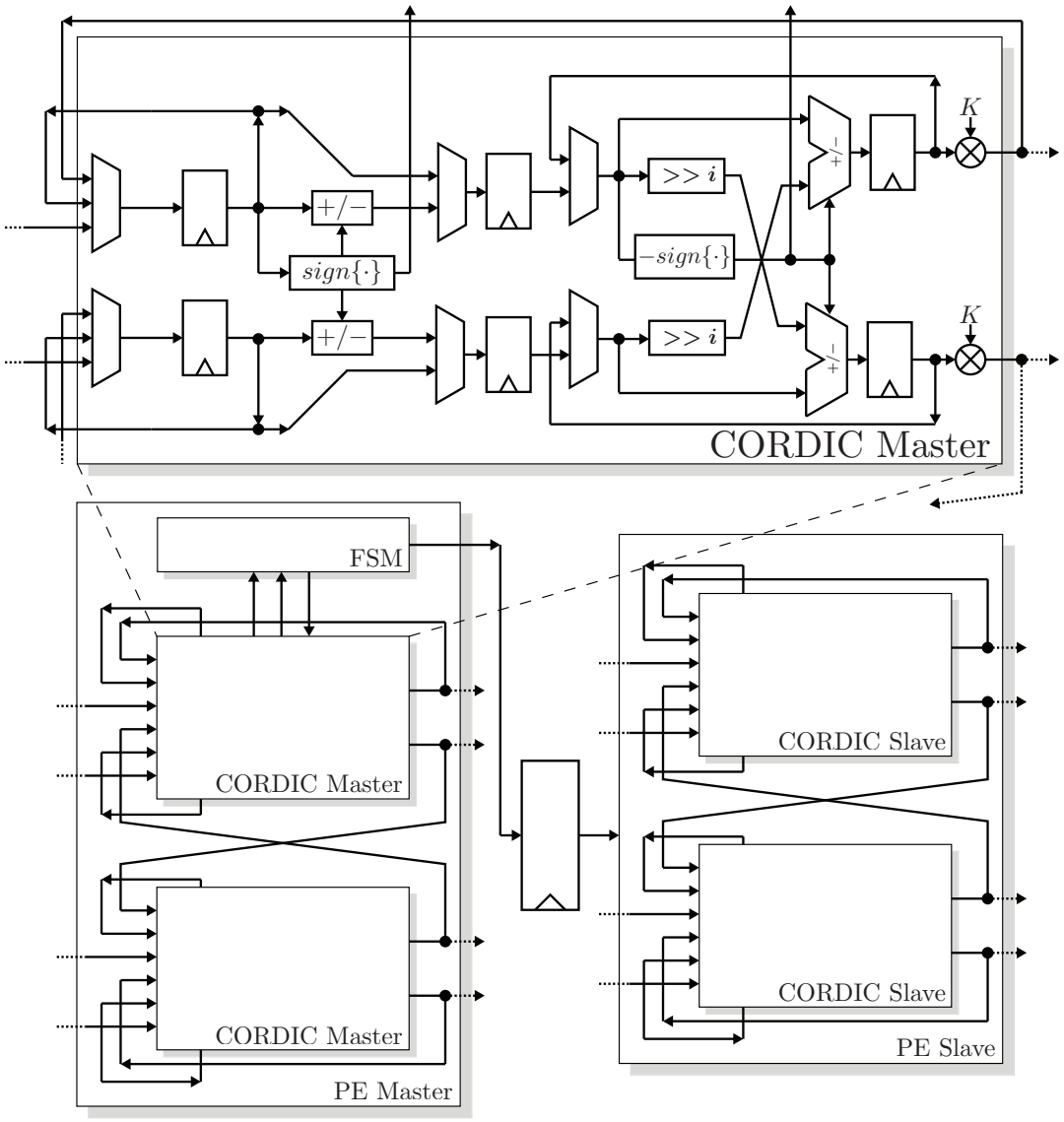


Figure 10: Proposed CORDIC master architecture. The CORDIC master datapath is properly pipelined for FSM states of each TACR stage in Fig. 9. Multipliers are used for scale-factor compensation, and $K \approx 1.743286$.

4.3 Data Transceivers

4.3.1 Runtime Address Computations

Data Transceivers have two main functions in the proposed architecture. First, they compute memory addresses in runtime for data retrieving and storing. As stated in Sec. 4.1, the purpose to compute memory addresses in runtime is to automate the address adjustments for scaling the architecture. The proposed architecture uses counters as address pointers to track the row index and the column index of requested data. By applying QRD algorithm column-wise from top to bottom, as suggested in Table 6, input matrix data needs to be retrieved row by row. Hence, memory addresses can be calculated by:

$$\texttt{address} = \texttt{row\_index} + \texttt{col\_index} * \texttt{dimension}. \tag{33}$$

With (33), the algorithm can normalize the first column of any size of matrices. However, to normalize the next column, both row and column indices have to start with an offset value, one, because of the dimension diminishing effect of QRD problems. Whereas, the effect does not apply for the accumulation of $\boldsymbol{Q}$, which creates a dimension gap between $\boldsymbol{R}$ and $\boldsymbol{Q}$ computations. In order to solve the complication, a circulating address algorithm is proposed in Table 7. The algorithm follows three basic ideas:

- First, an offset value is realized by a counter that starts from zero and increases only when both column and row counters reach (`dimension-1`). The offset value represents the index of the column that is being normalized;
- Second, the row index has to be set to the offset value after a column is fully normalized. It is realized by a counter that is capable of loading an initial up-counting value from offset counter after each offset value increment;
- Third, the column indices depend on the summation of column counter values

and offset values. Offset values adjust counter values for the dimension diminishing effect, but when the summation is beyond the matrix dimension, column indices should circulate back to zero and then start up-counting.

Table 7: A circulating address algorithm for proposed QRD architecture.

```
(1)  initialize variables to 0
(2)  while( !reset && !qrd_done ) {
(3)    col_count = col_count + 1;
(4)    if( (col_count + offset) > (dimension-1) ) {
(5)      col_index = col_count + offset - dimension;
(6)    } else {
(7)      col_index = col_count + offset; }
(8)    address = row_index + col_index * dimension;
(9)    if( col_count == (dimension-1) && row_index == (dimension-1) ) {
(10)     offset = offset + 1;
(11)     row_index = offset;
(12)     row_update_bypass = 1; }
(13)   if( col_count == (dimension-1) && !row_update_bypass ) {
(14)     row_index = row_index + 1;
(15)     row_update_bypass = 0; }
(16)   wait( data_request ); }
```

The circulating address algorithm is meant to fulfill the data requests from $\boldsymbol{Q}$ computations. Meanwhile, it does not disturb the computations of $\boldsymbol{R}$ because the circulating entries are always zeros. It should be pointed out that for matrices whose dimension can be expressed as $\{2^i | i \in \mathbb{Z}, i > 0\}$, the circulating addresses can be implemented very efficiently because of the wrapping nature of fixed-point numbers. In those cases, column indices are simply the sum of column counter values and offset values, and the multiplication in (33) is equivalent to shifting numbers to the right by $\log_2(\texttt{dimension})$.

The algorithm in Table 7 can be used for both retrieving and storing data with different initial offset values. The proposed architecture uses separate address computation engines for data retrieving and storing, and it coordinates them in a higher level.

4.3.2 Data Storing Policies

The second main function of data transceivers is to efficiently retrieve data from memories and to store prossed data to memories. The data storing unit is responsible for generating addresses for processed data from PEs, so data can be directly written from PEs to the correct locations in memories. For this purpose, the data storing unit is merely an address computation engine controlled by an additional FSM.

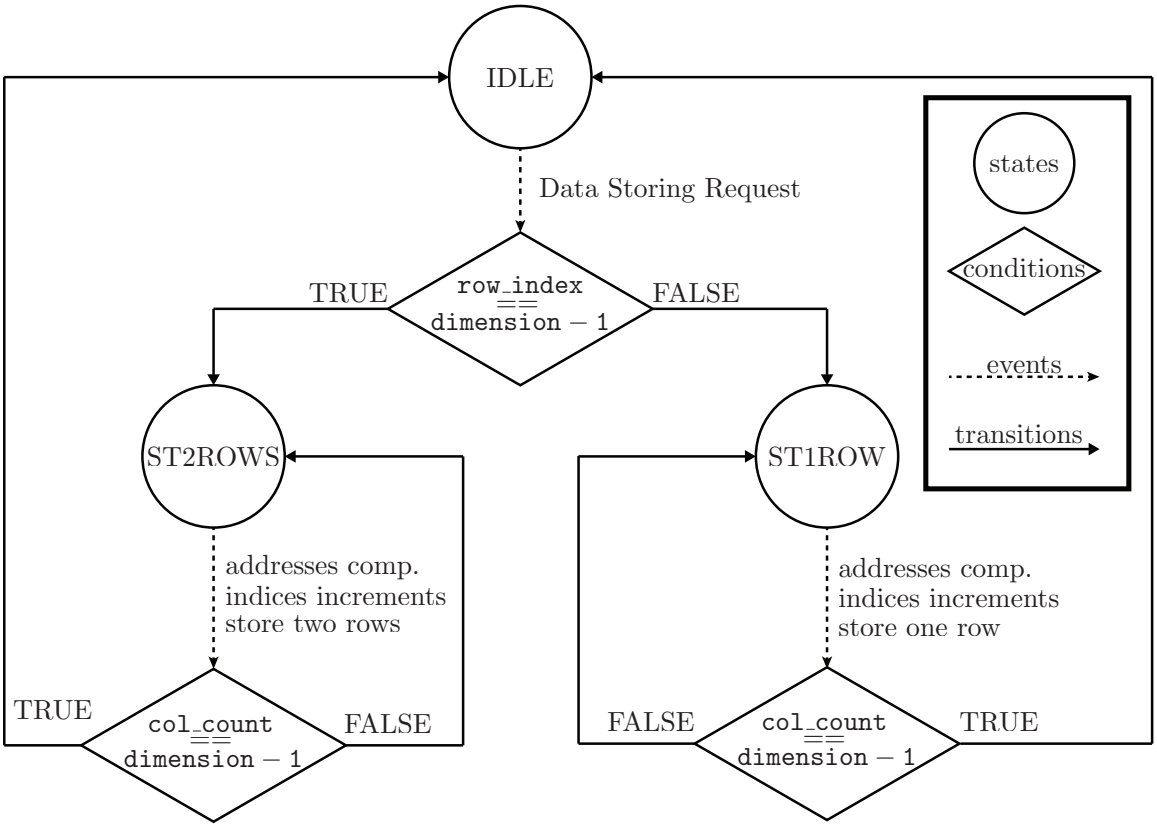


Figure 11: State decision diagram of proposed data storing policies. Data storing FSM follows the decision diagram to control the address computation engine for storing processed data from PEs.

The design principal is to have data storing polices as straightforward as possible so that data transceivers can spend most of their effort fetching data from memories and buffering the data in FIFOs. With applying QRD column-wise from top to bottom, the top row of the current QRD matrix stays in PEs until the bottom row is completely processed. In other words, in one scenario, only one row of matrix entries

needs to be stored. On the other hand, after PEs finish processing the bottom row, the top row entries need to be stored into the output memory, while bottom row entries are stored back to input memories. In Sec. 4.1, the decision to have output memories separated from input memories simplifies the second scenario, so data from both rows can be written to memories simultaneously. A three-state FSM can support these scenarios, and the state decision diagram is illustrated in Fig. 11. Both storing states take as many clock cycles as `dimension` to complete, and then the storing unit releases memory control to the data retrieving unit.

4.3.3 Data Retrieving Policies

The goal of data retrieving policies is to minimize computation stalls in PEs and to achieve near-theoretical latency in (31). Hence, the data retrieving unit either sends data directly to PEs while PEs are idle, or buffers data in FIFOs while PEs are still occupied. Just like data storing scenarios, PEs do not always request two rows of matrix entries from memories. Only the very first TACR to normalize a full column requires a double-row transaction, while the rest of TACRs for that column just needs a single-row transaction. Therefore, policies that are similar to the ones for data storing can be used to handle these two cases.

The buffering process, however, becomes complicated when the requested data is still being processed in PEs. With the existing policies, the retrieving unit will have to wait until the storing unit stores data into memories. In order to minimize the delay introduced by this scenario, the retrieving unit forwards the data that is just processed by PEs and not yet stored in memories, directly to PEs. With the new data forwarding feature, five operational states along with an idle state can cover all data retrieving scenarios, illustrated in Fig. 12.

The pipelined PE structure introduces two-clock-cycle stalls in each forwarding state. It is also important to point out that neither a 2×2 nor a 3×3 QRD processor

requires all data retrieving states in Fig. 12, and the control logic can be simplified in those two cases. Whereas, matrices with dimension of 4×4 and beyond require all states to complete QRD computations.

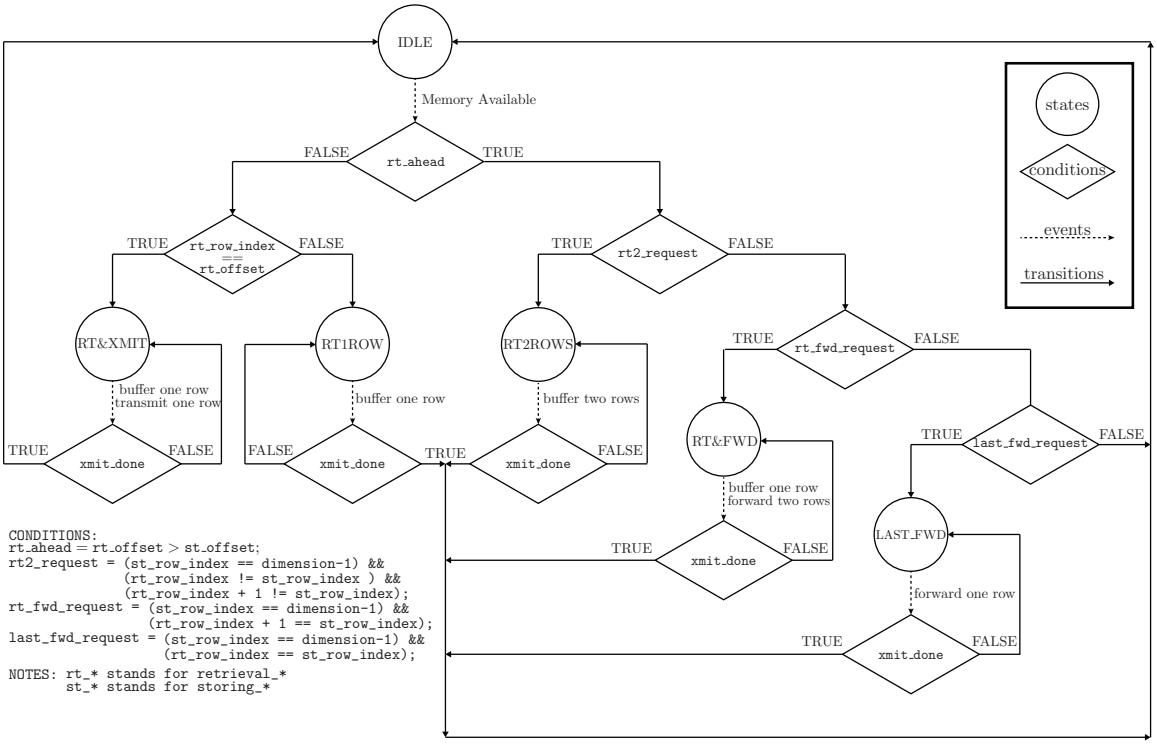


Figure 12: State decision diagram of proposed data retrieving policies. State transition conditions are generated by a coordinator that makes decisions based on the progress from both retrieving and storing units. Based on these decisions, the retrieving unit either buffers memory data in FIFOs or forwards data directly to PEs.

4.4 Adjustments for Sorted-QRD Algorithm

4.4.1 Sorting Circuit

Algorithms and policies designed in the previous sections are fully compliant to the proposed sorted-QRD architecture. The extension for the sorted-QRD architecture is the additional sorting circuit and the column-index RAM, previously shown in Fig. 8. The sorting circuit finds a minimum-norm column and permutes it to the currently-leading column position before PEs start normalizing each subsequent column. Table 3 has suggested that the squared Frobenius norm is used for finding

minimum-norm columns. It implies the use of multipliers and double-data-bandwidth adders in the sorting-circuit datapath, which is likely to become the critical path of the entire design. In order to simplify the circuit, a L_1-norm approximation is proposed, and the L_1-norm for a n-dimension complex-valued vector $\boldsymbol{v}$ is defined as follows:

$$\|\boldsymbol{v}\|_{L_1} = \sum_{i=1}^{n} (|\Re\{\boldsymbol{v}_i\}| + |\Im\{\boldsymbol{v}_i\}|). \tag{34}$$

The approximation can be realized by using two saturating adders separated by pipelining registers. After the norm is calculated, they have to be saved in memories for further updates in the algorithm. Hence, a block RAM to store norm values is placed at the sorting-circuit level. Meanwhile, a comparator takes in norm values and compares them with the norm of the currently-leading column. A active-high signal is generated if the minimal-norm column is not at the leading position. Finally, the minimal-norm column index and the corresponding norm value are exchanged with the ones at the leading column position in memories based on comparison results. The permutation process will be skipped if the indication signal is low.

Notice that the permutation does not take place in data memories. Hence, address computation engines cannot simply use column counter values as column indices for data retrieving and storing. Instead, they retrieve sorted column indices from the $\boldsymbol{p}$ memory using column counter values, and then they compute the addresses for matrix memories based on the sorted column indices. In this way, the proposed architecture avoids permutations inside data memories.

In addition, the proposed sorting circuit uses shadow memories to avoid memory access conflicts caused by retrieving and updating column indices and norm values at the same time. While main memories are being updated by the sorting circuit, the shadow memories supply data to requestors. Once the update is completed, the shadow memories will be synchronized with the main memories, so they contain the updated information.

The architecture of the proposed sorting circuit is depicted in Fig. 13. The sorting

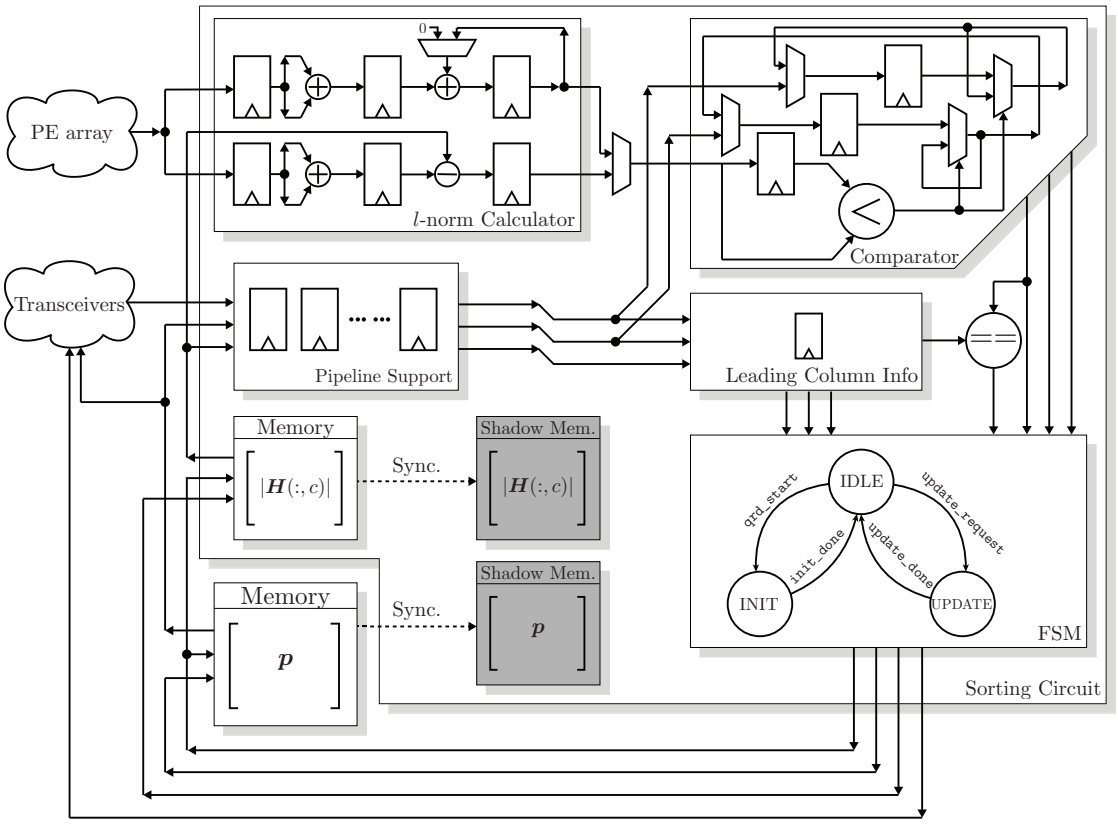


Figure 13: Proposed sorting circuit architecture. Column-index memory locates in the system level along with PE array and transceivers, but the column-norm memory as well as shadow memories lie at the sorting circuit level.

circuit introduces additional delays to the algorithm because no insured computation can proceed until the minimum-norm column is found.

4.4.2 Aggressive Processing

In order to minimize the additional delays introduced by column-ordering processes, the proposed sorted-QRD architecture uses an aggressive processing scheme to avoid unnecessary stalls. Notice that when a currently-leading column has minimum norm, a column-ordering process is irrelevant. To take advantage of this observation, the aggressive processing scheme assumes the minimum-norm column is always at the currently-leading position. The optimization allows computations in PEs proceed with no stall when the sorting circuit does not request permutations. However, if a permutation is required, i.e., the minimum-norm column is not the currently-leading

column, PEs have to scratch the data being processed and request a new set of data from memories. In order to support this functionality, Fig. 9 illustrates the use of a "restart" signal to interrupt PEs and resume computations from the beginning.

By using the aggressive processing scheme, the proposed sorted-QRD architecture avoids the worst case latency for most input matrices. An $n \times n$ matrix requires total $n-1$ column ordering processes. Let d_i to denote the logic decision made by the $(i+1)$-th ordering process, indicating whether or not the process requires a permutation. Then, a vector $\boldsymbol{d} = [d_0, d_1, \ldots, d_{n-2}]$ can be defined to represent all column-ordering scenarios that lead to different system latencies. Since d_i is a logic variable, $\boldsymbol{d}$ can be equivalently represented by a decimal number as:

$$D = \sum_{i=0}^{n-2} 2^{d_i}, \text{where } \; d_i = 1, \text{or } \; 0. \tag{35}$$

The worst case latency is represented by an all-one vector $\boldsymbol{d}$, or a decimal number $D = 2^{n-1} - 1$. Fig. 14 shows a histogram of occurring column-ordering scenarios in 4×4 sorted-QRD simulations. Evidently, the system escapes from worst case latency

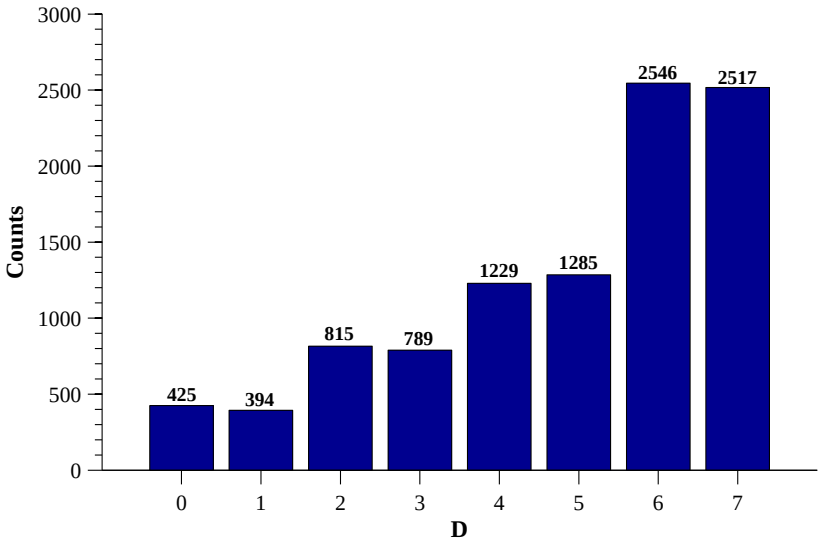


Figure 14: Histogram of occurring column-ordering scenarios for 4×4 sorted-QRD based on $10,000$ simulations. L_1-norm approximation was used to estimate the column norm in simulations.

for 74.8% of all input matrices.

4.5 Design Trade-off

An important design trade-off between the clock frequency and resource cost exists in many CORDIC-based applications. Since CORDIC iterations are identical, they can be unrolled and pipelined to further minimize the critical path [11]. Ideally, a N-iteration CORDIC routine can be realized by N identical iteration datapath, pipelined into each iteration per clock cycle. In that case, the critical path simply consists of a hard-wired shifting operation and an adder. However, pipelining unrolled-iterations does not improve the latency of a CORDIC-based design, which is the primary concern of the proposed QRD architecture. For this reason, we can try improving the latency by unrolling CORDIC iterations with a factor k without pipelining. In other words, k CORDIC iterations are now processed in one clock cycle instead of k cycles. Hence, the theoretical latency with an unrolling factor k can be reformulated as:

$$latency = (n(n-1)+1) \cdot \lceil (N+3)/k \rceil - n(n-1)/2. \tag{36}$$

The penalty of unrolling CORDIC iterations without pipelining is a lengthened critical path, which slows down the clock frequency. It is unlikely that unrolling by a factor of 2 will halve the clock frequency because of the simplification of shifting operations in hardware. Therefore, it is expected that a performance gain can be obtained by unrolling with $k = 2$. It is unclear whether or not throughput will continue to improve by further unrolling CORDIC iterations because the penalty on clock frequency and the higher resource cost on FPGAs may degrade the performance.

CHAPTER V

HARDWARE IMPLEMENTATION AND EVALUATION

Based on the proposed architecture discussed in the previous chapter, a scalable CORDIC-based QRD implementation is realized on Xilinx FPGA platforms. In this chapter, we evaluate both the numerical and the hardware performance of the implementation. Then, we compare the performance of our design to existing QRD processors in literatures. Finally, we address the limitations of our design.

5.1 Numerical Performance

In order to efficiently evaluate the numerical performance of the proposed architecture, we built a numerical model that was equivalent to the hardware implementation using C language. The model imitated fixed-point arithmetic with quantization functions. With this model, we were able to explore the numerical performance of the proposed architecture. Fig. 15 shows the dissimilarity and orthogonality of the proposed QRD architecture varied by matrices dimension. Both dissimilarity and orthogonality improve as more fraction bits are used in simulations, and more fraction bits are necessary for larger dimension matrices to combat numerical error.

It is important to realize that the dissimilarity, defined in Sec. 2.1, is collaboratively determined by the number of fraction bits and the number of CORDIC iterations. Numerical simulations have suggested that the number of fraction bits plays a more important role in dissimilarity, which is evidently shown on Fig. 16. For a particular number precision, for instance, a 15-bit fixed-point representation with 10-fraction bits, the dissimilarity saturates at eleven CORDIC iterations and more CORDIC iterations do not help lower the dissimilarity. With two or three less

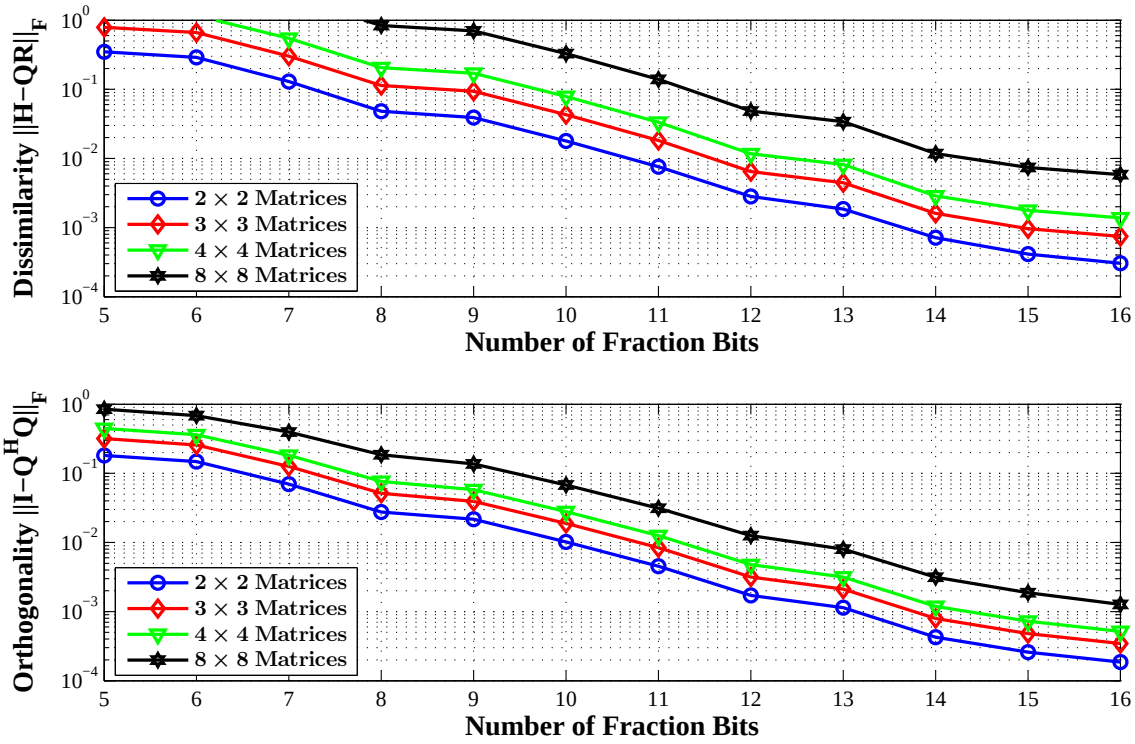


Figure 15: Numerical performance of the proposed architecture. For each point, 10,000 QRD simulations were conducted with randomly-generated matrices. The simulations used 5-integer bits and N-fraction bits, which ranges from 5 to 16. $N + 1$ CORDIC iterations were used in all simulations.

CORDIC iterations, the numerical performance only suffers from minimum dissimilarity degradation. In order to confirm that less CORDIC iterations can be used, a MIMO system level simulation has to be thoroughly conducted, which is out of the scope of this study.

5.2 Implementation and Hardware Performance

The hardware implementation was written in Verilog and was synthesized using Synopsys Synplify Pro. The synthesized design was further placed and routed by the Xilinx PAR tool to generate gate-level abstraction for Xilinx FPGAs. The static timing analysis was generated after the PAR process to provide accurate design timing statistics. We were able to functionally verify the hardware implementation by matching results with the numerical model using randomly-generated input matrices. The

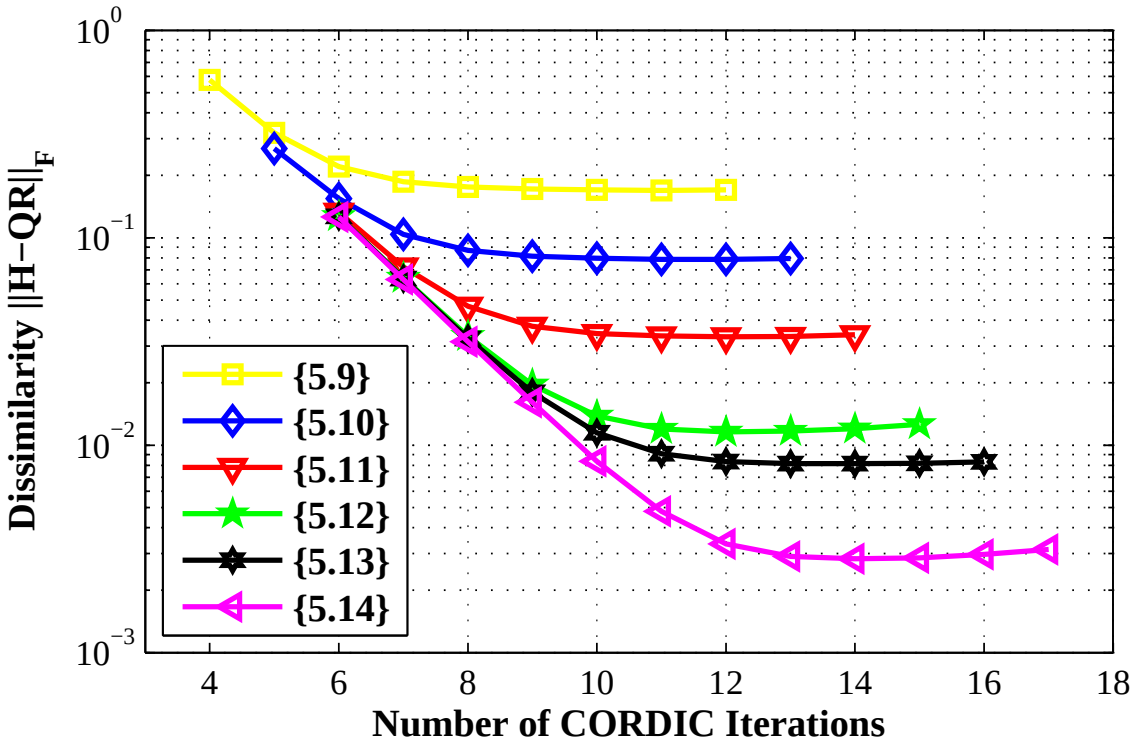


Figure 16: Impact of CORDIC iterations on the numerical performance of the proposed 4×4 complex-valued QRD algorithm. For each numerical precision, 10,000 QRD simulations were conducted with randomly-generated matrices. The numerical precision is represented by $\{i.f\}$, i.e., a fixed-point representation with i-integer bits and f-fraction bits.

functional verification was conducted in a bit-precise fashion to ensure the correctness of the implementation. The latency was also measured during simulations, and the throughput was calculated based on the measured latency and the clock frequency obtained from the post-PAR timing analysis. Based on the numerical performance, we picked the number precision that was closest to, yet better than, the dissimilarity of 10^{-1} for all implementations. It led to 5-integer bits and 8-fraction bits, i.e., $\{5.8\}$, for the 2×2 QRD implementation, and $\{5.9\}$, $\{5.10\}$ and $\{5.12\}$ for 3×3, 4×4 and 8×8 case respectively. $N + 1$ CORDIC iterations were chosen for best numerical performance, where N corresponds to the number of fraction bits. Important implementation characteristics including the latency, the post-PAR clock frequency and hardware utilization are tabulated in Table 8 for two different generations of Xilinx FPGAs.

From Table 8, we can see that using a more recent technology, the timing of all implementations are significantly improved, which further improves the throughput. Sorted-QRD designs consume more hardware resources than QRD designs due to the additional sorting circuit, but they are able to achieve the same timing performance as QRD designs with the proposed architecture.

Moreover, we conducted experiments to explore design trade-offs by unrolling the CORDIC iterations using factors from 2 to 4. Fig. 17 shows the trade-offs introduced by unrolling factors on our 2×2 QRD implementation using Xilinx Virtex-5 technology. As explained in Sec 4.5, unrolling by a factor of 2 promises an improvement of

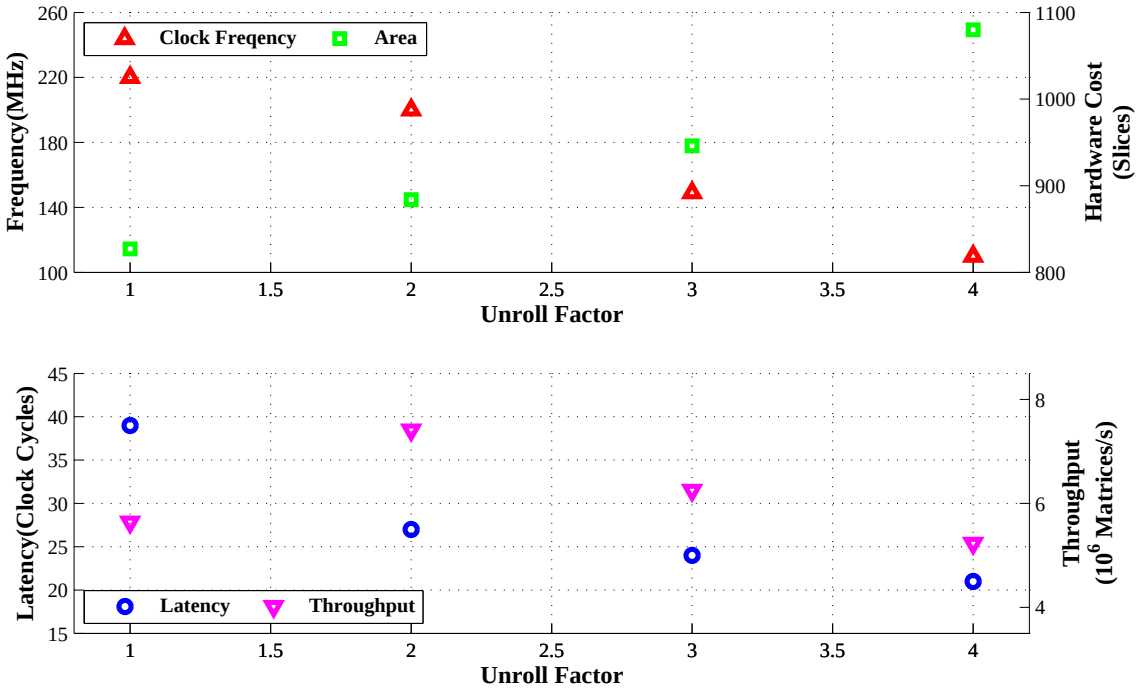


Figure 17: The two trade-offs: frequency v.s. hardware cost (above), and latency v.s. throughput (below), introduced by unrolling CORDIC iterations by factors from 2 to 4(a factor of 1 represents the case of no unrolling). The experiments were conducted on proposed 2×2 QRD architecture using Xilinx Virtex-5 technology with {5.8} fixed-point precision and 8 CORDIC iterations.

system throughput, which demonstrates the best throughput performance on Fig. 17. Even through further unrolling CORDIC iterations continues to lower the system latency, the latency improvement is not sufficient enough to compensate the loss in

clock frequency, which causes throughput to decrease.

5.3 Comparison to Existing Work

Existing QRD designs have used a variety of VLSI technologies with different transistor models. The technological difference complicates the performance comparison due to the significant impact on design timing and cost. Therefore, we focus on comparing our implementation to the designs that use similar algorithms to our work. Table 9 tabulates five different designs along with our implementation for solving 2×2 and 4×4 complex-valued QRD problems. All implementations deploy the CORDIC-based Givens rotation as a basic design component, but each work uses a different scheduling scheme in the algorithm supported by customized hardware architecture. While the majority of these designs used application-specific integrated circuit (ASIC) technology [6, 7, 8, 12], researchers in [15] chose Altera Stratix FPGA as the implementation platform. Due to the lack of the numerical evidence on number precision to support a MIMO system, we chose our design parameters including number precision and number of CORDIC iterations, based on the average of existing studies for a particular QRD dimension. Meanwhile, we unroll CORDIC iterations by a factor of 2 to achieve the best throughput results in a cost of higher hardware utilization.

Table 9 shows that designs in [6, 7, 12] have clear throughput advantages compared to our implementation performance. The reason for the difference in performance is twofold. First, while we only use an array of PEs in the proposed architecture, implementations in [6, 7, 12] adopt either a triangular-array type of architecture or a multi-stage datapath that can process the next matrix before the previous matrix is completed. Second, ASIC technology allows these implementations to pack multiple CORDIC iterations in a single clock cycle without a harsh timing penalty, which

greatly reduces the latency. In [8], however, the implementation uses a single normalization core with complex-valued multipliers for rotations, which is similar to ours except that we use the CORDIC algorithm for rotations. The design similarities result into similar performance between [8] and our work. Furthermore, on FPGA platforms, our design has out-performed the design in [15] in almost every category.

Similarly, we tabulate two more Sorted-QRD studies and compare them with our Sorted-QRD implementations in Table 10. The latency of our design increases almost 50% in average because of additional column ordering processes. The implementation in [8] proves to have better throughput even running at a lower clock frequency. In the case of 8×8 complex-valued sorted-QRDs, our implementation has approximately 26% lower latency compared to the one reported in [20]. Our design on Xilinx Virtex-5 platform also runs at the clock frequency about 128% higher than [20], which leads to a much better system throughput.

In summary, our design is able to achieve moderate throughput performance on FPGAs compared to the existing implementations. The 4×4 QRD throughput performance is accomplished with approximately 13% hardware utilization on the Xilinx Virtex 5 FPGA platform. The low hardware utilization is well desired, considering fitting a QRD processor and a MIMO detection algorithm in a single FPGA chip in the future. Moreover, compared to an ASIC design that targets a specific size of matrices and a particular system, our parametric FPGA design can be used for any size of matrices, and it is also compliant with most MIMO detection algorithms. The flexibility facilitates the design of MIMO detectors on FPGAs, and enables fast system-level prototyping for MIMO detectors. At last, all compared studies have used multiple-port memories to support their processors, while we manage to use single-port memories and achieve acceptable performance.

On the other hand, the comparison also exposes two main limitations of our proposed architecture. First, the PE-array type of architecture has the limit of processing

one matrix at a time. This limitation can be lifted by adopting a triangular systolic array architecture. However, a straightforward deployment will result into very high hardware cost, which may degrade timing performance on FPGAs. Second, the limited memory bandwidth is likely to hinder computations in high-dimension QRD problems. With a single port memory, computations may have to be stalled regularly in those cases, which drastically increases the latency of the proposed architecture.

Table 8: Hardware implementation results

Platforms	Design Chracteristics	Regular QRD				Sorted QRD			
GENERAL	**Dimension**	2×2	3×3	4×4	8×8	2×2	3×3	4×4	8×8
	Precision	{5.8}	{5.9}	{5.10}	{5.12}	{5.8}	{5.9}	{5.10}	{5.12}
	Latency	42	95	179	853	ave. 47†	ave. 114†	ave. 220†	ave. 1035†
XC2VP30-7	**Clock(MHz)**	165	160	155	130	165	160	155	130
	Hardware Use(Slices)	1,906	3,099	4,641	11,718	2,132	3,369	4,824	11,843
	Throughput	3.92M	1.68M	0.865M	0.152M	3.51M*	1.40M*	0.704M*	0.125M*
XC5VLX110-3	**Clock(MHz)**	220	215	200	185	220	215	200	185
	Hardware Use(Slices)	827	1,518	2,362	5,801	1,036	1,635	2,493	6,307
	Throughput	5.23M	2.26M	1.12M	0.217	4.68M	1.88M	0.909M	0.178M

† Averaged latency based on 45,000 simulation runs
* Throughput is calculated based on averaged latency

Table 9: QRD implementation comparison

	[7]	[12]	[8]	[6]	[15]	Proposed Architecture	
QRD Dimension	2×2	4×4	$8 \times 4^{\dagger}$	4×4	4×4	2×2	4×4
Technology	TSMC $0.18\mu m$	$0.13\mu m$ CMOS	$0.25\mu m$ CMOS	$0.18\mu m$ CMOS	EP1S25F780C7	XC5VLX110FF1153-3	
Number Precision	NA	{5.11}	{3.10}	{5.11}	{6.10}	{5.8}	{5.11}
Num. of CORDIC Iterations	8	8	9	9	8	8	8
Num. of Micro-Rot./cycle	NA	NA	NA	NA	NA	2	2
Clock(MHz)	202	270	125	100	121	200	175
Latency(clock cycles)	8	40	67	4	211	27	92
Throughput	est.‡ 25.3M	est.‡ 6.75M	1.56M	12.5M	est. 0.573M‡	7.41M	1.90M
Hardware Cost	17K gates	36K gates	54K gates	111K gates	1,528 logic cells	884 Slices	1,920 Slices

† MMSE-QRD

‡ Estimated based on the reported Clock and Latency Information

Table 10: Sorted-QRD implementation comparison

	[8]	[20]	Proposed Architecture	
QRD Dimension	$8 \times 4^{\dagger}$	8×8	4×4	8×8
Technology	$0.25\mu m$ CMOS	TSMC $0.18\mu m$	XC5VLX110FF1153-3	
Number Precision	{3.10}	13-bit word-length	{5.11}	{5.12}
Num. of CORDIC Iterations	9	NA	8	10
Num. of Micro-Rot./cycle	NA	NA	2	1
Clock(MHz)	125	81	175	185
Latency(clock cycles)	80	est. $1191^{\ddagger}$	avg. $136^{\circledast}$	avg. $885^{\circledast}$
Throughput	1.56M	0.0680M	$1.28M^{*}$	$0.209M^{*}$
Hardware Cost	54K gates	NA	2,091 Slices	5,056 Slices

† MMSE-QRD
‡ Estimated based on the report Clock and Throughput Information
$^{\circledast}$ Averaged latency based on 45,000 simulation runs
* Throughput is calculated based on averaged latency

CHAPTER VI

CONCLUSIONS

This thesis studied the complex-valued QRD and sorted-QRD algorithm, and realized the algorithm using CORDIC-based Givens rotations on FPGA platforms. It started with a real-valued QRD algorithm using Givens rotations, and extended it to the complex-valued case using TACRs. Then, CORDIC algorithm was introduced to use fixed-point arithmetic in the algorithm. The proposed algorithm simplified a CORDIC vectoring-and-rotation sequence with a master-slave architecture to improve latency and result precision. A PE array architecture was proposed based on the simplification to parallelize vector normalization and rotations. In the meantime, the study offered two novelties to allow the proposed architecture to achieve a near-theoretical latency. First, the study proposed data retrieving and storing to work with single-port memories and to minimize computation stalls in PEs. Second, an aggressive processing scheme was used for the sorted-QRD architecture to avoid the worst case latency for most matrices. Moreover, the proposed architecture was fully-parametric and could be scaled to solve different sizes of QRDs by changing RTL parameters and constants. Four QRD processors for the dimension of 2×2, 3×3, 4×4 and 8×8 were implemented on Xilinx FPGA platforms. The implementations achieved moderate performance compared to existing ASIC QRD designs and better performance compared to the existing design on FPGAs.

In order to fulfill higher throughput requirements in real-world MIMO communication systems or future generation standards, further studies have to be conducted to improve the throughput of the proposed architecture. Here is a list of suggestions for future research directions:

- To fully utilize the parallelism in a complex-valued QRD algorithm, one should consider multi-dimensional rotations instead of two-dimensional rotations. The deployment of multi-dimensional rotations will likely complicate the computations of $\boldsymbol{Q}$ matrices, which becomes the main challenge of this approach. Alternatively, one could add more 2-dimension normalization and rotation processors to improve the throughput. However, a detailed scheduling must be made to avoid low utilization of these additional processing cores.

- Throughput improvements will likely demand multiple data transactions per clock cycle from memories. The data management polices proposed in this study may not be applicable for these improvements. A better choice is to use multiple-port memories or register files to allow processors to read or write multiple data entries in one clock cycle. The target VLSI technology must support multiple-port memories in order to achieve efficient hardware implementations.

- To further improve the latency of sorted-QRD algorithms, one could try adding a prediction circuit that projects the next processing order for the following normalization. This is an alternative approach to the aggressive processing scheme proposed in this study. The focus will be to study whether or not the prediction can further lower the latency for sorted-QRD algorithms.

REFERENCES

[1] Burg, A., "VLSI circuits for MIMO communication systems," Ph.D. dissertation, IIS/ETH-Zurich, Feb. 2006.

[2] Chen, D. and Mihai, S., "Fixed-point CORDIC-based QR decomposition by Givens rotations on FPGA," in *Int. Conf. on Reconfigurable Computing and FPGAs*, (Cancún, Mexico), Nov. 2011, pp. 327–332.

[3] Chiu, P.-L., Huang, L.-Z., and Huang, Y.-H., "Scalable interpolation-based QRD architecture for subcarrier-grouped-ordering MIMO-OFDM system," in *43rd Asilomar Conf. on Signals, Syst., and Computers*, (Pacific Grove, USA), Nov. 2009, pp. 708–712.

[4] Gestner, B., Zhang, W., Ma, X., and Anderson, D. V., "Lattice reduction for MIMO detection: from theoretical analysis to hardware realization," *IEEE Trans. Circuits Syst. I. Reg. Papers*, vol. 58, pp. 813–826, Apr. 2011.

[5] Golub, G. H. and Loan, C. F. V., "Orthogonalization and least squares,", in *Matrix Computations*, 3rd ed. Baltimore, Maryland: The John Hopkins Univ. Press, 1996.

[6] Huang, Z.-Y. and Tsai, P.-Y., "Efficient implementation of QR decomposition for Gigabit MIMO-OFDM systems," *IEEE Trans. on Circuits and Syst.*, vol. 58, pp. 2531–2542, Oct. 2011.

[7] Hwang, Y.-T. and Chen, W.-D., "A low complexity complex QR factorization design for signal detection in MIMO OFDM systems," in *IEEE Int. Symp. on Circuits and Syst.*, (Seattle, USA), May 2008, pp. 932–935.

[8] Luethi, P., Burg, A., Haene, S., Perels, D., , Felber, N., and Fichtner, W., "VLSI implementation of a high-speed iterative sorted MMSE QR decomposition," in *IEEE Int. Symp. on Circuits and Syst.*, (New Orleans, USA), May 2007, pp. 1421–1424.

[9] Luethi, P., Studer, C., Duetsch, S., Zgraggen, E., Kaeslin, H., Felber, N., and Fichtner, W., "Gram-Schmidt-based QR decomposition for MIMO detection: VLSI implementation and comparison," in *IEEE Asia Pacific Conf. on Circuits and Syst.*, (Macao, China), Dec. 2008, pp. 830–833.

[10] Maltsev, A., Pestertsov, V., Maslennikov, R., and Khoryaev, A., "Triangular systolic array with reduced latency for QR-decomposition of complex matrices," in *Proc. of IEEE Int. Symp. on Circuit and Syst.*, (Island of Kos, Greek), May 2006.

[11] Meher, P. K., Valls, J., Juang, T., Sridharan, K., and Maharatna, K., "50 years of CORDIC: algorithms, architectures, and applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, pp. 1893–1907, Sep. 2009.

[12] Patel, D., Shabany, M., and Gulak, P. G., "A low-complexity high-speed QR decomposition implementation for MIMO receivers," in *IEEE Int. Symp. on Circuits and Syst.*, (Taipei, China), May 2009, pp. 33–36.

[13] Salmela, P., Burian, A., Sorokin, H., and Takala, J., "Complex-valued QR decomposition implementation for MIMO receivers," in *IEEE Int. Conf. on on Acoust., Speech and Signal Process.*, Apr. 2008, pp. 1433–1436.

[14] Singh, C. K., Prasad, S. H., and Balsara, P. T., "VLSI architecture for matrix inversion using modified Gram-Schmidt based QR decomposition," in *Int. Conf. on VLSI Design*, (Bangalore, India), Jan. 2007, pp. 836–841.

[15] Sobhanmanesh, F. and Nooshabadi, S., "Parametric minimum hardware QR-factoriser architecture for V-BLAST detection," *IEE Proc. Circuits, Devices and Syst.*, vol. 153, pp. 433–441, Oct. 2006.

[16] Strobach, P. and Goryn, D., "A computation of the sliding window recursive QR decomposition," in *IEEE Int. Conf. on Acoust., Speech, and Signal Process.*, (Minneapolis, USA), Apr. 1993, vol. 4, pp. 29–32.

[17] Taherzadeh, M., Mobasher, A., and Khandani, A. K., "LLL reduction achieves the receive diversity in MIMO decoding," *IEEE Trans. Inf. Theory*, vol. 53, pp. 4801–4805, Dec. 2007.

[18] Volder, J. E., "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Computers*, vol. EC-8, pp. 330–334, Sep. 1959.

[19] Volder, J. E., "The birth of CORDIC," *VLSI Signal Process.*, vol. 25, pp. 101–105, 2000.

[20] Wang, J.-Y., Lai, R.-H., Chen, C.-M., Ting, P.-A., and Huang, Y.-H., "A 2×2 - 8×8 sorted QR decomposition processor for MIMO detection," in *IEEE Asian Solid State Circuits Conf.*, (Beijing, China), Nov. 2010, pp. 1–4.

[21] Wong, K.-W., Tsui, C.-Y., Cheng, R. S.-K., and ho Mow, W., "A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels," *IEEE Int. Symp. on Circuits and Syst.*, vol. 3, pp. 273–276, 2002.

[22] Wübben, D., Böhnke, R., Rinas, J., Kühn, V., and Kammeyer, K., "Efficient algorithm for detecting layered space-time codes," in *IEEE Proc. of ITG Conf. on Source and Channel Coding*, (Berlin, Germany), Jan. 2002, pp. 399–405.

[23] Wübben, D., Böhnke, R., Kühn, V., and Kammeyer, K.-D., "MMSE extension of V-BLAST based on sorted QR decomposition," in *IEEE 58th Vehicular Technology Conf.*, (Orlando, USA), Oct. 2003, vol. 1, pp. 508–512.

[24] Yao, H. and Wornell, G. W., "Lattice-reduction-aided detectors for MIMO communication systems," in *Proc. of IEEE Globe Telecmmun. Conference*, (Taipei, Taiwan), Nov. 2002.

[25] Zhang, W. and Ma, X., "Low-complexity soft-output decoding with lattice-reduction-aided detectors," *IEEE Trans. Commun.*, vol. 58, pp. 2621–2629, Sep. 2010.

VITA

Minzhen Ren was born on November 5, 1987 in Fuzhou, China. He graduated from Fuzhou No.1 Middle School in 2006, and attended Heibei University of Technology from September 2006 to January 2007, majoring civil engineering. He came to the United States in May 2007 with the purpose of seeking a different career direction, and transferred to Georgia Institute of Technology (Atlanta, GA) in August 2008 as a sophomore. While at Georgia Tech, he majored in Electrical and Computer Engineering and received his bachelor's degree in May, 2011. Since August 2011, he has been pursuing his master's degree in Electrical and Computer Engineering at Georgia Tech, while working for Texas Instruments Inc. (Greenville, SC) as a digital design/verification engineer through the cooperative education program. At T.I., he helped develop and verify several future Power Management Bus (PMBus) DC solution products for cloud, computing and communication applications. His academic focus includes digital signal processing (DSP), telecommunication theory, and VLSI designs, and his current research interest is hardware realization of algorithms for DSP and telecommunication applications.