LATTICE REDUCTION FOR MIMO DETECTION: FROM THEORETICAL ANALYSIS TO HARDWARE REALIZATION

A Thesis
Presented to
The Academic Faculty

by

Brian J. Gestner

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2011

LATTICE REDUCTION FOR MIMO DETECTION: FROM THEORETICAL ANALYSIS TO HARDWARE REALIZATION

Approved by:

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

Professor Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology

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

Professor Santosh Pande
College of Computing
Georgia Institute of Technology

Professor John Barry
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: 28 March 2011

To my parents

ACKNOWLEDGEMENTS

I want to thank my family, my advisor, my committee, and all my colleagues at Georgia Tech who made my academic and professional growth possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Upper-case, bold-face letters	matrices.
Lower-case, bold-face letters	column vectors.
Superscript $^{\mathcal{H}}$	Hermitian.
Superscript ^{T}	transpose.
Superscript *	conjugate.
Superscript $^{-1}$	inverse.
$\lvert \cdot \rvert$	absolute value of a scalar, cardinality of a set.
$\lVert \cdot \rVert$	Frobenius norm.
$\lceil \cdot \rceil$	integer ceiling.
$\lfloor \cdot \rfloor$	integer floor.
$\lfloor \cdot \rceil$	integer rounding.
$E\left[\cdot\right]$	expectation.
$\det\left[\cdot\right]$	determinant of a matrix.
$X_{m,n}$	the (m,n)-th entry of the matrix $\boldsymbol{X}$.
x_n	the n-th entry of the column vector $\boldsymbol{x}$.
$\boldsymbol{I}_N$	the $N \times N$ identity matrix.
$\mathbf{1}_{M\times N}$	all-one matrix of size $M \times N$.
$\mathbf{0}_{M\times N}$	all-zero matrix of size $M \times N$.
$\Re\left[\cdot\right]$	the real part.
$\Im\left[\cdot\right]$	the imaginary part.
$P\left\{A\right\}$	the probabilty of event A.
$Q\left(a\right)$	$\frac{1}{\sqrt{2\pi}}\int_a^\infty e^{-\frac{1}{2}x^2}dx$.

GLOSSARY

AoA angle of arrival, p. 61.

AS angular spread, p. 61.

ASIC application specific integrated circuit, p. 57.

BER bit-error-rate., p. 1.

CCDF complementary cumulative distribution function, p. 85.

CLLL complex Lenstra, Lenstra, Lovász., p. 2.

CORDIC coordinate rotation digital computer, p. 35.

DMI direct matrix inversion, p. 8.

FIFO first-in-first-out, p. 45.

FPGA field programmable gate array, p. 14.

LR lattice reduction., p. 2.

LUT look-up table, p. 29.

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

ML maximum likelihood., p. 1.

MMSE minimum mean squared error, p. 7.

NR Newton-Raphson, p. 28.

PR place-and-route, p. 53.

PRNG pseudo random number generator, p. 74.

QAM quadrature amplitude modulation, p. 9.

RA reliability assessment, p. 65.

SD sphere decoding, p. 2.

SE Schnorr-Euchner, p. 55.

SIC successive interference cancellation., p. 1.

SNR signal-to-noise-ratio, p. 6.

ZF zero-forcing., p. 1.

SUMMARY

The objective of the dissertation research is to understand the complex interaction between the algorithm and hardware aspects of symbol detection that is enhanced by lattice reduction (LR) preprocessing for wireless MIMO communication systems. The motivation for this work stems from the need to improve the bit-error-rate performance of conventional, low-complexity detectors while simultaneously exhibiting considerably reduced complexity when compared to the optimal method, maximum likelihood detection. Specifically, we first develop an understanding of the complex Lenstra-Lenstra-Lovász (CLLL) LR algorithm from a hardware perspective. This understanding leads to both algorithm modifications that reduce the required complexity and hardware architectures that are specifically optimized for the CLLL algorithm. Finally, we integrate this knowledge with an understanding of LR-aided MIMO symbol detection in a highly-correlated wireless environment, resulting in a joint LR/symbol detection algorithm that maps seamlessly to hardware. Hence, this dissertation forms the foundation for the adoption of lattice reduction algorithms in practical, high-throughput wireless MIMO communications systems.

CHAPTER I

INTRODUCTION

Researchers and engineers in industry alike see multiple-input multiple-output (MIMO) communication systems as a possible solution to the increasing demand for high data rate, power-efficient, wireless communication systems. MIMO communication systems achieve higher data rates than single antenna systems by using multiple antennas to transmit and to receive multiple independent data streams simultaneously over a communication channel. Each receiving antenna acquires a superposition of these transmitted streams. The process of separating out each independent data stream is called MIMO detection. Although the optimal solution to the MIMO symbol detection problem, maximum-likelihood (ML) detection, is known, a brute-force ML detector implementation involves an exhaustive search over all possible transmitted symbol vectors. This approach is infeasible for hardware implementation when either a large signal constellation or large number of antennas is employed. Hence, the challenge in designing hardware for MIMO symbol detection is to achieve comparable bit-error-rate (BER) performance to the ML detector while having low hardware complexity and meeting throughput and latency requirements.

In many practical MIMO communication systems, detection is enabled by periodically characterizing the relative contribution of each signal transmitted on each antenna to the signal received on each antenna. This process is referred to as channel estimation and is accomplished by transmitting known training signals at the start of each packet. A variety of low-complexity methods such as zero-forcing (ZF) and successive interference cancellation (SIC) detection involve a preprocessing step that

transforms knowledge about the channel behavior into a form suitable for symbol detection. This preprocessing result is then reused for each symbol detection until the channel is characterized again, which usually occurs when the next packet is received. The symbol detection step exhibits considerably lower complexity and involves either matrix multiplication or straightforward linear system solving. These low-complexity methods map well to hardware but have greatly reduced BER performance compared to the ML detector.

Enhancements that improve the BER performance of symbol detection, such as sphere decoding (SD) [25, 67, 2], significantly increase the overall complexity of the packet processing because the complexity of each symbol detection is greatly increased. Instead, it is clearly desirable to explore detection algorithms that achieve ML or near-ML performance at the cost of increased preprocessing complexity as opposed to increased symbol-rate processing complexity. Lattice reduction (LR)-aided detectors, which can incorporate lattice reduction algorithms into the preprocessing part of ZF or SIC detectors and only increase the symbol-rate processing complexity slightly, fit naturally into this observation.[1] Variants of the complex Lenstra-Lenstra-Lovász (CLLL) LR algorithm [36] have been considered almost exclusively for hardware implementation as a result of both the low average complexity and desirable numerical properties that are exhibited by the algorithm. The iterative nature and high hardware complexity that is required to implement certain operations of the CLLL algorithm, however, has remained a challenge for hardware realization.

The purpose of the dissertation research is to understand the complex interaction between CLLL algorithm and hardware considerations and then apply this knowledge toward the development of new CLLL-aided symbol detection algorithms and architectures. The culmination of the dissertation is a hardware realization of a specific

[1] LR algorithms can also be incorporated into the preprocessing part of SD algorithms to improve the BER performance and to reduce the overall complexity [45, 50, 80, 60].

modified LR algorithm and demonstration of this hardware realization in a practical wireless communication system. The contributions of this dissertation, however, are not limited only to this specific hardware realization. Throughout the dissertation, we focus on providing general analytic justification or extensive empirical justification for all algorithm modifications considered. Therefore, different combinations of the dissertation contributions can be applied to develop other LR-aided detection algorithms and architectures.

We begin this exploration in Chapter 2 by providing background knowledge about MIMO detection. The purpose of this chapter is to provide a foundation of understanding for subsequent algorithm modifications and architecture development. We accomplish this task by introducing a model for the channel behavior, defining the MIMO detection problem, reviewing metrics used to characterize MIMO detection algorithms, and reviewing existing detection methods. Concurrent with the presentation of this information, we gradually build the motivation for LR-aided detection hardware realizations.

In the next two chapters, we examine modifications to the CLLL algorithm from two perspectives. In Chapter 3, we explore modifications that reduce the complexity of the CLLL algorithm while preserving the desirable numerical behavior inherent in the CLLL algorithm. In Chapter 4 we examine how the more complex operations in the CLLL algorithm can be modified for streamlined hardware operation. Specifically, we develop optimized hardware modules for completing the division and inverse square root operations required by the CLLL algorithm.

We next explore hardware realization of this modified CLLL algorithm in Chapter 5. This exploration involves first applying the analysis from Chapter 3 to establish a set of design parameters. We then develop an architecture based on these parameters and the optimized hardware modules from Chapter 4. Finally, we evaluate the proposed architecture along the metrics of throughput, packet-processing latency, and

hardware resource usage.

In the final three chapters we consider additional modifications to the CLLL-aided detection algorithm utilized in the preceding chapters. Motivation for these additional modifications stems from an evaluation of the proposed architecture using a more realistic model channel model in Chapter 6. We discover that additional instances of the proposed architecture are required to maintain the packet-processing latency observed in Chapter 5. To solve this problem, we propose a merging of the LR processing and symbol-rate processing of LR-aided detection algorithms in Chapter 7. The utility of this merged algorithm is then clearly demonstrated in Chapter 8.

CHAPTER II

FUNDAMENTALS OF MIMO DETECTION

Considerable research has been completed by researchers in the area of efficient implementation of ML detection and suboptimal methods that map well to hardware. In this chapter, we first formalize the MIMO symbol detection problem and formally define concepts that we utilize throughout the dissertation. We then review detection methods and corresponding hardware realizations that have been explored by other researchers. An understanding of the deficiencies of these detection methods naturally leads to consideration of the dissertation focus, LR-aided detection algorithms. After developing a detailed understanding of the computations required for LR-aided detection algorithms, we are able to identify deficiencies in existing LR algorithm hardware realizations.

2.1 Formulation of MIMO Detection Problem

Consider a flat-fading system with N_t transmit antennas and N_r receive antennas. For MIMO transmissions, the data stream is divided into N_t sub-streams and transmitted through N_t antennas. Let $\boldsymbol{s} = [s_1, s_2, \ldots, s_{N_t}]^T \in \mathcal{S}^{N_t}$ represent the $N_t \times 1$ transmitted data vector at one time slot, where $\mathcal{S}$ is the constellation set of each element in $\boldsymbol{s}$, let $\boldsymbol{H}$ be the $N_r \times N_t$ channel matrix, and let $\boldsymbol{y} = [y_1, y_2, \ldots, y_{N_r}]^T$ denote the received signal at one time slot from N_r receive antennas. The input-output relationship for one time slot is

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{s} + \eta\boldsymbol{w}, \tag{1}$$

where $\eta = \sqrt{\frac{E[\boldsymbol{s}^{\mathcal{H}}\boldsymbol{s}]}{N_t}}$ and $\boldsymbol{w} = [w_1, w_2, \ldots, w_{N_r}]^T$ is a white Gaussian noise vector that has zero mean and covariance matrix $E\left[\boldsymbol{w}\boldsymbol{w}^{\mathcal{H}}\right] = \sigma_w^2 \boldsymbol{I}_{N_r}$. We can then define

the signal-to-noise-ratio (SNR) in decibels (dB) as the following:

Definition 1 (Signal-to-Noise-Ratio) *Given the* $\boldsymbol{w}$ *and* η *in the system model in* (1)*, the per-antenna signal-to-noise-ratio (SNR) is*

$$SNR = -10 \log_{10} \sigma_w^2. \tag{2}$$

We assume that the elements of $\boldsymbol{H}$ are independent identically distributed (i.i.d.) complex Gaussian distributed coefficients with zero mean and unit variance. We also assume that the noise variance σ_w^2 is known at the receiver and $\boldsymbol{H}$ is known at the receiver but unknown at the transmitter. Given this system model, detection is the process of determining an estimate $\hat{\boldsymbol{s}}$ of the symbol vector $\boldsymbol{s}$ that was sent based on knowledge of $\boldsymbol{H}$, $\boldsymbol{y}$, and σ_w^2.

The BER performance of detection is characterized by both coding gain and diversity order. Coding gain refers to the signal-to-noise-ratio (SNR) gap between two different detection methods at equal BER performance. Diversity order refers to the asymptotic negative logarithmic slope of the BER versus SNR curve:

Definition 2 (Diversity) *Suppose that the error probability* $P_e = P\{\hat{\boldsymbol{s}} \neq \boldsymbol{s}\}$ *and the signal-to-noise ratio is SNR. The diversity order of a given system is defined as*

$$G_d = \lim_{SNR \to \infty} -\frac{\log P_e}{\log SNR}. \tag{3}$$

From this definition it is clear that when two detection methods exhibit different diversity, the coding gain between these two detection methods increases with increasing SNR.

2.2 Optimal Detection Method

When all $\boldsymbol{s} \in \mathcal{S}^{N_t}$ are equally likely to be transmitted, the symbol detection method that minimizes P_e in Definition 2 for the system model in (1) is the ML detector.

This detector produces an estimate $\hat{\boldsymbol{s}}_{ML}$ that is based on an exhaustive search over all possible transmitted $(N_t \times 1)$-sized symbol vectors in $\mathcal{S}^{N_t}$:

$$\hat{\boldsymbol{s}}_{ML} = \underset{\tilde{\boldsymbol{s}} \in \mathcal{S}^{N_t}}{\arg\min} \|\boldsymbol{y} - \boldsymbol{H}\tilde{\boldsymbol{s}}\|^2. \tag{4}$$

The complexity of this detection method is exponential in the number of transmit antennas ($\mathcal{O}\left(|\mathcal{S}|^{N_t}\right)$). Hence, the exhaustive search method is not feasible for hardware realization when either a large signal constellation is employed (large $|\mathcal{S}|$) or large number of transmit antennas is employed. Instead, SD algorithms can be used to intelligently reduce the search space of the exhaustive method using a tree-search and can achieve ML or near-ML BER performance. SD algorithms based on best-first, depth-first, and breadth-first search methods have been considered for hardware implementation [5, 27, 3, 76]. These implementations, however, suffer from variable complexity as the result of changing channel conditions [5, 27], prohibitively large hardware requirements [3], or greatly reduced BER performance as the result of degraded diversity [76]. We also note that for all these methods the tree-search must be executed for each received $\boldsymbol{y}$ even when the channel matrix is unchanged over multiple received $\boldsymbol{y}$.

2.3 Low-Complexity Detection Methods

A variety of low-complexity MIMO detection methods have been explored, including minimum mean squared error (MMSE) and MMSE-assisted SIC methods. These methods usually involve preprocessing $\boldsymbol{H}$ and reusing this preprocessing result when $\boldsymbol{H}$ does not change over multiple received symbol vectors.

Based on the model in (1), the linear MMSE equalizer equation is

$$\boldsymbol{x}^{(\mathrm{MMSE})} = \left(\boldsymbol{H}^{\mathcal{H}}\boldsymbol{H} + \sigma_w^2 \boldsymbol{I}_{N_t}\right)^{-1} \boldsymbol{H}^{\mathcal{H}}\boldsymbol{y}. \tag{5}$$

The subsequent symbol detection step involves application of the quantization function $\mathcal{Q}_{\mathcal{S}}$, which quantizes each vector element to the closest symbol in $\mathcal{S}$. This gives

a detection result of $\hat{\boldsymbol{s}}^{(\text{MMSE})} = \mathcal{Q}_{\mathcal{S}}\left[\boldsymbol{x}^{(\text{MMSE})}\right]$. Note that $\boldsymbol{x}^{(\text{MMSE})}$ can also be found by defining

$$\bar{\boldsymbol{H}} = \begin{bmatrix} \boldsymbol{H} \\ \sigma_w \boldsymbol{I}_{N_t} \end{bmatrix} \qquad \bar{\boldsymbol{y}} = \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{0}_{N_t \times 1} \end{bmatrix} \tag{6}$$

and then computing the least-squares solution to the (over-constrained) extended system $\bar{\boldsymbol{H}}\boldsymbol{x}^{(\text{MMSE})} = \bar{\boldsymbol{y}}$ [24, 75]

$$\boldsymbol{x}^{(\text{MMSE})} = \left(\bar{\boldsymbol{H}}^{\mathcal{H}}\bar{\boldsymbol{H}}\right)^{-1}\bar{\boldsymbol{H}}^{\mathcal{H}}\bar{\boldsymbol{y}}. \tag{7}$$

We can determine the MMSE-SIC "solution" $\hat{\boldsymbol{s}}^{(\text{MMSE-SIC})}$ by first finding the QR-decomposition $\bar{\boldsymbol{H}} = \bar{\boldsymbol{Q}}\bar{\boldsymbol{R}}$, where $\bar{\boldsymbol{Q}}$ is an $(N_r + N_t) \times N_t$ matrix and $\bar{\boldsymbol{R}}$ is an $N_t \times N_t$ upper triangular matrix. Then we substitute this factorization into (7) and obtain the following:

$$\bar{\boldsymbol{R}}\boldsymbol{x}^{(\text{MMSE})} = \bar{\boldsymbol{Q}}^{\mathcal{H}}\bar{\boldsymbol{y}}. \tag{8}$$

Lastly, we let $\bar{\boldsymbol{b}} = \bar{\boldsymbol{Q}}^{\mathcal{H}}\bar{\boldsymbol{y}}$ and use

$$\hat{s}_n^{(\text{MMSE-SIC})} = \mathcal{Q}_{\mathcal{S}}\left[\frac{\bar{b}_n - \sum_{j=n+1}^{N_t}\bar{R}_{n,j}\hat{s}_j^{(\text{MMSE-SIC})}}{\bar{R}_{n,n}}\right] \tag{9}$$

to complete the MMSE-SIC detection.

The above detection methods simplify to zero-forcing and SIC detection, respectively, when the noise variance is not utilized in the detection process ($\sigma_w^2 = 0$).

As a result of the regular structure of these algorithms, these low-complexity methods are attractive for hardware realization, especially for the challenging 4×4 $\boldsymbol{H}$ case. Hardware realizations of the preprocessing part of these algorithms include the QR-decomposition implementations in [40, 61, 8, 63] and Direct Matrix Inversion (DMI) implementations in [31, 13]. Complete implementations of the MMSE and MMSE-SIC detection algorithms, including both the preprocessing and symbol vector processing, can be found in [52, 33, 6]. We note, however, that these hardware realizations are fundamentally limited by the achievable diversity of these low-complexity methods, which is only $N_r - N_t + 1$ [22, 43].

2.4 Lattice-Reduction-Aided Detection

Given that these low-complexity methods are attractive for hardware realization, it is desirable to explore modifications to these methods that restore full diversity order N_r. To achieve this goal, LR algorithms have been incorporated with these low-complexity methods [14, 43, 70, 75, 77, 64]. The utility of LR-aided detection algorithms compared to other detection methods is evident in BER simulation results of the system model in (1) for 4×4 64-QAM transmissions, which are contained in Figure 1. Examining this figure, we observe a concrete demonstration that LR-aided detection algorithms achieve the same diversity as the ML detector and exhibit superior BER performance compared to low-complexity detection methods. Hence, we are motivated to develop a detailed understanding of LR-aided detection methods such that we can achieve this desirable BER performance in practical hardware realizations.

2.4.1 High-level Description of LR-aided Detection

LR techniques involve preprocessing $\boldsymbol{H}$ to produce a reduced-lattice basis $\tilde{\boldsymbol{H}} = \boldsymbol{HT}$, where $\boldsymbol{T}$ is a unimodular matrix. This factorization allows us to rewrite the system in (1) as

$$\boldsymbol{y} = \boldsymbol{HT}(\boldsymbol{T}^{-1}\boldsymbol{s}) + \eta\boldsymbol{w} = \tilde{\boldsymbol{H}}\boldsymbol{z} + \eta\boldsymbol{w}. \tag{10}$$

The LR-aided detection process involves first finding an estimate $\hat{\boldsymbol{z}}$ of the transmitted symbol vector in the z-domain using linear detection or SIC. Then we determine $\hat{\boldsymbol{s}}$ by transforming each element of $\hat{\boldsymbol{z}}$ back to the original signal constellation using $\hat{\boldsymbol{s}} = \mathcal{Q}_{\mathcal{S}}\left[\boldsymbol{T}\hat{\boldsymbol{z}}\right]$.

To compute this reduced-lattice basis, a variety of LR algorithms can be employed, including Brun's algorithm [55], Seysen's algorithm [58] and the LLL algorithm [36]. The complex-valued QR-decomposition formulation of the LLL algorithm, the CLLL

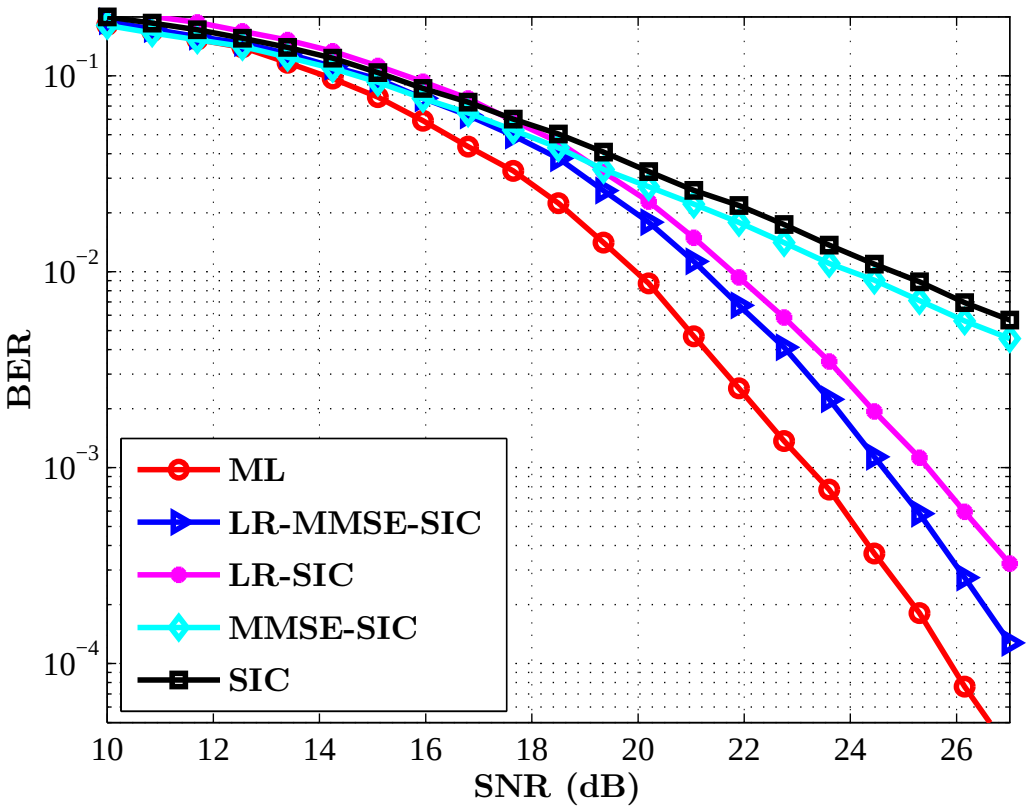

Figure 1: BER to SNR simulation results for 4×4 64-QAM transmissions and the system model in (1). A sufficient number of channel realizations is employed such that 8000 bit errors are generated per SNR value.

algorithm [14, 75], however, has been considered most frequently for hardware realization. Therefore we focus on understanding CLLL-aided detection algorithms.

The CLLL algorithm operates on the QR-decomposition of $\boldsymbol{H}$ to produce $\boldsymbol{T}$ and the QR-decomposition of $\tilde{\boldsymbol{H}}$. The $\tilde{\boldsymbol{H}} = \tilde{\boldsymbol{Q}}\tilde{\boldsymbol{R}}$ factorization returned by the CLLL algorithm satisfies the complex size reduction condition (11) and complex Lovász condition (12):

$$|\Re[\tilde{R}_{n,k}]| \leq \frac{1}{2}|\tilde{R}_{n,n}|, \ |\Im[\tilde{R}_{n,k}]| \leq \frac{1}{2}|\tilde{R}_{n,n}|, \ \ \forall\ 1 \leq n < k \leq N_t, \tag{11}$$

$$\delta|\tilde{R}_{k-1,k-1}|^2 \leq |\tilde{R}_{k,k}|^2 + |\tilde{R}_{k-1,k}|^2, \forall\ k \in [2, N_t], \tag{12}$$

where δ is a relaxation parameter that can be arbitrarily chosen from $(\frac{1}{2}, 1]$. To reduce

the complexity of the CLLL algorithm, the complex Lovász condition can be replaced with the Siegel condition [10, 1]:

$$|\tilde{R}_{k-1,k-1}|^2 \leq \zeta |\tilde{R}_{k,k}|^2, \forall\ k \in [2, N_t], \tag{13}$$

where ζ is chosen from $[2, 4]$.

Table 1 describes the operation of the original CLLL algorithm or the Siegel condition variant of the CLLL algorithm, depending on whether the complex Lovász condition or Siegel condition is used in Line 9, respectively. This listing forms the starting point for our algorithm modifications in Chapter 3.

2.4.2 Low-level Description of CLLL-aided Detection for QAM Constellation

Absent from the high-level description of LR-aided detectors are the details about how we determine $\hat{\boldsymbol{z}}$ when we utilize a given low-complexity method and signal constellation set. In this section, we consider the specific detection steps when the CLLL algorithm is utilized with the MMSE-SIC detector (CLLL-MMSE-SIC detection).

If the original signal constellation set consists of the infinite complex integer plane, then the signal constellation set in the z-domain also consists of the infinite complex integer plane. Therefore, during the initial detection step in the z-domain, the $\mathcal{Q}_{\mathcal{S}}$ function in (9) can be replaced with the element-wise integer-rounding operation. The signal constellation set for $\mathcal{M}$-ary QAM, however, is $\mathcal{S} = \{s \mid \Re[s], \Im[s] \in \mathcal{A}\}$, where $\mathcal{A} = \{-\sqrt{\mathcal{M}} + 1, ..., -1, 1, ...\sqrt{\mathcal{M}} - 1\}$. We are therefore motivated to reformulate (8) such that detection is carried out as if the real and imaginary parts of the original constellation set are drawn from the consecutive integers.

We define a new constellation set $\dot{\mathcal{S}} = \{\dot{s} = \frac{1}{2}(s + 1 + j) \,|s \in S\}$. The symbol vector in (1) can then be thought of as a symbol vector $\dot{\boldsymbol{s}} \in \dot{\mathcal{S}}^{N_t}$ that has been transformed by $2\dot{\boldsymbol{s}} - (1 + j)\,\mathbf{1}_{N_t \times 1}$. We can apply this idea to (8) by making the

substitution $\boldsymbol{x}^{(\mathrm{MMSE})} = 2\dot{\boldsymbol{x}}^{(\mathrm{MMSE})} - (1+j)\,\mathbf{1}_{N_t\times 1}$ and simplifying:

$$\bar{\boldsymbol{R}}\dot{\boldsymbol{x}}^{(\mathrm{MMSE})} = \frac{1}{2}\bar{\boldsymbol{Q}}^{\mathcal{H}}\left(\bar{\boldsymbol{y}} + \bar{\boldsymbol{H}}\,(1+j)\,\mathbf{1}_{N_t\times 1}\right). \tag{14}$$

This equation allows us to utilize the CLLL algorithm with the MMSE-SIC detector in (9). The CLLL algorithm returns the factorization $\bar{\boldsymbol{H}}\boldsymbol{T} = \tilde{\boldsymbol{Q}}\tilde{\boldsymbol{R}}$, where $\tilde{\boldsymbol{Q}}$ is an $(N_r + N_t) \times N_t$ matrix. Applying this relation to (14), we obtain the following:

$$\tilde{\boldsymbol{R}}\boldsymbol{T}^{-1}\dot{\boldsymbol{x}}^{(\mathrm{MMSE})} = \frac{1}{2}\left(\tilde{\boldsymbol{Q}}^{\mathcal{H}}\bar{\boldsymbol{y}} + \tilde{\boldsymbol{R}}\boldsymbol{T}^{-1}\,(1+j)\,\mathbf{1}_{N_t\times 1}\right). \tag{15}$$

Using the substitution $\boldsymbol{z} = \boldsymbol{T}^{-1}\dot{\boldsymbol{x}}^{(\mathrm{MMSE})}$ and partitioning $\tilde{\boldsymbol{Q}}$ into an $N_r \times N_t$ matrix $\tilde{\boldsymbol{Q}}^{(1)}$ and $N_t \times N_t$ matrix $\tilde{\boldsymbol{Q}}^{(2)}$ such that $\tilde{\boldsymbol{Q}} = \left[\left(\tilde{\boldsymbol{Q}}^{(1)}\right)^T \left(\tilde{\boldsymbol{Q}}^{(2)}\right)^T\right]^T$, we obtain the following:

$$\tilde{\boldsymbol{R}}\boldsymbol{z} = \frac{1}{2}\left(\left(\tilde{\boldsymbol{Q}}^{(1)}\right)^{\mathcal{H}}\boldsymbol{y} + \tilde{\boldsymbol{R}}\boldsymbol{T}^{-1}\,(1+j)\,\mathbf{1}_{N_t\times 1}\right), \tag{16}$$

which is in the same form as (8). We complete detection in the z-domain by first computing $\hat{\boldsymbol{z}}$ using (9) with the $\mathcal{Q}_{\mathcal{S}}$ function replaced with the integer-rounding function.[1] Then we determine the estimated symbol vector by computing $\hat{\boldsymbol{s}} = \mathcal{Q}_{\mathcal{S}}\left[2\boldsymbol{T}\hat{\boldsymbol{z}} - (1+j)\,\mathbf{1}_{N_t\times 1}\right]$. Hence, CLLL-MMSE-SIC detection for QAM requires that the CLLL algorithm generates the vector $\boldsymbol{T}^{-1}\,(1+j)\,\mathbf{1}_{N_t\times 1}$ in addition to $\tilde{\boldsymbol{Q}}^{(1)}$, $\tilde{\boldsymbol{R}}$, and $\boldsymbol{T}$.

2.4.3 CLLL Algorithm Operation

To generate these required matrices and the required $\boldsymbol{T}^{-1}\,(1+j)\,\mathbf{1}_{N_t\times 1}$ vector ($\boldsymbol{g}$ in Table 1), the CLLL algorithm operates by satisfying conditions (11) and (12) or (13) for progressively larger upper-left square sub-matrices of $\tilde{\boldsymbol{R}}$, iteratively updating the matrices and $\boldsymbol{g}$ as needed. The k variable in Table 1 indicates the size of the currently

[1] This is a sub-optimal detection step because the signal constellation in the z-domain does not consist of the infinite complex integer plane for (finite) $\mathcal{M}$-ary QAM. The resulting estimate in the z-domain may not correspond to a valid signal constellation, causing error propagation during the SIC detection process.

Table 1: Complex LLL Algorithm [43]

(1)	$[\tilde{\boldsymbol{Q}}, \tilde{\boldsymbol{R}}, \boldsymbol{T}] = \text{sorted QR}\ (\tilde{\boldsymbol{H}});\ \ k = 2;\ \boldsymbol{g} = (1+j)\,\mathbf{1}_{N_t \times 1};$
(2)	while $k \leq N_t$
(3)	for $n = k-1 : -1 : 1$
(4)	$u = \text{round}(\tilde{R}_{n,k} / \tilde{R}_{n,n})$
(5)	$\tilde{\boldsymbol{R}}_{1:n,k} = \tilde{\boldsymbol{R}}_{1:n,k} - u \cdot \tilde{\boldsymbol{R}}_{1:n,n}$
(6)	$\boldsymbol{T}_{:,k} = \boldsymbol{T}_{:,k} - u \cdot \boldsymbol{T}_{:,n}$
(7)	$g_n = g_n + u \cdot g_k$
(8)	end
(9)	if Cond. Fail $\begin{cases} \lvert\tilde{R}_{k-1,k-1}\rvert^2 > \zeta \lvert\tilde{R}_{k,k}\rvert^2 & \textbf{\textit{Siegel condition}} \\ \delta \lvert\tilde{R}_{k-1,k-1}\rvert^2 > \lvert\tilde{R}_{k,k}\rvert^2 + \lvert\tilde{R}_{k-1,k}\rvert^2 & \textbf{\textit{Lovász cond.}} \end{cases}$
(10)	$\boldsymbol{\Theta} = \frac{1}{\lVert\tilde{\boldsymbol{R}}_{k-1:k,k}\rVert} \begin{bmatrix} \tilde{R}^*_{k-1,k} & \tilde{R}_{k,k} \\ -\tilde{R}_{k,k} & \tilde{R}_{k-1,k} \end{bmatrix}$
(11)	$\tilde{\boldsymbol{R}}_{k-1:k,k-1:N_t} = \boldsymbol{\Theta} \tilde{\boldsymbol{R}}_{k-1:k,k-1:N_t}$
(12)	$\tilde{\boldsymbol{Q}}_{:,k-1:k} = \tilde{\boldsymbol{Q}}_{:,k-1:k} \boldsymbol{\Theta}^{\mathcal{H}}$
(13)	Swap $(k-1)$-th and k-th columns in $\tilde{\boldsymbol{R}}$ and $\boldsymbol{T}$
(14)	Swap $(k-1)$-th and k-th rows in $\boldsymbol{g}$
(15)	$k = \max(k-1, 2);$
(16)	else
(17)	$k = k + 1$
(18)	end
(19)	end

active upper-left square sub-matrix. When the CLLL algorithm is operating on a $k \times k$ sub-matrix, the size reduction condition is first forced true for the k-th column of $\tilde{\boldsymbol{R}}$ (Line 3-8). Each inner-loop iteration (Line 4-7) forces the size reduction condition true for a single element of this column and updates $\boldsymbol{T}$, the $\boldsymbol{g}$ vector (as shown in [54]), and the other elements of $\tilde{\boldsymbol{R}}$. Next, the Lovász condition or Siegel condition is checked in Line 9. If the condition is true, then the CLLL algorithm progresses to the $(k+1) \times (k+1)$ upper-left sub-matrix. If the condition is not satisfied, however, then a basis update (Line 10-14) is completed. The CLLL algorithm then restarts at the $(k-1) \times (k-1)$ upper-left sub-matrix. This process of incrementing and decrementing the active sub-matrix size continues until the entire $\tilde{\boldsymbol{R}}$ matrix satisfies

(11) and (12) or (13).

2.4.4 Existing Hardware Realization of LR Algorithms

The first implementation of an LR algorithm reported in literature is the Brun's LR algorithm implementation for MIMO precoding in [7]. This implementation features impressive throughput as the result of algorithm simplifications that reduce the complexity of the algorithm and make the algorithm more suitable for hardware realization. These simplifications, however, also reduce the BER performance considerably (5 dB gap in coding gain compared to when the CLLL algorithm is utilized [19]). Furthermore, the diversity under these simplifications has not been proven. The first implementations of LR algorithms that do not compromise BER performance include the CLLL algorithm field programmable gate array (FPGA) implementations in [19, 82] and a software implementation of Seysen's LR algorithm on a reconfigurable baseband processor in [72]. These implementations were then followed by implementations of CLLL algorithm variants in [1, 62, 69].

Lacking from these LR hardware contributions is a discussion of the complex interaction between LR algorithm and hardware considerations. As a result, LR algorithms have been modified without sufficient analytic justification. Instead, only system simulations have been used to justify design decisions. Additionally, the scalability of a subset of these implementations is not certain given that the implementations in [7, 72] require DMI as a preprocessing step and the algorithm implemented in [62] does not scale past 4×4 systems [66]. Hence, an open problem in LR research is how algorithm modifications to LR algorithms affect BER performance, operation under various channel conditions, and scalability for future generation wireless systems.

CHAPTER III

MODIFYING THE CLLL ALGORITHM FROM A FIXED-POINT PERSPECTIVE

Often algorithms, such as the LLL algorithm, are originally developed under the assumption that the variables involved in an algorithm are from an uncountable set of numbers or from an infinitely countable set of numbers. In a hardware or software realization of an algorithm, however, only finite hardware resources are available for both computing operations and storing algorithm variables and parameters. Therefore, a suitable finite-bit number representation must be chosen such that algorithm operation for a given target application under this number representation is not adversely affected. These finite-bit number representations are often classified as either fixed-point or floating-point, referring to the fixed or variable location of the decimal point.

Comparison of the specific details of fixed-point number representation to floating-point number representation reveals why fixed-point implementations are more amendable to hardware realization. A $(w+f)$-bit 2's complement fixed-point number x_{fix} having w bits for representing the integer part and f bits for representing the fractional part can be written as

$$x_{\text{fix}} = -2^{w-1}b_{w+f-1} + \sum_{i=0}^{w-2} 2^i b_{i+f} + 2^{-f}\sum_{i=0}^{f-1} 2^i b_i \tag{17}$$

$$= 2^{-f}\left(-2^{w+f-1}b_{w+f-1} + \sum_{i=0}^{w+f-2} 2^i b_i\right), \tag{18}$$

where b_i is the i-th bit of x_{fix}. As equation (18) suggests, we can implement fixed-point operations using digital hardware that operates on integers directly. This straightforward implementation is a natural consequence of the fixed location of the decimal

point.

Conversely, in floating-point number representation, the variable location of the decimal point naturally leads to additional hardware complexity. To observe this more concretely, we consider the form of a floating-point number. Often floating-point numbers are formed as a product of a sign term, a normalized mantissa term, and a power-of-two scaling term. For example, consider the case when the exponent of the scaling term is an M-bit 2's complement number E and mantissa term is a B-bit unsigned fixed-point number having 1 bit to represent the integer part and $B-1$ bits to represent the fractional part. A floating-point number x_{float} under this example can then be written as

$$x_{\text{float}} = (-1)^s \times \left(1 + 2^{-B+1} \sum_{i=0}^{B-2} 2^i b_i\right) \times 2^E, \tag{19}$$

where s is either 0 or 1 and b_i is the i-th bit of the mantissa term. From this equation, it is apparent that we cannot use an integer datapath immediately for computations on two numbers represented in this form. To illustrate this idea more concretely, we consider addition operations. The decimal points of the mantissa terms must first be aligned such that the exponents of the scaling terms are equal. The possibly shifted mantissas can then be added together using an integer datapath. Additional normalization steps are then required such that the result can be put back into the form of (19). Compared to fixed-point computations, both the alignment and normalization steps inherent in floating-point operations leads to additional hardware overhead in the form of barrel shifters. Additionally, the extra exponent term must be stored and updated accordingly, again leading to additional hardware overhead.

Given this qualitative comparison of fixed-point to floating-point number representations, we naturally desire to realize the CLLL algorithm in fixed-point hardware or software. Towards this goal, we first examine both the Lovász condition and Siegel condition variants of the CLLL algorithm from a fixed-point perspective. After finding that both CLLL variants exhibit desirable numerical properties, we consider

algorithm modifications that reduce the algorithm complexity and preserve these properties. Finally, after examining possible side-effects of these algorithm modifications for both CLLL variants, we conclude that the Siegel condition variant is more amendable for hardware realization given complexity considerations. Rigorous fixed-point magnitude analysis of the Siegel condition variant of the CLLL algorithm then forms the foundation for realizing the modified algorithm in hardware.

3.1 Numerical Properties of the CLLL Algorithm in MIMO Detection

3.1.1 Practical Bounds Inherent in the System Model

Based on the system model in (1) and extended system definition in (6), we can model the energy of the i-th $\bar{\boldsymbol{H}}$ column for a constant $\sigma_w = \sigma_{\max}$ using the following:

$$\|\bar{\boldsymbol{h}}_i\|^2 = \|\boldsymbol{h}_i\|^2 + \sigma_{\max}^2, \tag{20}$$

where $2\|\boldsymbol{h}_i\|^2$ is Chi-square distributed with degrees of freedom $2N_r$ and $\sigma_{\max}$ is the maximum value of σ_w.

Using the system model in (1) in a communication system simulation involves repeated i.i.d. trials of the random variable in (20). From an implementation perspective, it is instructive to consider how often this variable exceeds a particular magnitude when these variables are generated at a particular rate. We consider a general exercise that involves an IEEE 802.11n system that requires the processing of 128 MIMO channel matrices every 4 µs [23]. The results of this experiment are plotted in Figure 2 for the case $\sigma_{\max} = 0$ and a variety of channel matrix sizes.[1] Here we examine the expected time between overflow events when we adopt a particular number of integer bits (w in (18)) for fixed-point number representation of $\|\bar{\boldsymbol{h}}_i\|$. From this figure, it is clear that only a small number of integer bits is sufficient to

[1] We note here that for illustration purposes we also include results for channel matrix sizes larger than those specified in the 802.11n standard.

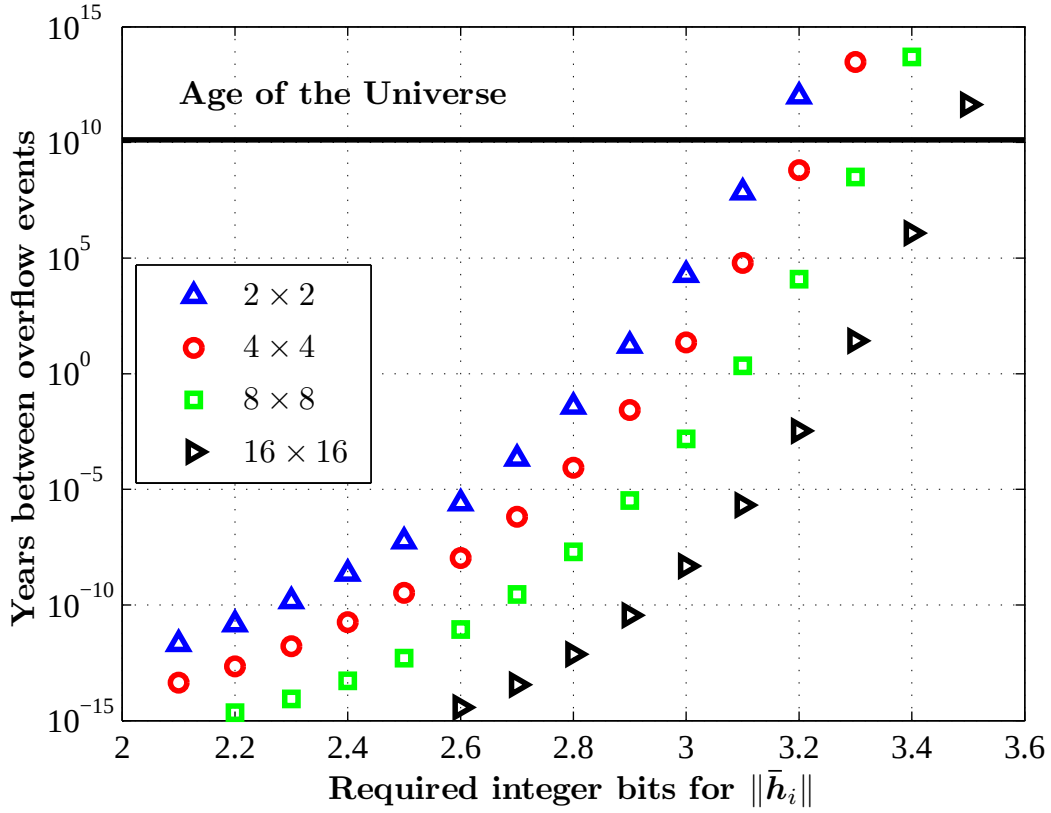

Figure 2: Expected number of years between overflow events when a particular number of integer bits is used to represent the square-root of the column energy of the extended channel matrix.

guarantee (practically) no occurrence of overflow events.

3.1.2 Application of Practical Bounds to CLLL

The QR-decomposition preprocessing step in Line 1 of Table 1 preserves the column energy of $\bar{\boldsymbol{H}}$. Additionally, the squared magnitudes of the $\tilde{\boldsymbol{R}}$ elements in the i-th column before the start of the CLLL algorithm are upper bounded by $\|\bar{\boldsymbol{h}}_i\|^2$. Therefore, to determine an upper bound B_{init} for the magnitudes of the $\tilde{\boldsymbol{R}}$ elements, we can consider the exercise in Section 3.1.1. For the case $N_r = N_t = 4$ and $\sigma_{\max} = 0.62$ (4.15 dB)[2], a choice of $B_{\text{init}} = 2^{2.82}$ yields one overflow event every 22.7 years.

[2]Below this SNR, the BER performance of MMSE-SIC is nearly identical to the BER performance of CLLL-MMSE-SIC for BPSK and 4/16/64-QAM.

Assuming that we adopt saturation quantization at the receiver, B_{init} safely upper bounds the elements of $\tilde{\boldsymbol{R}}$ at the start of the CLLL algorithm.

As a consequence of Lemma 1, the magnitudes of the $\tilde{\boldsymbol{R}}$ diagonal elements are upper bounded by B_{init} during operation of the CLLL algorithm. Lastly, recall that size reduction operations force the size reduction condition (11) true. It therefore follows that after size reduction on the k-th $\tilde{\boldsymbol{R}}$ column, the magnitudes of the real and imaginary parts of the off-diagonal elements in this column are upper bounded by $\frac{1}{2}B_{\text{init}}$.

Lemma 1 *During execution of the Lovász condition or Siegel condition variants of the CLLL algorithm, the maximum magnitude of the $\tilde{\boldsymbol{R}}$ diagonal elements does not increase.*

Proof: Notice that the CLLL algorithm only updates the diagonal elements of $\tilde{\boldsymbol{R}}$ when a basis update is required. Letting $\tilde{\boldsymbol{R}}'$ be the updated $\tilde{\boldsymbol{R}}$ matrix after the basis update and column swap, we can write the updated $(k-1)$-th and k-th diagonal elements as

$$|\tilde{R}'_{k-1,k-1}|^2 = |\tilde{R}_{k-1,k}|^2 + |\tilde{R}_{k,k}|^2, \tag{21}$$

$$|\tilde{R}'_{k,k}|^2 = \frac{|\tilde{R}_{k,k}|^2}{|\tilde{R}_{k-1,k}|^2 + |\tilde{R}_{k,k}|^2}|\tilde{R}_{k-1,k-1}|^2. \tag{22}$$

For the case of the Lovász condition CLLL algorithm variant, it follows that

$$|\tilde{R}'_{k-1,k-1}|^2 = |\tilde{R}_{k,k}|^2 + |\tilde{R}_{k-1,k}|^2 < \delta|\tilde{R}_{k-1,k-1}|^2 < |\tilde{R}_{k-1,k-1}|^2 \tag{23}$$

because $\frac{1}{2} \leq \delta < 1$.

For the case of the Siegel condition CLLL algorithm variant, it follows that

$$|\tilde{R}'_{k-1,k-1}|^2 \leq \frac{1}{2}|\tilde{R}_{k-1,k-1}|^2 + |\tilde{R}_{k,k}|^2 < \left(\frac{1}{2} + \frac{1}{\zeta}\right)|\tilde{R}_{k-1,k-1}|^2 \tag{24}$$

because $\zeta \geq 2$ and at Line 11 the $\tilde{\boldsymbol{R}}_{k-1,k}$ element satisfies the size reduction condition (11).

For either CLLL algorithm variant, it is clearly apparent from equation (22) that $|\tilde{R}'_{k,k}|^2 \leq |\tilde{R}_{k-1,k-1}|^2$. ■

3.2 Modifying the CLLL Algorithm

These numerical properties—tightly bounded $\tilde{\boldsymbol{R}}$ diagonal elements and tightly bounded size reduction operation results—exhibited by the CLLL algorithm are desirable in fixed-point hardware realization. Modifications to the CLLL algorithm should therefore be evaluated on both algorithm complexity reduction and preservation of these numerical properties in addition to BER performance.

3.2.1 Motivation

An algorithm modification in [37] resulted in the Effective LLL algorithm. Essentially, only the first inner-loop iteration (Line 4-7) is executed during each outer-loop iteration. For LLL-SIC detectors it is asserted in [37] that the symbol vector estimate is unchanged by this modification. Given that only an intuitive explanation is given in [37], we consider the following proposition:

Proposition 1 *Let $\tilde{\boldsymbol{R}}$, $\boldsymbol{T}$, and $\tilde{\boldsymbol{Q}}$ be the matrices obtained when the CLLL algorithm is run to completion* or *early terminated on any iteration. Let $\hat{\boldsymbol{s}}$ be the symbol vector estimate when these matrices are used for CLLL-MMSE-SIC detection. If full size reduction is completed on the $\tilde{\boldsymbol{R}}$ matrix, then the symbol vector estimate when $\tilde{\boldsymbol{Q}}$ and the updated $\tilde{\boldsymbol{R}}$ and $\boldsymbol{T}$ matrices are used for CLLL-MMSE-SIC detection is also $\hat{\boldsymbol{s}}$.*

Proof: See Appendix A.

In addition to providing justification of the Effective LLL algorithm, this proposition also indicates that the symbol vector estimate is unaffected by size reduction operations on the $\tilde{\boldsymbol{R}}_{k-1,k}$ elements for $2 \leq k \leq N_t$.

Although the Effective LLL algorithm has lower complexity than the original CLLL algorithm, the $\tilde{\boldsymbol{R}}$ elements that are not size reduced are allowed to increase

uncontrollably. Since this behavior is unacceptable in a fixed-point hardware implementation, we are motivated to modify the Effective LLL algorithm.

3.2.2 Relaxing the Size Reduction

Instead, we propose to *relax* the size reduction condition on all elements by associating a size reduction parameter $\phi_{n,k} \geq \frac{1}{2}$ with each of these elements:

Definition 3 (Relaxed Size Reduction Condition) *The $\tilde{R}_{n,k}$ element for $n < k$ satisfies a relaxed size reduction condition, which is defined by $\phi_{n,k} \geq \frac{1}{2}$, if $|\Re[\tilde{R}_{n,k}]| \leq \phi_{n,k}|\tilde{R}_{n,n}|$ and $|\Im[\tilde{R}_{n,k}]| \leq \phi_{n,k}|\tilde{R}_{n,n}|$ is satisfied.*

We therefore modify the CLLL algorithm such that during each k iteration, size reduction is performed on entries $\tilde{R}_{n,k}$ for $n < k-1$ (Line 4-7) only when this definition is not satisfied. By choosing larger ϕ's we can decrease how often size reduction is performed on these elements, decreasing the algorithm complexity while maintaining bounded size reduction results.

The motivation for the relaxation on the size reduction conditions for $n = k-1$ is not necessarily reduction of algorithm complexity. Instead, later in Section 4.1.1.3 we utilize relaxation of these elements to simplify the design requirements for the integer-rounded divider. Since the $\tilde{\boldsymbol{R}}_{k-1,k}$ elements influence the behavior of the $\tilde{\boldsymbol{R}}$ diagonal elements, we must examine this relaxation more carefully.

3.2.3 Effect of Relaxation on Lovász Condition CLLL Algorithm Variant

We consider operation of the CLLL algorithm in Table 1 when a relaxed size reduction condition, defined by the $\phi_{n,k}$'s, is adopted. Specifically, we only execute Line 4-7 if the $\tilde{R}_{n,k}$ element does not satisfy Definition 3 and run the modified algorithm to completion. Since the resulting $\tilde{\boldsymbol{R}}$ matrix satisfies Definition 3 and (12), the following is true for $2 \leq k \leq N_t$:

$$\left(\delta - 2\phi_{k-1,k}^2\right) |\tilde{R}_{k-1,k-1}|^2 \leq |\tilde{R}_{k,k}|^2. \tag{25}$$

To produce an upper-triangle matrix $\tilde{\boldsymbol{R}}^{\diamond}$ that satisfies the full size reduction condition in (11), we can execute the full size reduction procedure used in Lemma 6 (see Appendix A) on $\tilde{\boldsymbol{R}}$. If during this procedure the $\tilde{R}_{k-1,k}$ element requires size reduction (on either the real or imaginary part), then the magnitude of the $\tilde{R}_{k-1,k}$ element will decrease. Hence, the state of the Lovász condition will become uncertain. We therefore consider the worst case when all $\tilde{R}_{k-1,k}$ elements for $2 \leq k \leq N_t$ require size reduction. For this case the following is true (likewise for $|\Im[\tilde{R}_{k-1,k}]|$):

$$\frac{1}{2}|\tilde{R}_{k-1,k-1}| < |\Re[\tilde{R}_{k-1,k}]| \leq \phi_{k-1,k}|\tilde{R}_{k-1,k-1}|. \tag{26}$$

Assuming that we choose $\phi_{k-1,k}$'s less than 1, we see that $|\Re[\tilde{R}^{\diamond}_{k-1,k}]|$ satisfies (likewise for the $|\Im[\tilde{R}^{\diamond}_{k-1,k}]|$)

$$|\Re[\tilde{R}^{\diamond}_{k-1,k}]| = |\tilde{R}_{k-1,k-1}| - |\Re[\tilde{R}_{k-1,k}]|. \tag{27}$$

Combining this result with the inequalities in (26), we obtain

$$2(1-\phi_{k-1,k})^2|\tilde{R}_{k-1,k-1}|^2 \leq |\tilde{R}^{\diamond}_{k-1,k}|^2. \tag{28}$$

Finally, by adding the inequality in (28) to the inequality in (25), observing that the diagonal elements of $\tilde{\boldsymbol{R}}$ are unchanged by full size reduction, we see that

$$\left(\delta - 4\left(\phi_{k-1,k} - \frac{1}{2}\right)\right)|\tilde{R}^{\diamond}_{k-1,k-1}|^2 \leq |\tilde{R}^{\diamond}_{k,k}|^2 + |\tilde{R}^{\diamond}_{k-1,k}|^2. \tag{29}$$

Hence, $\tilde{\boldsymbol{R}}^{\diamond}$ satisfies both the full size reduction condition in (11) and the Lovász condition in (12) that has the δ parameter replaced with $\delta^{\diamond} = \delta - 4\left(\phi_{k-1,k} - \frac{1}{2}\right)$.

If we choose the $\phi_{k-1,k}$'s such that $\delta^{\diamond}$ is greater than $\frac{1}{2}$, then the diversity proof in [43] remains valid under this size reduction condition relaxation. Additionally, the δ parameter in the CLLL algorithm controls the BER performance for CLLL-aided detectors [43]. Specifically, as the δ parameter is increased, the BER performance improves while the complexity increases. Therefore when we use $\tilde{\boldsymbol{R}}^{\diamond}$ (with the corresponding $\boldsymbol{T}^{\diamond}$) in CLLL-MMSE detection, BER performance degradation will come in

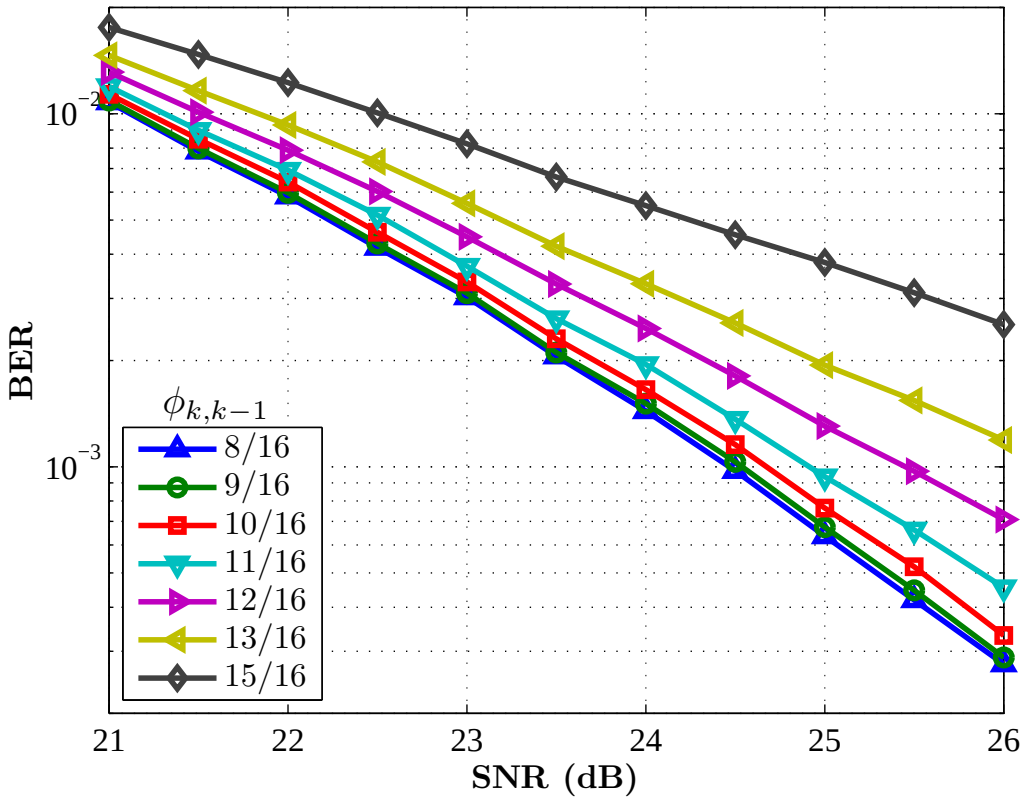

Figure 3: Effect of relaxation of the $\phi_{k-1,k}$'s size reduction conditions on the Lovász condition CLLL algorithm variant. For all simulations, a 4×4 64-QAM system is considered and a choice of $\delta = \frac{3}{4}$ is used for the CLLL-MMSE-SIC algorithm. A sufficient number of channel realizations is considered to generate a minimum of 3000 bit errors per SNR value.

the form of an increase in the coding gain gap to ML BER performance. Based on the conclusion in Appendix A, this result is also true when we use $\tilde{\boldsymbol{R}}$ in CLLL-MMSE-SIC detection.

To demonstrate that simulation results are consistent with this analytic result, we simulate a 4×4 64-QAM system with a choice of $\delta = \frac{3}{4}$ for the CLLL-MMSE-SIC algorithm. The result of this experiment for various choices of uniform $\phi_{k-1,k}$'s is shown in Figure 3. From this figure we see that for $\phi_{k-1,k} = 9/16$, full-diversity detection is achieved and negligible reduction in BER performance occurs. When the CLLL-MMSE-SIC algorithm is run for larger $\phi_{k-1,k}$'s, however, the BER performance

degrades rapidly.

3.2.4 Effect of Relaxation on Siegel Condition CLLL Algorithm Variant

We again consider operation of the CLLL algorithm in Table 1 when a relaxed size reduction condition, defined by the $\phi_{n,k}$'s, is adopted. In this exercise, however, we consider the behavior of the Siegel condition variant of the CLLL algorithm.

During a basis update the $\tilde{\boldsymbol{R}}_{k,k-1}$ element, which becomes the k-th $\tilde{\boldsymbol{R}}$ diagonal after the column swap in Line 13, is updated at Line 11 such that the magnitude of this element is less than or equal to the magnitude of the $\tilde{\boldsymbol{R}}_{k-1,k-1}$ element (Lemma 1). The $\tilde{\boldsymbol{R}}_{k-1,k}$ element, which becomes the $(k-1)$-th $\tilde{\boldsymbol{R}}$ diagonal after the column swap, is updated such that the squared magnitude is equal to $|\tilde{\boldsymbol{R}}_{k,k}|^2 + |\tilde{\boldsymbol{R}}_{k-1,k}|^2$. Applying the failed Siegel condition and the relaxed size reduction condition, we see that the following is true:

$$|\tilde{\boldsymbol{R}}_{k,k}|^2 + |\tilde{\boldsymbol{R}}_{k-1,k}|^2 < \left(\frac{1}{\zeta} + 2\phi_{k-1,k}^2\right) |\tilde{\boldsymbol{R}}_{k-1,k-1}|^2. \tag{30}$$

Given that the size reduction condition on the $\tilde{\boldsymbol{R}}_{k-1,k}$ element is relaxed, it is now possible that a basis update could increase the maximum squared magnitude of the $\tilde{\boldsymbol{R}}$ diagonal elements by a factor of $\left(\frac{1}{\zeta} + 2\phi_{k-1,k}^2\right)$. Assuming that the maximum number of basis updates is G and that all $\phi_{k-1,k}$'s are the same, we define a new upper bound for the magnitude of the $\tilde{\boldsymbol{R}}$ diagonal elements:

$$B = \begin{cases} B_{\text{init}} & \frac{1}{\zeta} + 2\phi_{k-1,k}^2 \leq 1, \\ \left(\frac{1}{\zeta} + 2\phi_{k-1,k}^2\right)^{\frac{G}{2}} B_{\text{init}} & \frac{1}{\zeta} + 2\phi_{k-1,k}^2 > 1. \end{cases} \tag{31}$$

3.2.5 Bounding the Intermediate Size Reduction Result for Hardware Implementation

Given that the Siegel condition variant of the CLLL algorithm exhibits reduced complexity compared to the Lovász condition variant [10, 1], we focus on examining the behavior of the size reduction operations for the Siegel condition variant. To utilize

Definition 3 in the CLLL algorithm, we must have an understanding of how large intermediate size reduction results can grow during size reduction on an $\tilde{\boldsymbol{R}}$ column (Line 5).

In the modified CLLL algorithm with a relaxed size reduction condition, a size reduction operation is required for each inner-loop iteration (Line 4-7) only if the $\tilde{R}_{n,k}$ element in Line 5 does not satisfy Definition 3. As an additional modification toward a fixed-point hardware implementation, we force a size reduction operation each inner-loop iteration when the magnitude of the real or imaginary part of the $\tilde{R}_{n,k}$ element in Line 5 exceeds $\frac{1}{2}B$. For simplicity we also assume that all $\phi_{n,k}$'s for $n = k - 1$ in Definition 3 are equal.

We now examine how the $\tilde{\boldsymbol{R}}$ elements can increase during a relaxed size reduction process that includes the enforcement of the absolute $\frac{1}{2}B$ upper bound. Here, we simultaneously generalize and simplify our original analysis in [20] to accommodate Definition 3. We also note that for fixed B, the following analysis is independent of the channel model.

For size reduction operations on the k-th column, we let $\tilde{R}'_{n,k}$ represent the intermediate value of $\tilde{R}_{n,k}$ after the first $(k - n - 1)$ inner-loop iterations but before size reduction on the n-th row element of the k-th column. We also let $u_{l,k}$ be equal to 0 when execution of the $(k - l)$-th inner-loop iteration is not required ($\tilde{R}_{n,k}$ at Line 4 satisfies Definition 3 and the $\frac{1}{2}B$ upper bound) and be equal to the u value in Line 4 at the $(k - l)$-th inner-loop iteration when execution of this inner-loop iteration is required. We can then write the intermediate size reduction result as

$$\tilde{R}'_{n,k} = \tilde{R}_{n,k} - \sum_{l=n+1}^{k-1} u_{l,k}\tilde{R}_{n,l}. \tag{32}$$

The summation on the righthand side involves $\tilde{R}_{n,l}$ elements, which are the results of size reduction operations during previous outer-loops (when the CLLL algorithm was operating on upper-left square matrices smaller than $k \times k$). By applying Definition

3 to these $\tilde{R}_{n,l}$ elements, we can upper bound the magnitude of the real component of $\tilde{R}'_{n,k}$ in (32) by

$$\left|\Re[\tilde{R}_{n,k}]\right| + \sum_{l=n+1}^{k-1} \left(|\Re[u_{l,k}]| + |\Im[u_{l,k}]|\right) \phi_{n,l} \left|\tilde{R}_{n,n}\right|. \tag{33}$$

To remove the dependence of (33) on the $u_{l,k}$'s, we notice that given our definition of the $\tilde{R}'_{n,k}$ elements, the enforcement of the absolute $\frac{1}{2}B$ upper bound results in the following relation:

$$\left|\Re[\tilde{R}'_{l,k}] - \Re[u_{l,k}]\tilde{R}_{l,l}\right| < \frac{1}{2}B. \tag{34}$$

Using signed-magnitude techniques, we can write this as

$$|\Re[u_{l,k}]| \left|\tilde{R}_{l,l}\right| < \frac{1}{2}B + \left|\Re[\tilde{R}'_{l,k}]\right|. \tag{35}$$

To remove the dependence of (35) on $\tilde{R}_{l,l}$, we notice that the induction proof found in [43, Appendix B] that has been slightly altered to accommodate the Siegel condition (13) results in

$$\left|\tilde{R}_{n,n}\right| < \zeta^{\frac{l-n}{2}} \left|\tilde{R}_{l,l}\right|. \tag{36}$$

Finally, substitution of (36) into (33) followed by substitution of (35) results in the following:

$$\begin{aligned} \left|\Re[\tilde{R}'_{n,k}]\right| < &\left|\Re[\tilde{R}_{n,k}]\right| + \\ &\sum_{l=n+1}^{k-1} \zeta^{\frac{l-n}{2}} \phi_{n,l} \left(B + \left|\Re[\tilde{R}'_{l,k}]\right| + \left|\Im[\tilde{R}'_{l,k}]\right|\right). \end{aligned} \tag{37}$$

We obtain a similar upper bound on the magnitude of $\Im[\tilde{R}'_{n,k}]$ by repeating steps (33)-(37).

We can first trivially determine these upper bounds for $n = k - 1$ and then recursively substitute as we determine these upper bounds for smaller n. Assuming

that the $\phi_{n,k}$'s do not change during execution of the algorithm, we obtain

$$\left|\Re[\tilde{R}'_{n,k}]\right| < \gamma_{n,k}B + \left|\Re[\tilde{R}_{n,k}]\right| + \sum_{p=n+1}^{k-1} \alpha_{p,k}\left(\left|\Re[\tilde{R}_{p,k}]\right| + \left|\Im[\tilde{R}_{p,k}]\right|\right), \tag{38}$$

where the $\alpha_{p,k}$'s and $\gamma_{n,k}$'s are determined during the recursive substitution process.

Note that at the end of an outer-loop for a particular k, $\left|\tilde{R}_{p,k}\right|$ is upper bounded by B for $p = k-1$ as a result of possible basis updates[3] and by $B/\sqrt{2}$ for $p \neq k-1$. Therefore, the maximum energy that could be re-distributed among the $\tilde{\boldsymbol{R}}_{1:k-1,k}$ sub-vector elements as the result of subsequent basis updates (as Siegel conditions fail and the CLLL algorithm operates on smaller matrix sizes) is the following:

$$\sum_{p=1}^{k-1} |\tilde{R}_{p,k}|^2 \leq B^2\left(1 + \frac{k-2}{2}\right). \tag{39}$$

To maximize the righthand side in (38), we assume that subsequent basis updates distribute the energy among the sub-vector elements to maximize the upper bound. Solving this constrained maximum problem (see Appendix B), we obtain

$$\left|\Re[\tilde{R}'_{n,k}]\right| < B\left(\gamma_{n,k} + \sqrt{k\left(\frac{1}{2} + \sum_{p=n+1}^{k-1} \alpha_{p,k}^2\right)}\right) \tag{40}$$

and reach a similar upper bound for the imaginary components. By designing hardware around these upper bounds, we can safely utilize variants of the Effective CLLL algorithm in fixed-point implementations.

[3]The magnitude of $\tilde{R}_{k-1,k}$ is upper bounded by $B/\sqrt{2}$ when no basis update occurs. If a basis update occurs, then this upper bound is B.

CHAPTER IV

MODIFYING THE CLLL ALGORITHM BASED ON DATAPATH CONSIDERATIONS

In the previous chapter we focus on fixed-point modifications that are relevant for both software and hardware implementations. In this chapter, however, we consider more hardware-focused CLLL algorithm modifications. Specifically, we develop and analyze a hardware datapath for the division operation required by the size reduction operation and a hardware datapath for handling the basis update operation. Although these two datapath exercises appear to be disparate, we reach the same conclusion about either datapath exercise: Applied understanding of the CLLL algorithm naturally leads to additional algorithm modifications that we can use to simplify the seemingly complex datapaths requirements.

4.1 Handling the Size Reduction Condition

Each size reduction operation requires the computation of an integer-rounded quotient (Line 4). This computation, however, can often be avoided by noticing that $|\Re[\tilde{R}_{n,k}]| < \frac{1}{2}|\tilde{R}_{n,n}|$ implies $\Re[u] = 0$ and $\frac{1}{2}|\tilde{R}_{n,n}| \leq |\Re[\tilde{R}_{n,k}]| < \frac{3}{2}|\tilde{R}_{n,n}|$ implies $|\Re[u]| = 1$. We handle the case $|\Re[u]|, |\Im[u]| > 1$ in [16] using an integer-rounded divider based on a single Newton-Raphson (NR) iteration, taking advantage of the divisor reuse inherent in the CLLL algorithm. Assuming reciprocals are buffered, this reciprocation-based approach is also useful for the subsequent SIC detection step because the stored reciprocals can be used for the SIC recursion in (9). Here we adopt our original basic architecture but alter it slightly based on our algorithm modifications in Chapter 3.

4.1.1 Single Newton-Raphson Iteration for Integer-rounded Division

We first consider a more formal description of the Newton-Raphson iteration-based division method. To compute n/d for $n > 0$ and $d > 0$, we first normalize d such that $d2^{\psi} = d_n$, where $1 \leq d_n < 2$. An estimate r'_n of the reciprocal of d_n is then computed from an initial estimate $r_n + \epsilon$, which is often obtained from a look-up table (LUT), using the following equation:

$$r'_n = (r_n + \epsilon)\left(2 - d_n\left(r_n + \epsilon\right)\right), \tag{41}$$

where $r_n = 1/d_n$ and ϵ is the error of the initial estimate.

4.1.1.1 Fixed-Point Formulation

To alter this expression for a fixed-point hardware implementation, we first introduce the notation $\{w.f\}$ to indicate an unsigned number representation having w integer bits and f fraction bits, let the function $\mathcal{Q}_1$ indicate truncation quantization to $\{2.(f+1)\}$, and let the function $\mathcal{Q}_2$ indicate truncation quantization to $\{2.f\}$. We also allow an additional LUT for $(r_n + \epsilon)^2$. Assuming that n has $\{w_n.f\}$ representation, d has $\{w.f\}$ representation, and the $(r_n + \epsilon)$ LUT has $\{1.a\}$ representation, we then update (41) to the following:

$$r'_n = 2\left(r_n + \epsilon\right) - \mathcal{Q}_2\left[\left(\mathcal{Q}_1\left[d_n + 2^{-(f+1)}\right]\right)\left(r_n + \epsilon\right)^2\right] - 2^{-f}. \tag{42}$$

Given that d_n has $\{1.f + w - 1\}$ representation, the $\mathcal{Q}_1$ function and $2^{-(f+1)}$ term together introduce an error ϵ_1 bounded by $2^{-(w+f-1)} \leq \epsilon_1 \leq 2^{-(f+1)}$. Given that $(r_n + \epsilon)^2$ has $\{1.2a\}$ representation, the $\mathcal{Q}_2$ introduces an error ϵ_2 bounded by $-2^{-f} + 2^{-(f+1+2a)} \leq \epsilon_2 \leq 0$. These observations allow us to rewrite (42) as

$$\begin{aligned} r'_n &= 2\left(r_n + \epsilon\right) - \left(\left(d_n + \epsilon_1\right)\left(r_n + \epsilon\right)^2 + \epsilon_2\right) - 2^{-f} \\ &= r_n + \epsilon', \end{aligned} \tag{43}$$

where $\epsilon' = -\left(d_n\epsilon^2 + \epsilon_1(r_n + \epsilon)^2 + \epsilon_2 + 2^{-f}\right)$. By considering the bounds of ϵ_1 and ϵ_2, we see that for this particular fixed-point formulation $\epsilon' < 0$.

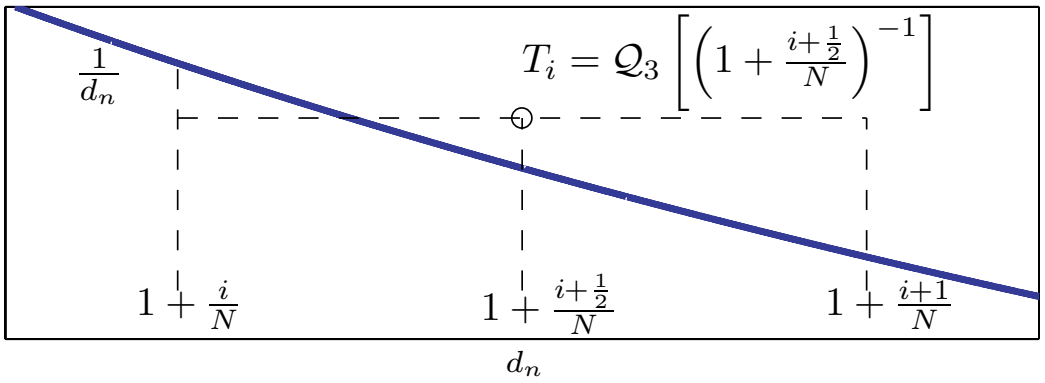

Figure 4: Enlarged view of the d_n values that map into the i-th entry of the $(r_n + \epsilon)$ LUT. The function $\mathcal{Q}_3$ represents truncation quantization to $\{1.a\}$.

4.1.1.2 Relative Error Analysis

The magnitude of the relative error, which is the absolute value of the ratio of the absolute quotient error to the quotient, can be upper bounded by setting each fixed-point error term in ϵ' to the maximum possible value:

$$\left|\frac{n\epsilon' 2^{\alpha}}{n r_n 2^{\alpha}}\right| = d_n|\epsilon'| < d_n\left(d_n\epsilon^2 + 2^{-f} + \left(\frac{1}{d_n} + \epsilon\right)^2 2^{-(f+1)}\right). \tag{44}$$

Assuming f is constant, the maximum value of this upper bound is only a function of LUT length N and LUT entry number representation $\{1.a\}$. A standard method for indexing into a LUT of length $N = 2^{\gamma}$ involves using the first γ fraction bits of d_n as the index i into the LUT. Each LUT entry T_i is then ${d'_n}^{-1}$ quantized to $\{1.2a\}$ by the function $\mathcal{Q}_3$, where d'_n is equal to the midpoint value of the d_n values that map to index i [11]. Figure 4 illustrates this convention.

Adopting this convention, we may express the upper bound of (44) as a piece-wise function of d_n consisting of N functions $f_i(d_n)$, where

$$f_i(d_n) = {d_n}^2\left(T_i - \frac{1}{d_n}\right)^2 + d_n\left(2^{-f} + {T_i}^2 2^{-(f+1)}\right). \tag{45}$$

Since the second derivative of f_i with respect to d_n is always positive, it follows that the limit of the maximum value of f_i must occur at $d_n = 1 + \frac{i}{N}$ or $d_n = 1 + \frac{i+1}{N}$.

Therefore, to compute an upper bound $M_{\gamma,a}$ for (44) given a table length $N = 2^{\gamma}$ and LUT number representation $\{1.a\}$, we evaluate

$$M_{\gamma,a} = \max_{i=0\ldots 2^{\gamma}-1} \left(\max \left(f_i \left(1 + \frac{i}{2^{\gamma}} \right), f_i \left(1 + \frac{i+1}{2^{\gamma}} \right) \right) \right). \tag{46}$$

We demonstrate the utility of the $M_{\gamma,a}$ upper bound for an example system in Appendix C. An additional optimization not considered here then involves simultaneously increasing the number of d_n truncated bits and the corresponding bias term in (42).

4.1.1.3 Integer-Rounding Considerations

The inherent assumption of the relative error analysis is that we compute $n2^{\psi}r'_n$ to full precision. Since we only concern ourselves with the integer-rounded quotient, this precision is unnecessary. The product nr'_n requires w_n integer bits and $2f$ fraction bits for full-precision representation. We round up $n2^{\psi}r'_n$ when the following inequality is satisfied:

$$\sum_{i=0}^{2f-\psi-1} b_i 2^{(-2f+i+\psi)} \geq \frac{1}{2}, \tag{47}$$

where b_i is the i-th bit of the nr'_n product. Conversely, we return the integer part of $n2^{\psi}r'_n$ when the condition in (47) is not satisfied.

We can use the condition of $b_{(2f-\psi-1)}$ to determine the state of the condition in (47). When $b_{(2f-\psi-1)} = 1$ we have the following:

$$\sum_{i=0}^{2f-\psi-1} b_i 2^{(-2f+i+\psi)} = \frac{1}{2} + \sum_{i=0}^{2f-\psi-2} b_i 2^{(-2f+i+\psi)} \geq \frac{1}{2}, \tag{48}$$

indicating that (47) is true. When $b_{(2f-\psi-1)} = 0$ we have the following:

$$\sum_{i=0}^{2f-\psi-1} b_i 2^{(-2f+i+\psi)} = \sum_{i=0}^{2f-\psi-2} b_i 2^{(-2f+i+\psi)} \leq \left(\frac{1}{2} - 2^{-2f+\psi} \right) < \frac{1}{2}, \tag{49}$$

indicating that (47) is false.

Since 2^{ψ} normalizes d, which has $\{w.f\}$ representation, the maximum value of ψ is f. Therefore we only need compute the first $f+1$ fraction bits (to the right of the decimal point) of nr'_n.

4.1.2 Exploiting the Relaxed Size Reduction Condition

Before utilizing the upper bound $M_{\gamma,a}$ to exploit the relaxed size reduction condition in Chapter 3, we prove a necessary lemma. Letting $q = n/d$ and $q' = n2^{\psi}r'_n$, we consider the following:

Lemma 2 *For the fixed-point Newton-Raphson formulation given in (42), the following is true:*

$$-\left(\frac{1}{2}+\epsilon_q\right) < \xi < \frac{1}{2}, \tag{50}$$

where $\epsilon_q = q - q'$ and $\xi = \lfloor q' \rceil - q$.

Proof: Since $\epsilon' < 0$, it follows that $\epsilon_q > 0$. We see that $\lfloor q' \rceil$ satisfies $q' - \frac{1}{2} < \lfloor q' \rceil \leq q' + \frac{1}{2}$, which can be rewritten as $-\left(\frac{1}{2}+\epsilon_q\right) < \xi \leq \frac{1}{2} - \epsilon_q$. Since the upper bound of this interval is $\frac{1}{2}$, ξ satisfies (50). ■

Then as an initial step toward exploiting the relaxed size reduction in the integer-rounded divider, we consider the following lemma, which concerns the integer-rounded quotients computed from full-precision quotients:

Lemma 3 *Given that the fixed-point Newton-Raphson in (42) that has maximum relative error $M_{\gamma,a}$ is used to compute the u for size reduction on the $\tilde{R}_{n,k}$ entry (Line 4, Table 1), the relaxed size condition for this entry will always be satisfied if the following is true:*

$$\left|\frac{\Re[\tilde{R}_{n,k}]}{\tilde{R}_{n,n}}\right|, \left|\frac{\Im[\tilde{R}_{n,k}]}{\tilde{R}_{n,n}}\right| < \frac{\phi_{n,k} - \frac{1}{2}}{M_{\gamma,a}}, \tag{51}$$

where $\phi_{n,k} \geq \frac{1}{2}$ is the relaxed size condition factor associated with the $\tilde{R}_{n,k}$ entry.

Proof: Let u' be the integer-rounded quotient produced by using the fixed-point Newton-Raphson formulation. If we set $n = |\Re[\tilde{R}_{n,k}]|$ and $d = |\tilde{R}_{n,n}|$, then $|\Re[u']| = \lfloor q' \rceil$. We see that Lemma 2 implies the following:

$$-\left(\frac{1}{2} + \epsilon_q\right) + \left|\frac{\Re[\tilde{R}_{n,k}]}{\tilde{R}_{n,n}}\right| < |\Re[u']| < \left|\frac{\Re[\tilde{R}_{n,k}]}{\tilde{R}_{n,n}}\right| + \frac{1}{2}. \tag{52}$$

The absolute error ϵ_q is upper bounded by $\left|\frac{\Re[\tilde{R}_{n,k}]}{\tilde{R}_{n,n}}\right| M_{\gamma,a}$, which according to (51) is upper bounded by $\left(\phi_{n,k} - \frac{1}{2}\right)$. By applying this result to the lower bound in (52) and additionally applying the $\phi_{n,k} \geq \frac{1}{2}$ assumption to the upper bound in (52), we obtain the following:

$$-\phi_{n,k} + \left|\frac{\Re[\tilde{R}_{n,k}]}{\tilde{R}_{n,n}}\right| < |\Re[u']| < \left|\frac{\Re[\tilde{R}_{n,k}]}{\tilde{R}_{n,n}}\right| + \phi_{n,k}. \tag{53}$$

This inequality shows that if we use $|\Re[u']|$ for the magnitude of the real part of u in Line 5, then the real part of the updated $\tilde{R}_{n,k}$ entry will satisfy the $\phi_{n,k}$ relaxed size reduction condition. Identical arguments can be used to prove this lemma for the imaginary part of $\tilde{R}_{n,k}$. ■

This lemma is not immediately useful because the integer-rounded divider does not produce the actual quotient. Instead, we consider an additional lemma.

Lemma 4 *Given the fixed-point Newton-Raphson in (42) that has maximum relative error $M_{\gamma,a}$,* (51) *is satisfied if the following is true:*

$$|\Re[u']|, |\Im[u']| < \frac{1 - M_{\gamma,a}}{M_{\gamma,a}}\left(\phi_{n,k} - \frac{1}{2}\right) - \frac{1}{2}. \tag{54}$$

Proof: Application of Lemma 2 and $\epsilon_q < \left|\frac{\Re[\tilde{R}_{n,k}]}{\tilde{R}_{n,n}}\right| M_{\gamma,a}$ together imply that $-\frac{1}{2} + \left|\frac{\Re[\tilde{R}_{n,k}]}{\tilde{R}_{n,n}}\right|(1 - M_{\gamma,a}) < |\Re[u']|$. This inequality when reconciled with (54) implies that (51) is true. We can use identical arguments to prove this lemma for $|\Im[u']|$. ■

According to this lemma, the extra rounding error detection and correction proposed in [16] is not necessary when (54) is satisfied. When (54) is not satisfied, this extra logic remains unnecessary if we allow repeated CLLL iterations. In other words,

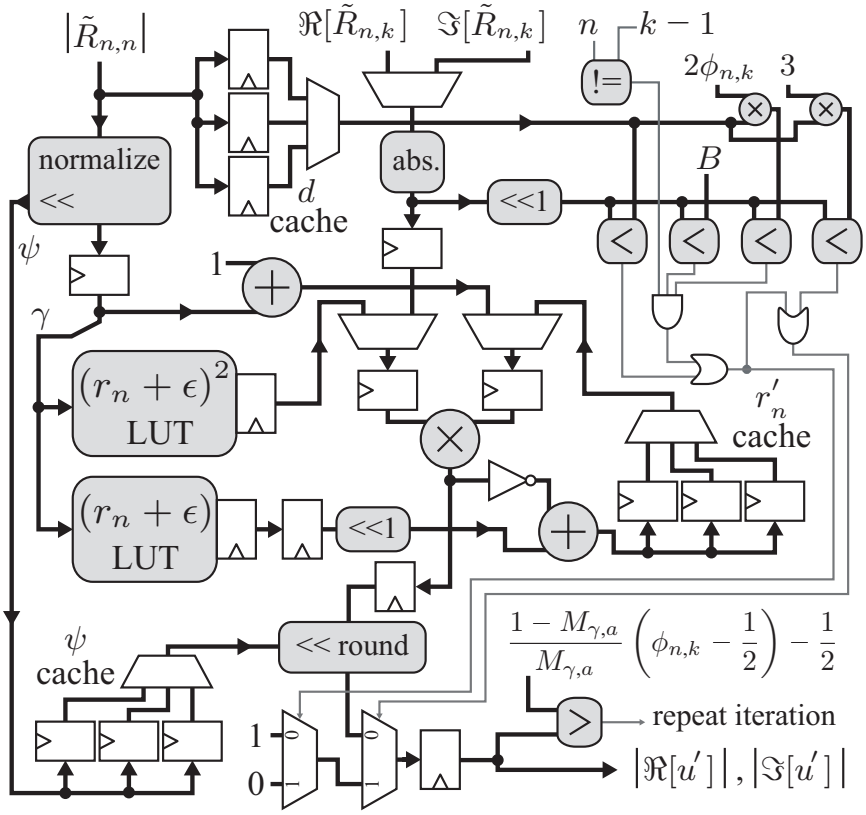

Figure 5: Proposed single Newton-Raphson iteration-based integer-rounded divider. A single multiplier is shared between the reciprocation and dividend datapaths. A collection of comparators and straightforward logic are used to evaluate the relaxed size reduction conditions and detect trivial integer quotients.

we simply repeat a CLLL iteration until all u' values generated during an iteration satisfy (54) and all off-diagonal elements in the k-th column of $\tilde{\boldsymbol{R}}$ satisfy the $\frac{1}{2}B$ absolute upper bound after the size reduction operation for that iteration. Since computed u' component magnitudes are always less than or equal to the actual u component magnitudes, the analysis in Chapter 3 remains valid.

4.1.3 Integer-Rounded Divider Architecture

Application of the ideas described in Section 4.1.1 for a CLLL system that processes 4×4 $\tilde{\boldsymbol{R}}$ matrices results in the straightforward architecture illustrated in Figure 5. The d input ($|\tilde{R}_{n,n}|$) input is in signed-magnitude form, while the n inputs ($\Re[\tilde{R}_{n,k}]$

and $\Im[\tilde{R}_{n,k}]$) are both in 2's complement form. The integer quotient outputs are also in 2's complement form. The two LUTs are implemented with a single 32×18-bit ROM. The reciprocal computation begins at the *normalize* module and requires 4 cycles, while a reciprocal-reuse division only requires 4 cycles. Both the shift amount ψ and the computed r'_n are stored in a three entry buffer. We accomplish the introduction of the -2^f term found in (42) by computing the one's complement of the truncated $\left(d_n + 2^{-(f+1)}\right)(r_n + e)^2$ product. Another 2 cycles are then required to multiply the reciprocal by the dividend, shift the truncated product by the stored ψ value, integer-round the result, and finally restore the sign. The architecture also contains straightforward logic for detecting trivial u values and evaluating the relaxed size reduction conditions.

4.2 Handling the Siegel Condition

Evaluation of the Siegel condition (Line 9) is straightforward, while computation of $\mathbf{\Theta}$ (Line 10) requires the inverse square-root operation, which has high hardware complexity. A numerically stable and efficient method, however, becomes apparent by forming the vector

$$\boldsymbol{v} = \left[|\tilde{R}_{k,k}| \;\; \Re[\tilde{R}_{k-1,k}] \;\; \Im[\tilde{R}_{k-1,k}]\right]^T \tag{55}$$

and viewing these computations as a vector normalization problem [20]. Then $\mathbf{\Theta}$ can be formed from the elements of $\boldsymbol{v}/\|\boldsymbol{v}\|$, and the updated $(k-1)$-th diagonal after the $\mathbf{\Theta}$ multiplication in Line 11 and column swap in Line 13 is $\|\boldsymbol{v}\|$.

4.2.1 Traditional Householder CORDIC Architecture Solution

We can solve this vector normalization problem by applying the Householder CORDIC algorithm [26]. Similar to classic CORDIC algorithms, vectoring (rotating a vector to an axis) and rotation (rotating a vector around an arbitrary axis) operations are computed using iterative application of low hardware complexity shifts and additions.

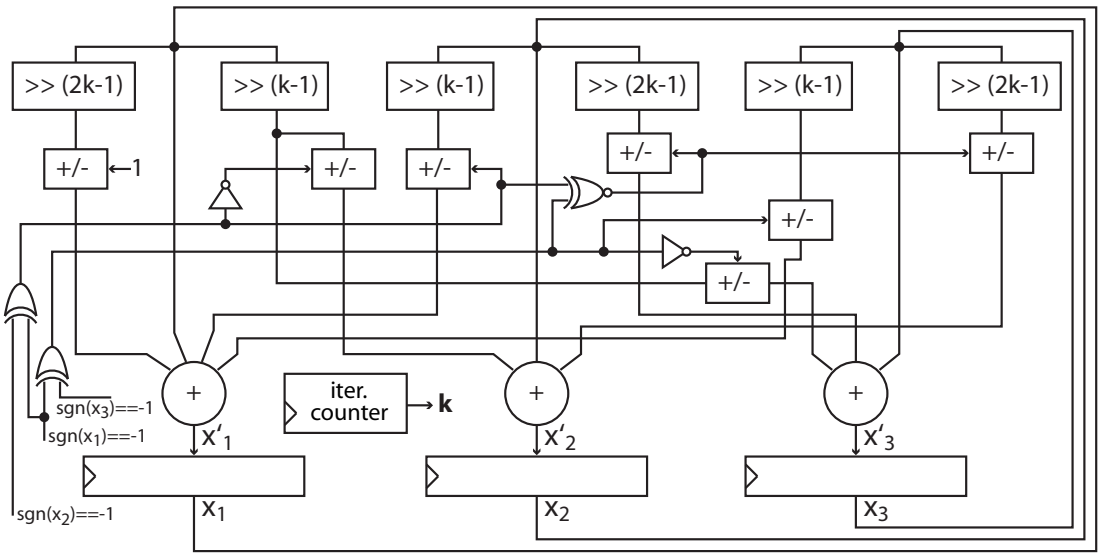

Figure 6: Single-iteration-per-cycle Householder CORDIC architecture.

A sequence of J Householder vectoring iterations can be used to compute $\|\boldsymbol{v}\|$ to a certain precision within a constant CORDIC gain factor, $C = \prod_{i=1}^{J} (1 + 2^{-2i+1})$ [26]:

$$C\left(\|\boldsymbol{v}\| + \epsilon\right) \boldsymbol{e}_1 = \boldsymbol{A}^{(J)} \cdots \boldsymbol{A}^{(1)} \boldsymbol{v}, \tag{56}$$

where $\boldsymbol{A}^{(i)}$ is determined from the sign of the vector elements at the end of each vectoring iteration and $(\boldsymbol{A}^{(i)})^T \boldsymbol{A}^{(i)} = \left(1 + 2^{-2i+1}\right)^2 \boldsymbol{I}$, $\boldsymbol{e}_1 = [1\ 0\ 0]^T$, and ϵ is an error term introduced by the finite number of vectoring iterations (for simplicity we neglect the error terms in the second and third elements). Conveniently, multiplication by $\boldsymbol{A}^{(i)}$ can be implemented with bit-shifts of length i and $2i$ (easily realized on an FPGA using J:1 multiplexers) and addition operations. We can then compute $\boldsymbol{v}/\|\boldsymbol{v}\|$ by rotating the vector $(1/C)\boldsymbol{e}_1$ using the transpose of the $\boldsymbol{A}^{(i)}$ matrices in the opposite order:

$$(\boldsymbol{A}^{(1)})^T \cdots (\boldsymbol{A}^{(J)})^T \left(\frac{1}{C}\boldsymbol{e}_1\right) = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\| + \epsilon}. \tag{57}$$

We can use the single-iteration-per-cycle Householder CORDIC architecture in Figure 6 to implement both (56) and (57). Here, all buses are in 2's complement format and the evolving vector elements are stored in the x_1, x_2, and x_3 registers.

Additionally, the negation blocks ("+/-") output the 2's complement of the input when the control input is "1" and the input when the control input is "0." The shifting and negation blocks essentially implement the multiplication of the vector elements by the elements of the $\boldsymbol{A}^{(i)}$ or $\left(\boldsymbol{A}^{(i)}\right)^T$ matrices. The three adders (toward the bottom of the figure) implement the summation part of this matrix multiplication. During vectoring iterations, the signs of the vector elements control the negation blocks. During rotation iterations, the signs of the vector elements stored from a previous sequence of vectoring iterations control the negation blocks.

4.2.2 Modified Householder CORDIC Architecture Solution

As a result of the reversed order that the $(\boldsymbol{A}^{(i)})^T$ matrices are applied in (57), the normalized $\boldsymbol{v}$ vector computation must begin after these matrices are determined. Therefore for a single-iteration-per-cycle Householder CORDIC architecture, $2J$ cycles are required for the $\boldsymbol{\Theta}$ matrix computation. It is clearly desirable to overlap the computation of $\|\boldsymbol{v}\|$ and $\boldsymbol{v}/\|\boldsymbol{v}\|$. Towards this goal, we consider a slight manipulation of (56):

$$\frac{v_i}{\|\boldsymbol{v}\| + \epsilon} = \boldsymbol{e}_1^T \boldsymbol{A}^{(J)} \cdots \boldsymbol{A}^{(1)} \left(\frac{1}{C}\boldsymbol{e}_i\right), \tag{58}$$

where v_i is the i-th element of $\boldsymbol{v}$ and $\boldsymbol{e}_i$ is the i-th standard Euclidean basis vector. This formulation shows that we can compute the i-th element of $\boldsymbol{v}/\|\boldsymbol{v}\|$ by rotating $\boldsymbol{e}_i/C$ using the $\boldsymbol{A}^{(i)}$ in the *same order* as applied in the vectoring iterations.

We can implement (56) and (58) by employing a single-iteration-per-cycle Householder CORDIC architecture that has been unrolled and pipelined by a factor of four such that a vectoring iteration and three rotation iterations can be executed concurrently. A schematic of this unrolling scheme, which we originally propose in [20], is shown in Figure 7. Each pipeline stage can either operate in vectoring mode (compute an $\boldsymbol{A}^{(i)}$ and then apply the $\boldsymbol{A}^{(i)}$ on the input vector) or rotation mode (apply a previously computed $\boldsymbol{A}^{(i)}$ on the input vector). The pipeline is initially filled by

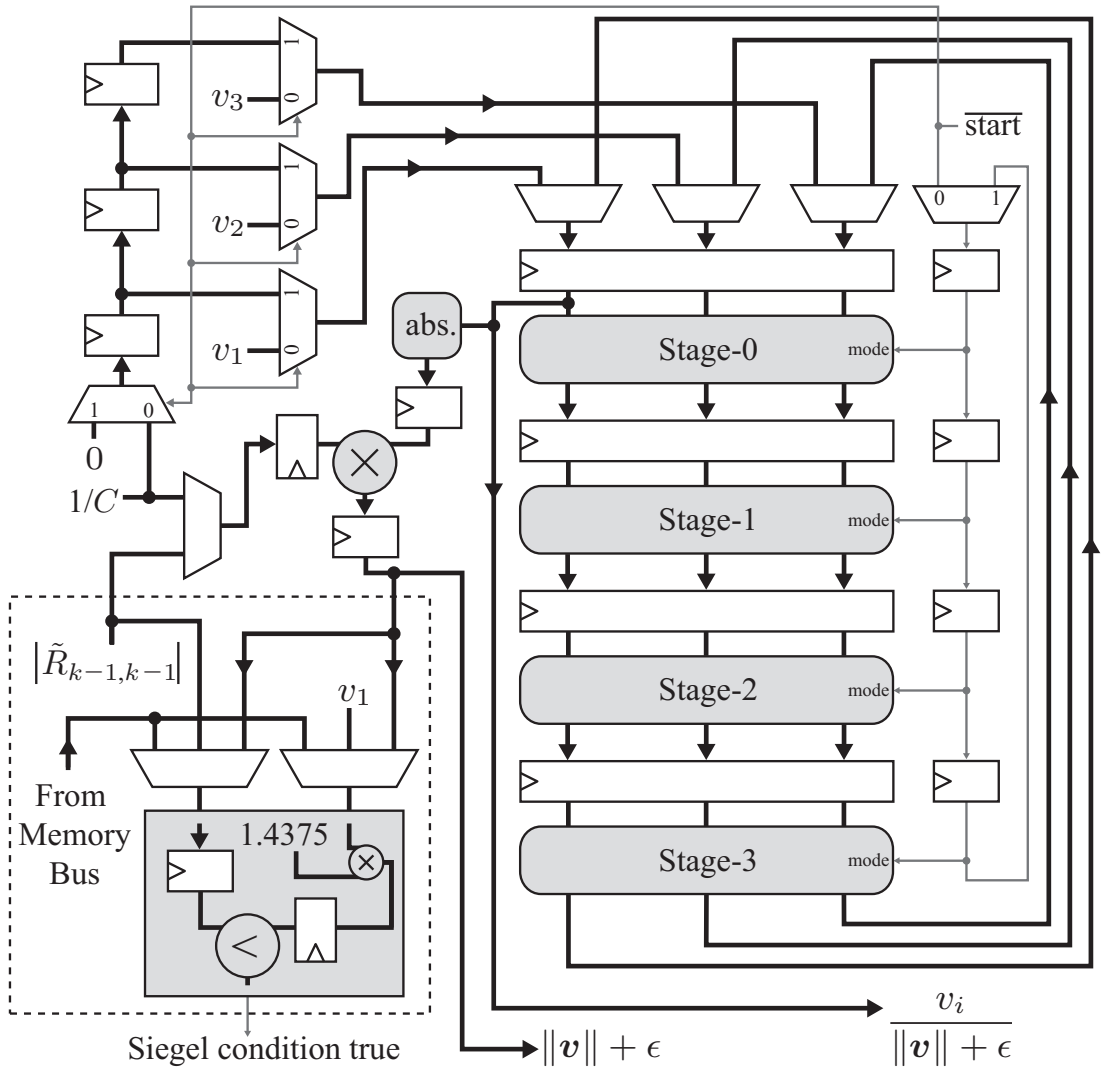

Figure 7: Proposed Householder CORDIC architecture consisting of single-cycle-per-iteration CORDIC module that has been unrolled and pipelined by a factor of 4. Each stage can operate in either vectoring mode (determining and applying an $\boldsymbol{A}^{(i)}$) or rotating mode (applying a previously determined $\boldsymbol{A}^{(i)}$). A secondary datapath (inside the dashed box) is used to evaluate the Siegel condition for $\zeta = 2.06640625$.

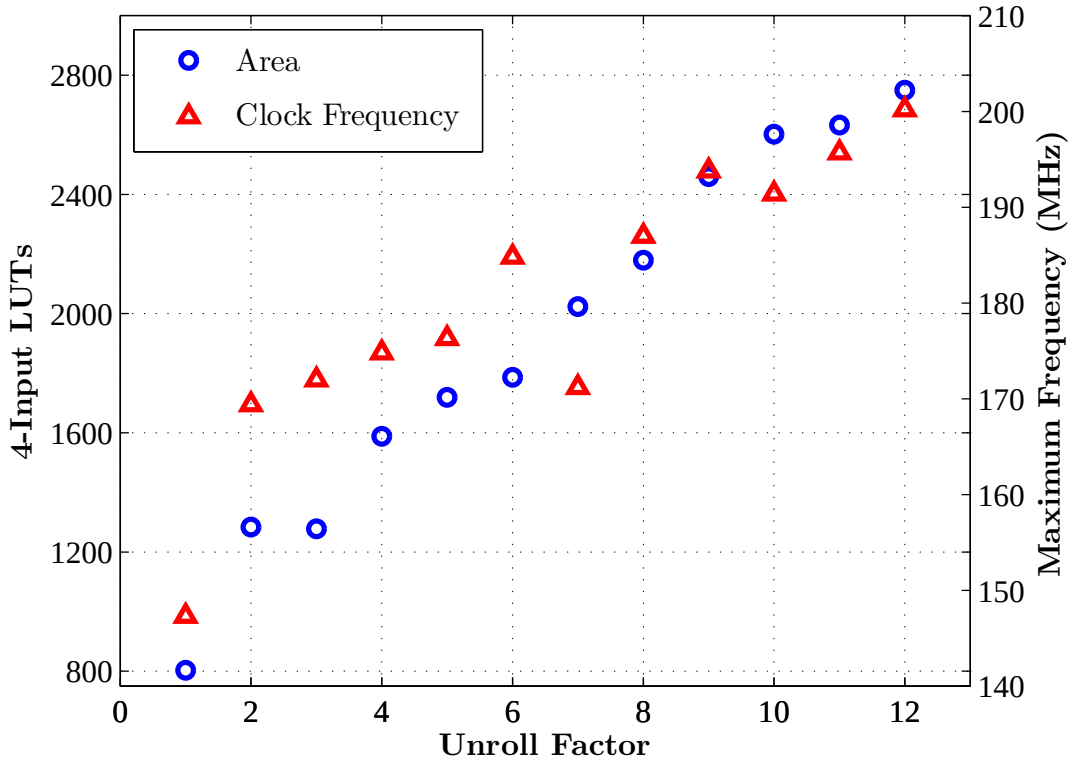

Figure 8: Effect of simultaneously unrolling and pipelining a 12-iteration single-iteration-per-cycle Householder CORDIC architecture on FPGA hardware resources and maximum clock frequency. All results are generated using an XC4VLX80-12 target. The original single-iteration-per-cycle Householder CORDIC architecture has a {8.13} fixed-point datapath.

inputting $\boldsymbol{v}$ into stage-0 in vectoring mode during the first cycle of initialization and inputting $\boldsymbol{e}_1/C$, $\boldsymbol{e}_2/C$, and $\boldsymbol{e}_3/C$ in rotation mode during the next three cycles, respectively. The results of these vectoring and rotations proceed through the pipeline, feeding back to stage-0 when the end of the pipeline is reached. After J cycles the computed $C\left(\|\boldsymbol{v}\| + \epsilon\right)$ exits the pipeline, and the computed elements of $\boldsymbol{v}/\left(\|\boldsymbol{v}\| + \epsilon\right)$ exit the pipeline in the following three cycles. Hence, if we adopt this architecture, then $J + 3$ cycles are required to compute the $\boldsymbol{\Theta}$ matrix.

This improvement over the original approach in (57) comes at the cost of four times as many adders and shifters. The complexity, however, of the individual shifters is

considerably decreased. Since the unrolling allows part of the shifting operations to be performed with wire shifts, each stage only requires $\lceil J/4 \rceil$:1 multiplexers. Given that large multiplexers map poorly to FPGAs (resource intensive and long critical path through LUTs), this is a desirable result. In addition, the unrolling allows more effective register re-timing because automated synthesis tools can move registers across the stages to improve the critical path.

As we demonstrate in Figure 8 for a 12-iteration single-iteration-per-cycle Householder CORDIC with an {8.13} fixed-point datapath, unrolling by a factor of four approximately doubles the FPGA hardware resource requirements. The clock rate, however, is increased by approximately 20%. More generally, the figure demonstrates the general trend of increasing clock rate with increasing unrolling factor.

4.2.3 Modifications that Prevent Unnecessary Computation

Since the $\boldsymbol{\Theta}$ matrix is only needed when the Siegel condition is false and basis updates are infrequent [1], we desire not to speculatively compute $\boldsymbol{\Theta}$. We therefore require a low-complexity method for evaluating the Siegel condition. Towards this goal, we notice that evaluating

$$\left|\tilde{\boldsymbol{R}}_{k-1,k-1}\right| \leq \left(1 + 2^{-1} - 2^{-4}\right) \left|\tilde{\boldsymbol{R}}_{k,k}\right| \tag{59}$$

is equivalent to evaluating the Siegel condition for $\zeta = 2.06640625$. Clearly, we can implement this inequality evaluation in hardware using only a comparator and two adders. Later in Section 5.3 we demonstrate that running the CLLL algorithm with this choice of ζ introduces a negligible degradation in BER performance.

The ability to quickly evaluate Siegel conditions allows us to incorporate easily the re-evaluation of Siegel conditions after basis updates into the Householder CORDIC architecture. We note that after a basis update for $k = k'$, the state of the Siegel condition becomes uncertain for $\max(2, k' - 1) \leq k \leq \min(k' + 1, N_t)$. We can utilize

this observation in a collection of two-state state machines, one for each Siegel condition, that track the condition of each Siegel condition. Each state machine indicates either "satisfied" or "uncertain" for each Siegel condition. When all state machines indicate "satisfied," then the CLLL algorithm can be terminated immediately because the CLLL-MMSE-SIC symbol vector estimate is unaffected by size reduction operations (as shown in Appendix A).

Figure 7 contains a secondary datapath for evaluating the Siegel condition using the proposed method. The secondary datapath also interfaces to an external data bus such that Siegel conditions can be evaluated as the $\tilde{\boldsymbol{R}}$ matrix memory is being filled. Lastly, the secondary datapath operates independently from the multiplier in the Householder CORDIC architecture, which is used for both Householder CORDIC gain compensation (multiplication of $C\left(\|\boldsymbol{v}\| + \epsilon\right)$ by $1/C$) and partial computation of basis updates (Section 5.2).

CHAPTER V

HARDWARE REALIZATION OF MODIFIED CLLL

In the previous chapters, we focus on analyzing both modifications of the CLLL algorithm and sub-modules for computing the more hardware-complex operations of the CLLL algorithm. In this chapter we apply this developed theory toward designing a hardware architecture of the modified CLLL algorithm from Chapter 3. After realizing the proposed architecture using a hardware description language and hardware synthesis tools, we are able to evaluate the proposed architecture along the metrics of hardware resources, throughput, and latency in a practical wireless communication system.

5.1 Establishing Design Parameters

Specifically, we can now apply the analysis in the previous sections, fixed-point simulation results, and the design parameters established in [42] to the design of a CLLL hardware architecture for 4×4 CLLL-MMSE-SIC detection. We assume that $\zeta = 2.06640625$ and that the communication channel can be approximated by the system model in (1). For the fixed-point simulations, we utilize hardware models of the Householder CORDIC and integer-rounded divider architectures for the CLLL algorithm. We also utilize the reciprocals computed during the execution of the CLLL algorithm for the SIC detection step. Lastly, we note that the design procedure in this section can also be used for other channel model assumptions once a suitable B is determined for a particular channel model.

5.1.1 Analytical

We first consider how to choose the relaxed size reduction parameters. By choosing $\phi_{n,k} = \frac{3}{2}$ for all $\tilde{\boldsymbol{R}}$ elements except those on the first off-diagonal, we can share one of the comparators required in the trivial u value detection (Figure 5). CLLL simulations consisting of 1.2 million channel realizations reveal that on average size reduction on these elements is then required only approximately 6.6% of all $1 \leq n \leq k-2$ inner-loop iterations (Line 4-7). Additionally, the maximum encountered $\Re[u], \Im[u]$ magnitude during these simulations is 10. A choice of $\phi_{k-1,k} = 0.51$ under the assumption that $G = 15$ guarantees that B is upper bounded by 7.31 (Application of (31)). Assuming that the maximum $\Re[u], \Im[u]$ magnitude that can be handled during a single iteration is 10 and that $\phi_{k-1,k} = 0.51$, we see that the integer-rounded divider should be designed such that $M_{\gamma,a} < 0.001$ (Application of Lemma 3). By representing u with 11 bits and allowing a CLLL algorithm iteration to be repeated when any u component magnitude exceeds 10, we can also safely handle size reduction operations for $|\Re[u]|, |\Im[u]| > 10$ efficiently. Simulations of the CLLL algorithm with this choice of relaxed size reduction conditions reveal that only 9.6% of all inner-loop iterations produce a u that has non-zero real and imaginary parts.

Given the $\phi_{n,k}$'s and computed B above, we are now able to determine the following integer bit parameters:

- Since the diagonal elements of $\tilde{\boldsymbol{R}}$ are upper bounded by B, the NR-based reciprocation datapath only requires 3 (unsigned) integer bits.

- The dividends in Line 4 are the components of the $\tilde{R}'_{n,k}$ elements defined in Section 3.2.5. Evaluation of the upper bound in (40) reveals that given our chosen relaxed size reduction conditions, all the dividend magnitudes are bounded above by $2^{6.60}$ (8 dividend integer bits).

- After the size reduction operation of a particular k outer-loop iteration, the

magnitude of each off-diagonal element in the k-th column is upper bounded by $\frac{B}{\sqrt{2}}$, but subsequent basis updates could increase the magnitude of these matrix elements. In the worst case the maximum energy of $2^{2(3.37)}$ (Evaluating (39) for $k = 4$) could be redistributed to the real or imaginary component of a single matrix element. Therefore, at the beginning of a size reduction process, at most 5 integer bits are required to represent each real and imaginary component of $\tilde{\boldsymbol{R}}$.

- The output of the Householder CORDIC datapath is only utilized when the Siegel condition is not true. In this case, the magnitude of $\boldsymbol{v}$ in (55) is upper bounded by B. It follows then that the righthand side in (56) is upper bounded by CB. Therefore, the internal Householder CORDIC datapath requires 5 integer bits.

5.1.2 Empirical

The *orthogonality deficiency threshold* ϵ_{th} design parameter in [42] affects how often CLLL preprocessing must be completed on the channel matrices to maintain BER performance. Linear detection is utilized when the *orthogonality deficiency* (od[1]) of $\tilde{\boldsymbol{R}}$ is below ϵ_{th} and sphere decoders are utilized otherwise. The ϵ_{th} parameter can be used to trade off how often sphere decoding is executed with the coding gain gap to ML detection. Since CLLL-aided detection is a full-diversity detection scheme, we can apply this idea to the CLLL-MMSE-SIC detection of interest in this paper by employing a hybrid MMSE-SIC/CLLL-MMSE-SIC detector. If we set $\epsilon_{th} = 0.955$, then on average only 40% of all channel matrices need be processed with the CLLL algorithm to achieve a 0.2 dB gap to ideal CLLL-MMSE-SIC detection.

When choosing design parameters that affect the computation precision of the

[1]The orthogonality deficiency od satisfies $0 \leq od \leq 1$. An orthogonal matrix has $od = 0$, while a singular matrix has $od = 1$ [42].

hardware implementation, we first notice that no loss of precision can occur during the size reduction process because there is no expansion in the number of fraction bits. An expansion of fraction bits only occurs when a basis update occurs, which involves computation of $\boldsymbol{\Theta}$ and application of this matrix on $\tilde{\boldsymbol{R}}$ and $\tilde{\boldsymbol{Q}}$. Assuming $\epsilon_{th} = 0.955$, we utilize additional fixed-point simulations of the hybrid detector to verify that BER performance is maintained with a choice of 13 fraction bits to represent the $\tilde{\boldsymbol{R}}$ and $\tilde{\boldsymbol{Q}}$ matrices, 9 Householder CORDIC iterations, and a maximum of 15 CLLL iterations We can then determine the integer-rounded divider LUT requirements based on the $\tilde{\boldsymbol{R}}$ fraction bit choice. Using the procedure in [16] and the $M_{\gamma,a} < 0.001$ condition determined in Section 5.1.1, we find that $\gamma = 5$ and $a = 6$.

Lastly, we use 9 integer bits to represent the elements of $\boldsymbol{T}$ per real and imaginary part, which was verified in [1] with a real-time prototyping system.

5.2 Proposed Architecture

Developing a suitable top-level architecture for the CLLL algorithm is complicated by the fact that the dataflow of the CLLL algorithm is dynamic—each random channel matrix results in a different sequence of memory accesses and operations. We first notice, however, that operations on $\boldsymbol{T}$ and $\boldsymbol{g}$ only depend on the generated u values from size reduction operations and operations on $\tilde{\boldsymbol{Q}}$ only depend on the $\boldsymbol{\Theta}$'s generated from basis updates. This suggests an architecture where an $\tilde{\boldsymbol{R}}$ Processor generates u's and $\boldsymbol{\Theta}$'s, and a $\boldsymbol{T}$ Processor and a $\tilde{\boldsymbol{Q}}$ Processor are slaves, consuming these operands. We can adopt First-in-First-Out (FIFO) structures to store these operands and have each processor track the CLLL algorithm state separately. Then the slave processors need not consume these operands at the same rate as the $\tilde{\boldsymbol{R}}$ Processor.

We could adopt separate multiplier pipelines for each processor, but given our observations in Section 5.1.1 we consider a different approach. Since the generated u values are sparse when the relaxed size reduction condition is adopted and only a

fraction of all CLLL iterations require basis updates [1], it is advantageous toward high multiplier utilization to choose a shared multiplier/accumulator structure with arbitration. Furthermore, we can use a multiplier pipeline that implements complex multiplication via separate real and imaginary component multiplication to exploit the low frequency of fully complex u values (as shown in Section 5.1).

A detailed block diagram of the proposed architecture is shown in Figure 9. The "$\tilde{\boldsymbol{R}}$ Processor" effectively consists of the shared Multiplier Pipeline, Column Accumulator, and all remaining modules except the $\boldsymbol{T}$ and $\tilde{\boldsymbol{Q}}$ processors. In the remaining sub-sections we consider the individual operation of each processor and how we can use a straightforward double-buffering scheme to overlap the CLLL processing of different channel matrices.

5.2.1 $\tilde{\boldsymbol{R}}$ Processor

The CLLL algorithm operates heavily on a single column during the size reduction process each iteration. We therefore base the $\tilde{\boldsymbol{R}}$ Processor around a partial column buffer that stores the $\tilde{\boldsymbol{R}}_{1:k-1,k}$ intermediate size reduction results. This choice also naturally follows from the analysis results in Section 5.1.1, which demonstrate that the $\tilde{\boldsymbol{R}}$ elements magnitude upper bound during the size reduction process is greater than the $\tilde{\boldsymbol{R}}$ elements magnitude upper bound at both the start and end of the size reduction process. Therefore, in the partial column buffer architecture, the $\tilde{\boldsymbol{R}}$ memory need only be sufficiently wide to represent $\tilde{\boldsymbol{R}}$ at the start of the size reduction process.

The $\tilde{\boldsymbol{R}}$ Processor is additionally based around a single-port, single complex entry memory for storing $\tilde{\boldsymbol{R}}$. Address mapping is employed for column swapping. The datapath modules include the shared Multiplier Pipeline, Int-Rounded Divider that is designed according to the results in Section 5.2, and the proposed Householder CORDIC architecture that has been partitioned into CORDIC Pipeline Support (secondary datapath) and CORDIC Pipeline. Parallel operation among

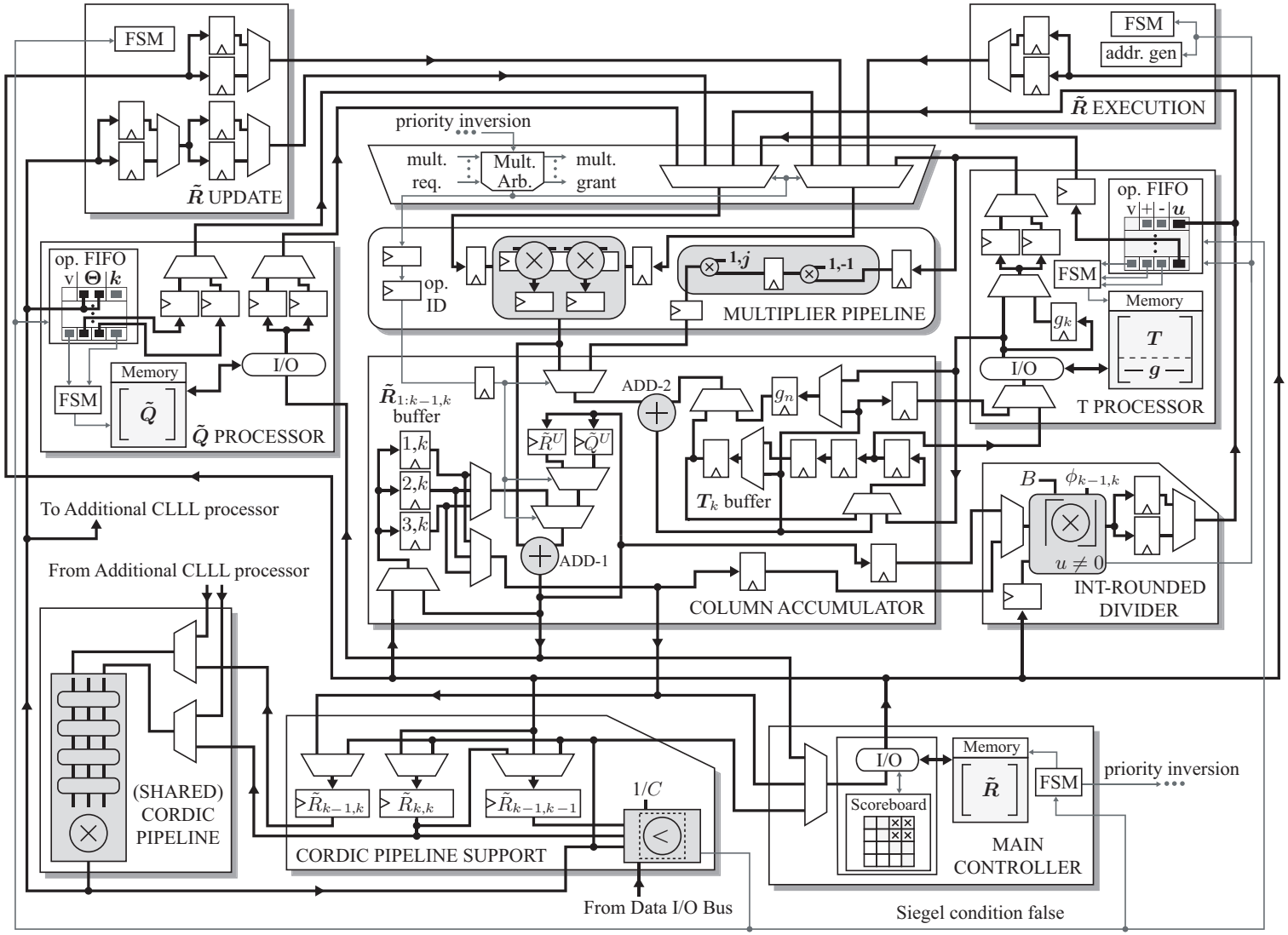

Figure 9: Proposed architecture for the CLLL processor based on a shared multiplier pipeline and column accumulators for operations on $\tilde{\boldsymbol{R}}$ and $\boldsymbol{T}$.

these modules is enabled through a combination of forwarding paths, speculative execution, and slight reordering of the original CLLL algorithm. At the beginning of each CLLL iteration, MAIN CONTROLLER directs the following:

- The $\tilde{\boldsymbol{R}}_{1:k-1,k}$ buffer in COLUMN ACCUMULATOR is loaded either from the $\tilde{\boldsymbol{R}}$ memory, or the current contents of the buffer from the previous CLLL iteration are reused (If during the previous iteration $k \neq 2$ and a basis update was required). The $\tilde{R}_{k-1,k}$ element is also speculatively sent to CORDIC PIPELINE SUPPORT.

- The $\tilde{R}_{k-1,k-1}$ element is sent to CORDIC PIPELINE SUPPORT from either the $\tilde{\boldsymbol{R}}$ memory or a forwarding path. INT-ROUNDED DIVIDER either receives this element and begins reciprocation, or it reuses a stored reciprocal.

- The $\tilde{R}_{k,k}$ element is sent to CORDIC PIPELINE SUPPORT from either the $\tilde{\boldsymbol{R}}$ memory or a forwarding path. This module then begins evaluating the Siegel condition according to (59).

5.2.1.1 Size Reduction

The $\tilde{R}_{k-1,k}$ element is then forwarded from the $\tilde{\boldsymbol{R}}_{1:k-1,k}$ buffer to the dividend input of INT-ROUNDED DIVIDER, and the size reduction condition evaluation is initiated in this module. The real and imaginary components each require 1 cycle for this evaluation operation. The results of the evaluation are written into a small table that $\tilde{\boldsymbol{R}}$ EXECUTION accesses. When this table indicates that a nonzero u has been generated, $\tilde{\boldsymbol{R}}$ EXECUTION begins fetching the required $\tilde{\boldsymbol{R}}$ column from the $\tilde{\boldsymbol{R}}$ memory and simultaneously issues single u component multiplications to MULTIPLIER PIPELINE, starting with the $(k-1)$-th row. The multiplier results are then sequentially added via ADD-1 to their corresponding elements in the $\tilde{\boldsymbol{R}}_{1:k-1,k}$ buffer as they exit MULTIPLIER PIPELINE, and the buffer is updated with these new values.

As the updated $\tilde{R}_{k-1,k}$ element is written to the buffer, it is simultaneously forwarded to Int-Rounded Divider. Since Int-Rounded Divider already contains the required diagonal element and reciprocal (stored in caches), size reduction on the next $\tilde{\boldsymbol{R}}$ element can then begin as the remaining elements complete. This process continues until size reduction on the k-th column is complete. The gradual write-back of the $\tilde{\boldsymbol{R}}_{1:k-1,k}$ buffer elements to the $\tilde{\boldsymbol{R}}$ memory is overlapped with this operation.

5.2.1.2 Siegel Condition and Basis Updates

Concurrent with this size reduction process is operation of CORDIC Pipeline Support and CORDIC Pipeline. If CORDIC Pipeline Support indicates that the Siegel condition is true, then Main Controller is signaled that k can be incremented (and the next iteration initiated once the size reduction process is complete or the entire CLLL processing terminated if all Siegel conditions are now satisfied). If the Siegel condition is false, then CORDIC Pipeline waits until either the size-reduced $\tilde{R}_{k-1,k}$ element is forwarded or Int-Rounded Divider indicates that size reduction on this element is not required. The $\boldsymbol{\Theta}$ (Line 10) calculation can then begin because the necessary operands have already been speculatively loaded at the start of the iteration. Once the specified number of CORDIC iterations have been completed, the uncompensated $C\|\boldsymbol{v}\|$ result (56) streams out of CORDIC Pipeline to CORDIC Pipeline Support for gain compensation. This is followed by three cycles of the $\boldsymbol{\Theta}$ elements streaming out to CORDIC Pipeline Support, $\tilde{\boldsymbol{R}}$ Update, and the $\tilde{\boldsymbol{Q}}$ Processor. As these elements input into CORDIC Pipeline Support, the elements are appropriately signed and multiplied by the buffered $\tilde{R}_{k-1,k-1}$ element to form the updated elements of the $(k-1)$-th column (due to a required basis update) in Line 11. These elements and the computed $\|\boldsymbol{v}\|$ are then sent to Main Controller for writeback. If there are remaining $\tilde{\boldsymbol{R}}$ elements that must be updated, then Main

Controller marks these elements as "pending" in the scoreboard structure, triggers $\tilde{\boldsymbol{R}}$ Update to compute these remaining updates, immediately decrements k, and effectively swaps the $\tilde{\boldsymbol{R}}$ columns by updating the address mapping register. Concurrent with this operation is the reevaluation of affected Siegel conditions (as described in Section 4.2.3).

As Main Controller initiates the next iteration, $\tilde{\boldsymbol{R}}$ Update gradually fetches required elements from the $\tilde{\boldsymbol{R}}$ memory and issues two multiplications to Multiplier Pipeline when access is granted by the multiplier arbitration module. Hence, each $\tilde{\boldsymbol{R}}$ element updated by a $\boldsymbol{\Theta}$ multiplication requires 3 accesses to this module. The $\tilde{R}^U$ register and complex ADD-1 adder in Column Accumulator accumulate the partial $\boldsymbol{\Theta}$ multiplication results from Multiplier Pipeline. Upon the final accumulation for a particular element, the ADD-1 output is immediately written back to the $\tilde{\boldsymbol{R}}$ memory, and the corresponding scoreboard entry is updated.

5.2.1.3 Memory Contention

Given that multiple modules access the $\tilde{\boldsymbol{R}}$ memory and basis updates on $\tilde{\boldsymbol{R}}$ are overlapped with subsequent CLLL iterations, a memory arbiter is included in Main Controller. If no data dependency is present, then the highest priority is assigned to memory reads associated with size reduction and lowest on memory read requests from $\tilde{\boldsymbol{R}}$ Update. If instead the scoreboard indicates that a currently requested element is "pending," then the $\tilde{\boldsymbol{R}}$ Processor stalls and $\tilde{\boldsymbol{R}}$ Update read requests are promoted to highest priority. The processor remains in this "priority inversion" state until the dependency is resolved.

5.2.2 $\boldsymbol{T}$ Processor and $\tilde{\boldsymbol{Q}}$ Processor

We base the $\boldsymbol{T}$ Processor on the hardware structures in the $\tilde{\boldsymbol{R}}$ Processor that handle the size reduction. The $\boldsymbol{T}$ Processor issues operations to Multiplier Pipeline to implement Line 6-7. It contains a single-port memory to store an augmented matrix

that consists of $\boldsymbol{T}$ concatenated with $\boldsymbol{g}$ (If MMSE processing is desired). The FIFO that the $\boldsymbol{T}$ Processor utilizes contains both non-zero u values and control flags that indicate the state of the $\tilde{\boldsymbol{R}}$ Processor when a u was generated. These include flags that indicate if the currently retrieved u was the last u generated for that CLLL iteration and if k was incremented or decremented during that iteration. These flags when combined with internal address mapping for column swapping, independent tracking of k, and the separate single-port memory, allow the $\boldsymbol{T}$ Processor to operate independent of the $\tilde{\boldsymbol{R}}$ Processor state. In addition, each FIFO entry contains flags that indicate if the currently retrieved u has real or imaginary components equal to ± 1. Hence, the $\boldsymbol{T}$ Processor can either issue single u component multiplications or utilize the trivial ± 1 multiplication path in MULTIPLIER PIPELINE.[2] Results of these integer operations are accumulated using ADD-2 and a straightforward shift register ($\boldsymbol{T}_k$ buffer) in COLUMN ACCUMULATOR.

The $\tilde{\boldsymbol{Q}}$ Processor is nearly identical to $\tilde{\boldsymbol{R}}$ UPDATE except that the $\boldsymbol{\Theta}$ parameters are retrieved from a FIFO. The FIFO also contains an entry for the value of k associated with the currently accessed $\boldsymbol{\Theta}$. Since the $\tilde{\boldsymbol{Q}}$ Processor also contains a separate, single-port memory for $\tilde{\boldsymbol{Q}}$, it can complete $\tilde{\boldsymbol{Q}}$ basis updates independently of the $\tilde{\boldsymbol{R}}$ Processor state. Partial $\boldsymbol{\Theta}$ multiplication results are accumulated in the $\tilde{Q}^U$ register located in COLUMN ACCUMULATOR.

To prevent the $\boldsymbol{T}$ Processor or $\tilde{\boldsymbol{Q}}$ Processor FIFO entries from being overwritten before being processed, we include "nearly full" status flags in these FIFOs. For the $\boldsymbol{T}$ Processor, this flag is first asserted when the FIFO only has a sufficient number of empty entries to store the maximum number of possible non-zero u values generated during a single CLLL iteration. For the $\tilde{\boldsymbol{Q}}$ Processor, this flag is first asserted when the FIFO only has a sufficient number of empty entries to store one additional $\boldsymbol{\Theta}$.

[2]We do not use this ± 1 multiplication path for operations issued by $\tilde{\boldsymbol{R}}$ EXECUTION because additional multiplexing is required.

When either of these flags is asserted, the MAIN CONTROLLER is signalled to stall the $\tilde{\boldsymbol{R}}$ Processor during the next CLLL iteration.

5.2.3 Arbitration and Overlapped Processing

It is apparent from the above architecture description that arbitration is required to handle contention for access to MULTIPLIER PIPELINE among the various modules. Clearly, in general an arbitration scheme should be adopted such that the $\tilde{\boldsymbol{R}}$ Processor can progress through CLLL iterations as quickly as possible. Therefore, when no data dependency exists the multiplier arbitration module assigns highest priority to $\tilde{\boldsymbol{R}}$ EXECUTION followed by the $\boldsymbol{T}$ Processor, $\tilde{\boldsymbol{R}}$ UPDATE, and the $\tilde{\boldsymbol{Q}}$ Processor. When a data dependency exists ("priority inversion"), requests from $\tilde{\boldsymbol{R}}$ UPDATE are promoted to highest priority, and requests from $\tilde{\boldsymbol{R}}$ EXECUTION are demoted to lowest priority.

As a final addition to the proposed architecture, we consider overlapped execution of the CLLL algorithm on multiple channel matrices among the three processors. This can be accomplished by employing two banks of memory in each processor. As a processor is operating on one memory bank, the other memory bank can be simultaneously filled with the next matrix/vector associated with the next channel matrix to process. Then once the processor completes operations on the current memory bank, it can simultaneously output the current memory bank contents and immediately begin processing on the other memory bank (Assuming the FIFOs are not empty in the $\boldsymbol{T}$ and $\tilde{\boldsymbol{Q}}$ Processor case).

5.3 Experimental Results

5.3.1 Performance of the Proposed Architecture

To evaluate the proposed architecture, we first implemented the CLLL processor in Verilog. Through functional simulations of the architecture, we determined that the $\tilde{\boldsymbol{Q}}$ Processor and $\boldsymbol{T}$ Processor FIFO depth should be 9 and 16, respectively, such that the $\tilde{\boldsymbol{R}}$ Processor stalls infrequently. We verified the design using the fixed-point

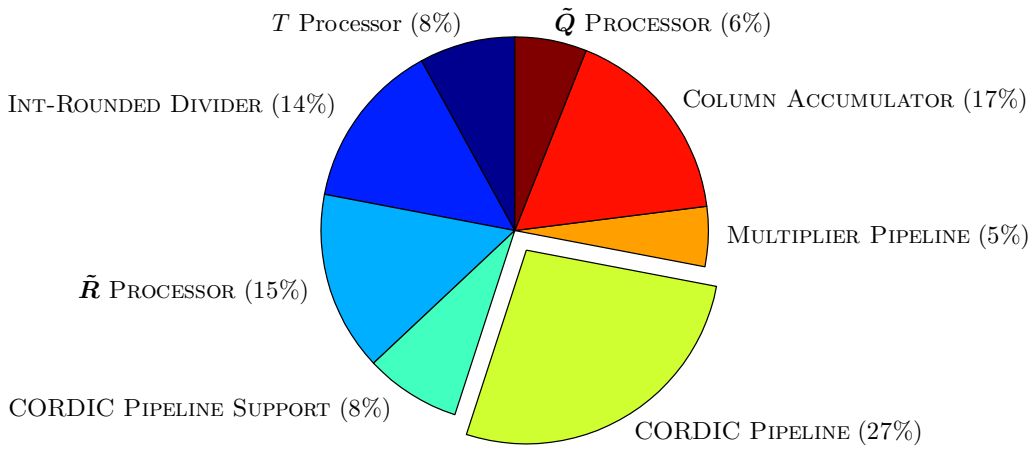

Figure 10: Distribution of the required XC4VLX80-12 FPGA slices for the proposed architecture. Although CORDIC Pipeline occupies the most slices compared to the other modules, it can be shared among additional CLLL processors.

model employed in Section 5.1.1 and a co-simulation environment that we developed (see Appendix E). Hardware realization was then completed by using an FPGA flow consisting of Synplify Pro for synthesis and Xilinx ISE 9.1 for place-and-route (PR). The proposed architecture hardware realization results for a variety of FPGA targets is summarized in Table 2. The approximate contribution of each module to the total required number of slices for the XC4VLX80-12 FPGA target is contained in Figure 10.

The average processing cycles per channel matrix under the assumption of the system model in (1) is included for both $\epsilon_{th} = 0$ (CLLL always applied) and $\epsilon_{th} = 0.955$ (system average). An estimate for the maximum cycles required by the proposed architecture to process a channel matrix is obtained by considering one million channel matrices with $od > 0.9999$, i.e., close to singular.

The 64-QAM hard detection BER simulation results of the proposed hybrid MMSE-SIC/CLLL-MMSE-SIC detector and previously implemented algorithms are contained

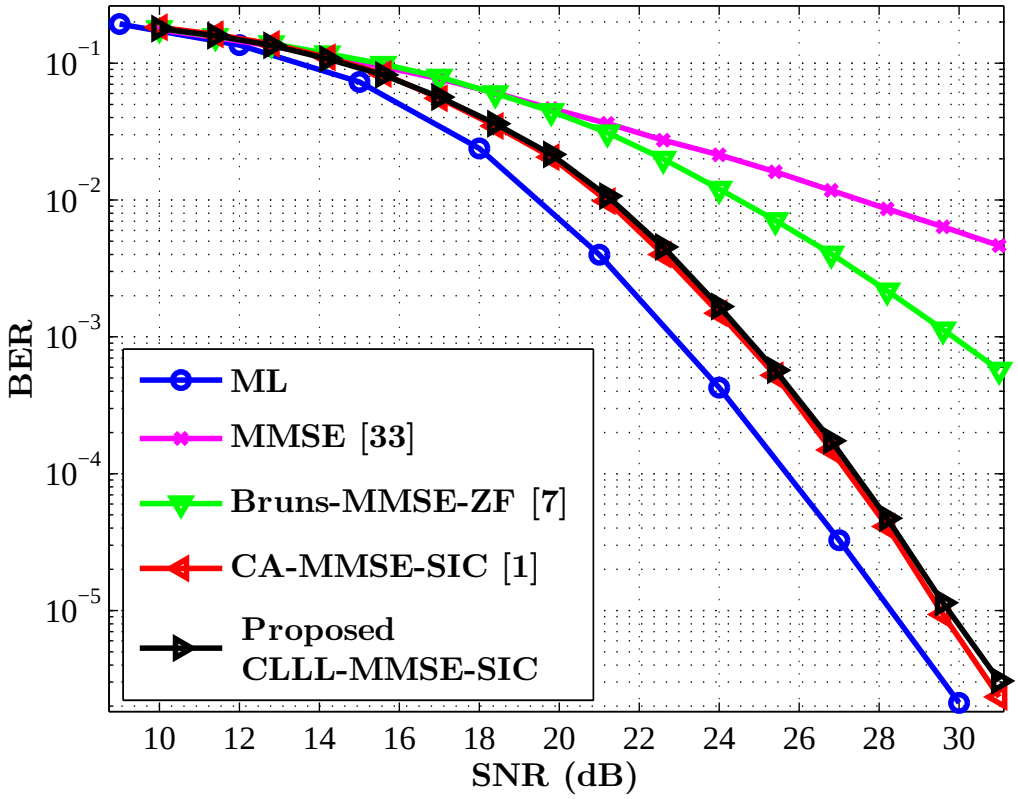

Figure 11: 64-QAM simulation results for proposed architecture and previously implemented algorithms. It is assumed that $\epsilon_{th} = 0.955$ and the maximum number of CLLL iterations is 15 for BER simulations of the proposed CLLL-MMSE-SIC. For each SNR a sufficient number of channel realizations are simulated to generate a minimum of 3000 bit errors.

in Figure 11. We obtain the proposed detection simulation results by employing a bit accurate model of the proposed architecture. The BER performance of previously implemented algorithms is obtained from ideal algorithm models (unlimited iterations and floating-point precision). From the figure we see that by completing CLLL processing on only 40% of all channel matrices on average, a considerable BER performance improvement is achieved over the MMSE detection in [33]. We also see that the proposed architecture achieves a 5 dB improvement in BER performance compared to the work in [7] and is within 1.5 dB of optimal ML detection.

We evaluated the proposed architecture from a system perspective by simulating

the packet structure of an 802.11n system in *Mixed Mode* [23]. The OFDM symbol length in this case is 4 µs, and there are 52 sub-carriers. We assumed that the sorted QR-decomposition of the channel matrix for each sub-carrier is completed just as the corresponding symbol vector associated with that sub-carrier in the first OFDM symbol is received. We measured the latency at the end of the first transmitted OFDM symbol, use the Virtex5 synthesis results, and set $\epsilon_{th} = 0.955$. Simulations of this system configuration indicate that the probability of the latency exceeding 12.08 µs (3.02 OFDM symbols) is 0.5%, and the average latency is 5.7 µs. Hence, a single CLLL processor with an OFDM symbol buffer is sufficient to handle medium to large size packets (10-100 OFDM symbols).

5.3.2 Comparison to Existing Work

Since BER performance is always relative to complexity (and therefore hardware resources), we compare the proposed architecture to previously proposed architectures that achieve full diversity and are within 1 dB - 2 dB of the proposed architecture BER performance.

5.3.2.1 Sphere Decoding

The work in [76] achieves 4×4 ML diversity but requires an 8×8 system with repetition coding. In the sphere decoding FPGA implementation domain we therefore only consider the implementations in [27] and [3], which achieve "native" ML BER performance. Direct comparison of these two implementations to the proposed architecture is not possible because we focus on efficient implementation of an enhancement for channel matrix preprocessing, not a symbol processing enhancement. Instead, we consider the relative hardware required for an example MIMO system using both approaches under the assumptions present in the prior work.

The 16-QAM SD algorithm implementation with Schnorr-Euchner (SE) enumeration for 4×4 systems in [27] achieves ML detection BER performance at an average

throughput of 5.1M symbol-vector/s. Hence to reach the throughput of 802.11n, at least three sphere decoding cores must be utilized (More may be needed to handle the large variance of this SD algorithm). Given the preprocessing assumptions in [27], channel coherence time on the order of milliseconds and transmission frames on the order of 100 milliseconds, a single CLLL processor is sufficient to support the preprocessing in the example system. From Table 2 we see that compared to the three sphere decoding cores the required CLLL processor is only 30% of the slices and 3.7% of the multipliers. We also note that this SD implementation requires both sorted QR-decomposition and $\boldsymbol{R}$ inversion preprocessing, while the CLLL processor only requires sorted QR-decomposition preprocessing.

The fixed sphere decoding (FSD) implementation in [3], which appears to be the only 64-QAM sphere decoding FPGA implementation published, achieves ML detection BER performance at a fixed throughput of 18.75M symbol-vector/s. By adopting four CLLL processors we can achieve a CLLL latency of 2.31 µs on average and 6.20 µs system-worst-case.[3] From Table 2 it is apparent that compared to the FSD implementation, the four CLLL processors require 56%[4] of the slices and 6.4% of the multipliers. We also note that the FSD implementation requires DMI, Cholesky decomposition, and the FSD ordering of the channel matrix as preprocessing steps, which are not part of the FSD implementation.

5.3.2.2 Lattice Reduction

We are not able to compare the proposed architecture in detail to the work in [62] and [69], which do not contain hardware resource requirements. Additionally, comparison to the Seysen's algorithm implementation in [72] is complicated by the fact that a 65

[3]We compute these latencies using the 802.11n packet simulation results adjusted for a Virtex2-Pro and assuming pessimistically that parallelization is only possible at the end of the first OFDM symbol.

[4]Here we have conservatively not included CORDIC Pipeline sharing, which is possible with the proposed architecture.

nm application specific integrated circuit (ASIC) is the target hardware platform. To conduct a conservative comparison, however, we consider the proposed architecture synthesis results for a Virtex5 target, which is fabricated in a 65 nm process. Using the posted clock frequency and worst-case processing cycles in Table 2, we see that the channel matrix processing latency is 2.17 µs for the proposed architecture and 3.42 µs for the Seysen's algorithm implementation. Only half the number of multipliers is required to achieve this nearly 37% reduction in worst-case latency, which could be further improved with adoption of a 65 nm ASIC design flow.

From Table 2 we see that the proposed architecture achieves considerable improvement in all hardware resource metrics and requires less than an eighth of the processing cycles compared to the Clarkson's algorithm implementation in [1]. An examination of the differences between these two architectures demonstrates that the algorithm modifications and design decisions taken in our work contribute significantly to this improvement:

- The architecture in [1] utilizes a shared division unit for both computing u values (via reciprocation multiplication) and computing $\boldsymbol{\Theta}$ matrices. We instead choose to utilize a reduced-precision reciprocation unit in addition to a collection of comparators for detecting trivial u values. In addition, the reciprocals are sufficiently accurate for use in the subsequent SIC detection step.

- We choose to relax the size reduction condition on the $\tilde{\boldsymbol{R}}$ elements as opposed to eliminating size reduction operations (as done in [37] and implemented in [1]). This allows us to upper bound the $\tilde{\boldsymbol{R}}$ elements during the CLLL processing. The slight increase in the number of size reduction operations is more than compensated by the efficient utilization of Multiplier Pipeline in the proposed architecture, which is over 81% for the system simulation described in Section 5.3.1.

- The 3-dimension Householder CORDIC algorithm employed in the proposed architecture requires only one sequence of vectoring iterations, while the 2-dimension Givens rotation-based CORDIC unit in [1] requires two sequences of vectoring iterations. The unrolling inherent in the proposed Householder CORDIC architecture, which is required to support the concurrent $\mathbf{\Theta}$ computation, results in CORDIC Pipeline requiring the largest percentage of hardware resources (as shown in Figure 10). We note, however, that the critical path of CORDIC Pipeline, which limited the achievable clock frequency in [19], is improved by over 20% and that this module can now be easily shared among CLLL processors.

- The proposed architecture does not compute the Siegel condition exactly as done in [1]. Instead, a low-complexity approximation is employed that results in a negligible degradation in BER performance. As a result, the proposed architecture is able to re-evaluate Siegel conditions rapidly without multiplication and use this information to terminate the CLLL algorithm earlier (as justified in Appendix A).

Table 2: VLSI implementation results

	Proposed Architecture (CLLL-MMSE-SIC)			**Lattice Reduction**		**Sphere Decoding**	
				[1]	**[72]**	**[3]**	**[27]**
Platform	XC4VLX80-12	XC5VLX110-3	XC2VP30-7	XC2VP30-7	0.65 nm ASIC	XC2VP70-7	XC2VP30-7
Multipliers	4	4	4	24	8	252	36
Hardware Usage	3,640 slices	1,758 slices	3,571 slices	7,349 slices	67,000 gates	24,815 slices	3,880 slices
Clock (MHz)	173	206	140	100 †	400	150	251
cycle/matrix	49 avg., 96 system avg., 447 worst-case			420 avg.	1368 worst-case	NA	NA
symbol-vector/s	NA	NA	NA	NA	NA	18.75M	5.1M
Modulation	*	*	*	*	*	64-QAM	16-QAM

* BPSK, 4/16/64-QAM † Clock frequency fixed by development board

CHAPTER VI

BEHAVIOR OF LR WHEN SPATIAL CORRELATION EXISTS

The inherent assumption of the channel matrix model in (1) is that no spatial correlation exists at the receiver and transmitter. This assumption is only relevant for a rich-scattering environment and sufficient antenna spacing. This channel matrix model, which is often referred to as the canonical channel matrix model, is often adopted because it enables tractable analysis of communication systems. Clearly, the channel matrix model can significantly affect the behavior of communication system simulations. For example, in [2] the authors use the channel matrix model in [32] to demonstrate that the complexity of sphere detection increases rapidly with increasing spatial correlation at the receiver or transmitter. We are therefore motivated to understand how spatial correlation is modeled and to examine how spatial correlation affects the complexity of LR processing. We then find that the apparent increase in LR complexity is unnecessary when we examine LR in the context of MIMO detection.

6.1 Channel Model with Spatial Correlation

In [32] the effects of correlation at both the transmitter and receiver are modeled by pre-multiplying and post-multiplying the channel matrix in (1) with the matrix square root of covariance matrices. We let $\mathbf{\Sigma}_{rx}$ and $\mathbf{\Sigma}_{tx}$ be the positive definite complex receive and transmit covariance matrices with dimensions $N_r \times N_r$ and $N_t \times N_t$, respectively. We also let $\boldsymbol{H}'$ be an $N_r \times N_t$ complex matrix having independent identically distributed (i.i.d.) complex Gaussian distributed elements with zero mean

and unit variance. Then we can define the correlated channel model as follows:

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{s} + \eta\boldsymbol{w}, \tag{60}$$

$$\boldsymbol{H} = \boldsymbol{\Sigma}_{rx}^{\frac{1}{2}}\boldsymbol{H}'\boldsymbol{\Sigma}_{tx}^{\frac{1}{2}}, \tag{61}$$

where η, $\boldsymbol{w}$, $\boldsymbol{y}$, and $\boldsymbol{s}$ are defined as in (1). Conveniently, a software implementation of this channel model for a linear array of antennas is available at [12], which is part of the IST-2000-30148 I-METRA project [65]. A top-level function in this implementation allows the user to generate the covariance matrices based on the relative antenna spacing, angle of arrival (AoA), and angular spread (AS) parameters.

6.2 Effect of Correlation on CLLL-MMSE-SIC

We now apply our understanding of both the channel model and the CLLL-MMSE-SIC detection algorithm to examine how spatial correlation at the receiver and transmitter affects the complexity of LR processing. We assume that the carrier frequency is 5 GHz [23], AoA is 45 degrees, the AS is 40 degrees, and the antennas at the receiver and transmitter are spaced uniformly in a linear array. Under this configuration, we let ρ_{tx} and ρ_{rx} be the adjacent-antenna correlation coefficients at the transmitter and receiver, respectively. Additionally, we let $N_t = N_r = 4$, and we use the worst-case $\sigma_w = 0$ choice for the augmented channel matrix in (6). We choose this worst case such that we do not benefit from the complexity reduction normally achieved by incorporating a noise estimate into the detection algorithm. For simplicity we assume that the spatial correlation at the transmitter and receiver is identical.

Using the software implementation in [12], we generate 4×4 $\boldsymbol{H}$ matrices for various spatial correlations between adjacent antennas (achieved by varying the relative antenna spacing) and execute the CLLL algorithm for each channel matrix realization. Figure 12 shows the results of this initial experiment in terms of both algorithm complexity measures and hardware cycles of the proposed architecture in Chapter 5. In addition to the average number of CLLL algorithm iterations, we also consider

the average number of basis updates as a relevant measure of algorithm complexity (executions of Lines 10-12 in Table 1). The results in Figure 12 demonstrate that the average complexity of the CLLL algorithm increases rapidly as the spatial correlation increases, resulting in a similar increase in hardware latency. From this initial experiment we also conclude that without additional algorithm enhancements to the modified CLLL algorithm in Chapter 3, additional CLLL processors will need to be utilized to maintain the throughput and latency reported for the uncorrelated channel case in Chapter 5.

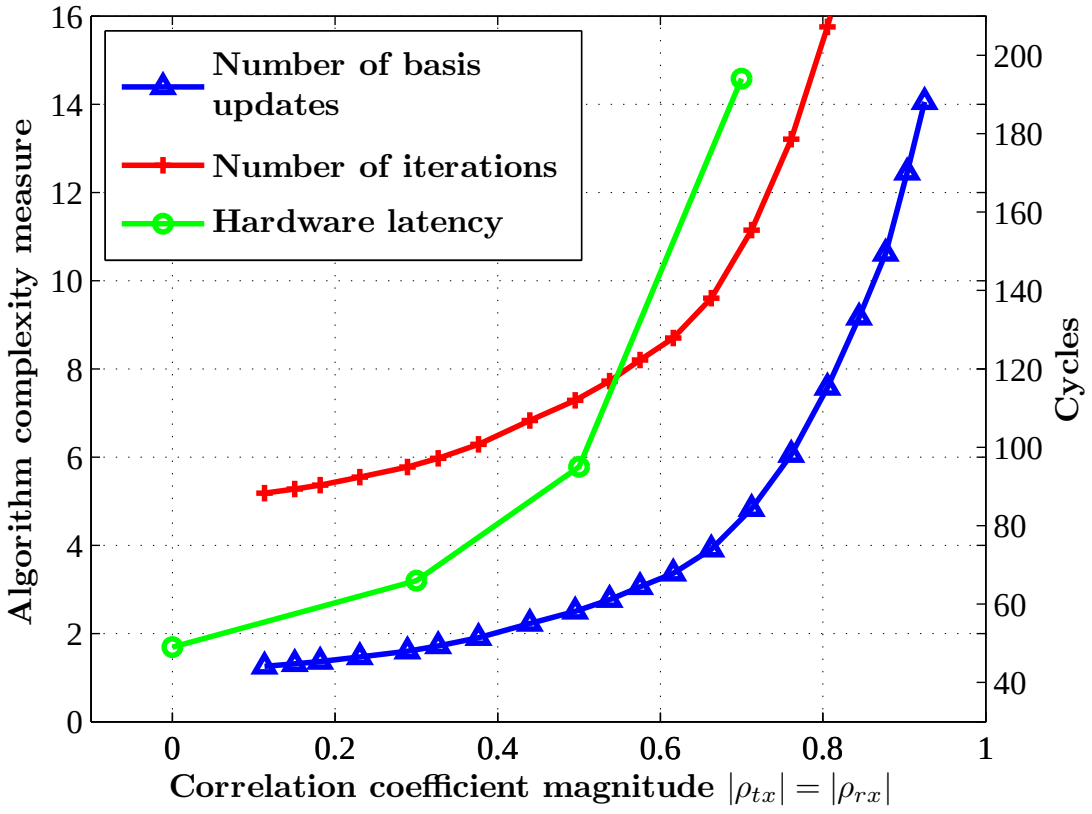

Figure 12: Effect of spatial correlation on the algorithm complexity of the CLLL algorithm for 4×4 $\boldsymbol{H}$ matrices and hardware latency of the proposed CLLL architecture in Chapter 5. The average number of basis updates and iterations for each spatial correlation case is obtained from simulations of one million channel realizations.

6.3 Ideal Early Termination

We are therefore motivated to determine if these additional CLLL iterations and basis updates are necessary when spatial correlation is present. More importantly from a communication system perspective, we desire to determine if the observed increase in hardware latency in Figure 12 is necessary in the context of MIMO detection.

Towards this goal, we consider a hypothetical experiment. Suppose that for a given $\bar{\boldsymbol{H}}$ and $\boldsymbol{y}$ we first run the CLLL algorithm to completion, obtaining an estimate $\hat{\boldsymbol{s}}$ of the transmitted symbol vector. We refer to this estimate as the actual $\hat{\boldsymbol{s}}$. Now imagine rerunning the CLLL algorithm again on $\bar{\boldsymbol{H}}$. At the end of each iteration, however, we complete the CLLL-MMSE-SIC detection process (using the intermediate $\tilde{\boldsymbol{R}}$, $\tilde{\boldsymbol{Q}}$, and $\boldsymbol{T}$) to obtain a symbol vector estimate for each iteration. We early terminate the CLLL algorithm when the symbol vector estimate at the end of an iteration exactly matches the actual $\hat{\boldsymbol{s}}$. This experiment therefore provides us with information about the required number of CLLL basis updates.

We run this experiment under the same assumptions as the experiment in Section 6.2 except that we vary $1/{\sigma_w}^2$ (SNR values). For each spatial correlation and SNR case, we record the average number of basis updates for both the CLLL algorithm (no early termination) and the CLLL algorithm with ideal early termination (required basis updates). Figure 13 contains the results of this experiment for 64-QAM transmissions. We report the average number of required basis updates as a fraction of average number of CLLL algorithm basis updates. Even when the spatial correlation is high, i.e., $|\rho_{tx}| = |\rho_{rx}| = 0.7$, on average most CLLL algorithm basis updates are unnecessary for reasonable SNR values.

Hence, early termination of the CLLL algorithm appears fundamentally possible in the context of MIMO detection. Given that Figure 12 indicates that the latency of the proposed CLLL architecture is roughly proportional to the number of basis updates, it also should be possible to reduce the latency of the proposed CLLL architecture.

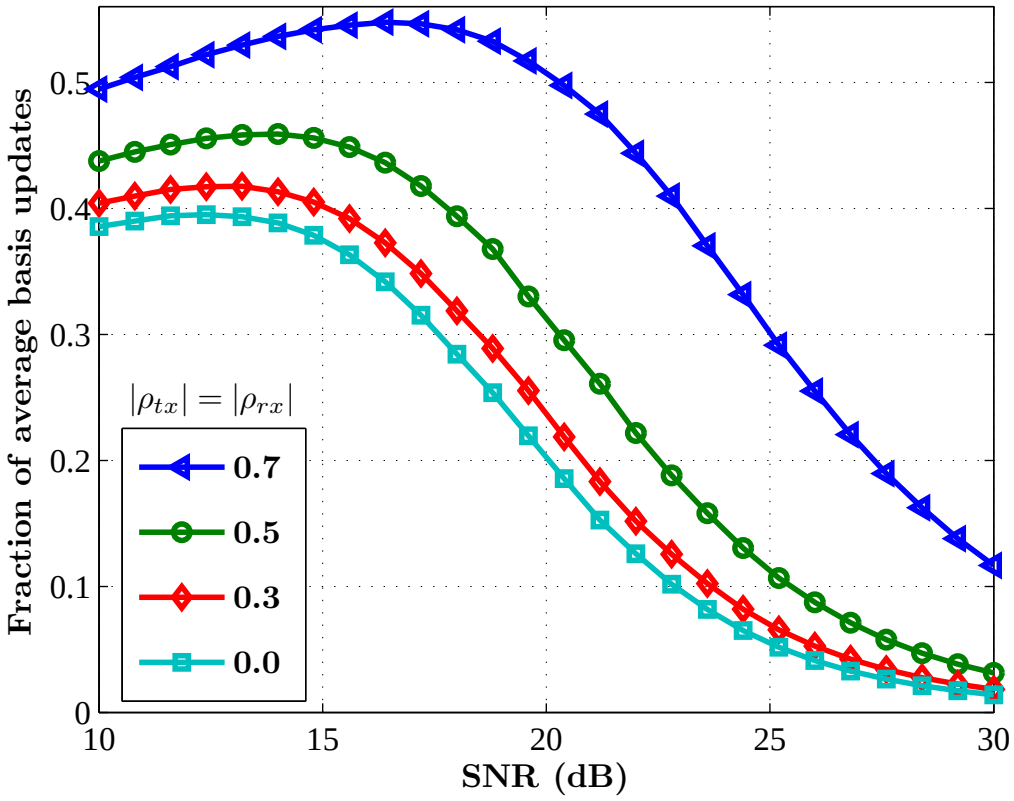

Figure 13: Fraction of average number of original CLLL algorithm basis updates that are required when the ideal early termination scheme in the genie experiment is employed for CLLL-MMSE-SIC detection. Under the assumptions in Section 6.3, the correlation coefficient magnitudes of 0.3, 0.5, and 0.7 correspond to 6.6 cm, 3.9 cm, and 2.1 cm antenna spacings, respectively. For each SNR a sufficient number of channel realizations is simulated to generate a minimum of 4000 bits errors.

CHAPTER VII

DEVELOPING INCREMENTAL LR

Given that the results in the preceding chapter demonstrate that early termination of LR algorithms is possible, we are motivated to develop a practical early termination scheme for LR-aided detectors. We base this early termination scheme on the hybrid detector concept in [68]. After reviewing this work, we present novel analytical results that are based on the spatial correlation channel model in the preceding chapter. Application of these analytical results then leads us to introduce and to develop incremental LR (ILR), which achieves full-diversity detection and enables early termination of LR algorithms. As a final step, we integrate the utilized early termination condition computation with both the LR and symbol detection computations, resulting in the proposed incremental CLLL-MMSE-SIC algorithm.

7.1 Hybrid Detectors

As a step toward early termination in the CLLL algorithm, we examine the hybrid detector in [68], which is a combination of a low-complexity detection algorithm and a high-complexity full-diversity detection algorithm. In this section we rigorously analyze this hybrid detector and demonstrate that it enables full-diversity detection, forming the foundation for its subsequent application to CLLL algorithm early termination.

7.1.1 General Description

The hybrid detector in [68] employs a reliability assessment (RA) to determine if the initial symbol vector estimate generated by a low-complexity detection algorithm is reliable enough to maintain ML BER performance. If the symbol vector estimate is

not sufficiently reliable, then a high-complexity algorithm is employed that exhibits ML BER performance. To formalize and to generalize this method we let $\hat{\boldsymbol{s}}_{\text{init}}$ be a symbol vector estimate obtained from any low-complexity detection method and $\hat{\boldsymbol{s}}_{\text{sec}}$ be the symbol vector estimate obtained from a higher-complexity detection method that exhibits improved BER performance. Then we can define the general hybrid detector as

$$\hat{\boldsymbol{s}} = \begin{cases} \hat{\boldsymbol{s}}_{\text{init}} & \|\boldsymbol{y} - \boldsymbol{H}\hat{\boldsymbol{s}}_{\text{init}}\| \leq A\sigma_w, \\ \hat{\boldsymbol{s}}_{\text{sec}} & \text{otherwise}, \end{cases} \tag{62}$$

where A is a positive, fixed threshold parameter. The authors in [68] demonstrate through simulations that they achieve full-diversity detection when they determine $\hat{\boldsymbol{s}}_{\text{sec}}$ using a full-diversity detection method.

7.1.2 Full Diversity Detection

Before adopting this hybrid detector, however, it is instructive to understand why this hybrid detector can collect full diversity for the system model in (60). We first define the random vector $\boldsymbol{d} = \boldsymbol{y} - \boldsymbol{H}\hat{\boldsymbol{s}}$. Then we can condition the probability of error $P\{\hat{\boldsymbol{s}} \neq \boldsymbol{s}\}$ on $\boldsymbol{d}$:

$$\begin{aligned} P\{\hat{\boldsymbol{s}} \neq \boldsymbol{s}\} = P\{\hat{\boldsymbol{s}} \neq \boldsymbol{s} \mid \|\boldsymbol{d}\| \leq A\sigma_w\}P\{\|\boldsymbol{d}\| \leq A\sigma_w\} + \\ P\{\hat{\boldsymbol{s}} \neq \boldsymbol{s} \mid \|\boldsymbol{d}\| > A\sigma_w\}P\{\|\boldsymbol{d}\| > A\sigma_w\}. \end{aligned} \tag{63}$$

Assuming that we employ a full-diversity detection method for the case when $\|\boldsymbol{d}\| > A\sigma_w$, we need only examine the first term in (63). Since this first term can be written as $P\{\|\boldsymbol{d}\| \leq A\sigma_w, \hat{\boldsymbol{s}} \neq \boldsymbol{s}\}$, we need only consider the joint pairwise error probability $P\{\|\boldsymbol{d}\| \leq A\sigma_w, \boldsymbol{s} \to \hat{\boldsymbol{s}}\}$, where $\boldsymbol{s} \neq \hat{\boldsymbol{s}}$. This task is simplified by noticing that because $P\{\boldsymbol{s} \to \hat{\boldsymbol{s}}\} \leq 1$ it follows that $P\{\|\boldsymbol{d}\| \leq A\sigma_w, \boldsymbol{s} \to \hat{\boldsymbol{s}}\} \leq P\{\|\boldsymbol{d}\| \leq A\sigma_w \mid \boldsymbol{s} \to \hat{\boldsymbol{s}}\}$. Finally, after consideration of the following lemma, it is clear that the hybrid detector exhibits full diversity for the system model in (60):

Proposition 2 *Given the system model in* (60)*, it follows that* $P\{\|\boldsymbol{d}\| \leq A\sigma_w \mid \boldsymbol{s} \rightarrow \hat{\boldsymbol{s}}\}$ *is upper bounded by the following:*

$$(2\eta M_A)^{2N_r} B_e \left(\frac{1}{\sigma_w^2}\right)^{-N_r}, \tag{64}$$

where

$$B_e = \frac{1}{\left(\pi \left\| \boldsymbol{\Sigma}_{tx}^{\frac{1}{2}} (\boldsymbol{s} - \hat{\boldsymbol{s}}) \right\|^2\right)^{N_r} |\det(\boldsymbol{\Sigma}_{rx})|}, \tag{65}$$

$$M_A = \max\left(\frac{16}{3\eta}A, 4\sqrt{\pi}Q\left(\frac{8A}{3\eta\sqrt{2}}\right)\right). \tag{66}$$

Proof: See Appendix D.

7.1.3 Using CLLL-MMSE-SIC Detection in a Hybrid Detector

We can form the hybrid detector in (62) by using the MMSE-SIC detection result for $\hat{\boldsymbol{s}}_{\text{init}}$ and the CLLL-MMSE-SIC detection result for $\hat{\boldsymbol{s}}_{\text{sec}}$. To determine the reduction in average number of basis updates that this CLLL-MMSE-SIC-based hybrid detector achieves, we utilize system simulations under the assumptions in Section 6.3. Since the main idea here is to determine the ideal best-case performance of this hybrid detector, we manually choose the A threshold parameters. More specifically, for each correlation case, we first run multiple simulations to determine an A that results in 10% higher BER than if the simulation is run with $A = 0$. Then we record the number of basis updates executed when the chosen A is used.

The results of this experiment are contained in Figure 14. We plot the ratio of the average number of basis updates for the hybrid detector to the average number of basis updates when we employ the ideal early termination in Section 6.3. Since the majority of the plotted ratios are greater than unity, it is clear that the hybrid detector alone does not realize the full early termination potential demonstrated in Section 6.3. Consequently, we seek additional modifications to the CLLL algorithm.

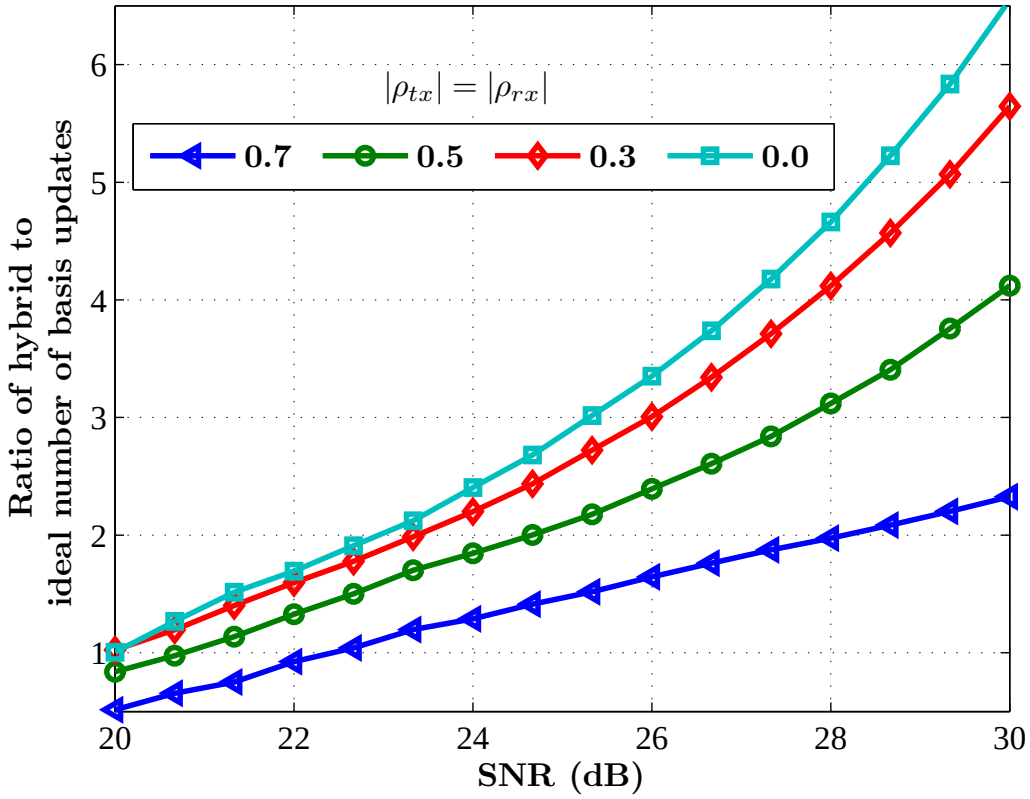

Figure 14: Effectiveness of a CLLL-MMSE-SIC-based hybrid detector at reducing the average number of executed basis updates. This is demonstrated by plotting the ratio of the average number of basis updates when the hybrid detector is used to the average number of basis updates when the ideal early termination condition from Section 6.3 is used. For each SNR a sufficient number of channel realizations is simulated to generate a minimum of 4000 bits errors.

7.2 Incremental Lattice Reduction

A natural progression from the CLLL-MMSE-SIC-based hybrid detector is the repeated application of this hybrid detector during the CLLL algorithm execution. More specifically, after each CLLL iteration, we complete the CLLL-MMSE-SIC detection process (using the intermediate $\tilde{\boldsymbol{R}}$, $\tilde{\boldsymbol{Q}}$, and $\boldsymbol{T}$) to obtain a symbol vector estimate $\hat{\boldsymbol{s}}_{\text{int}}$. Then the next iteration of the CLLL algorithm is only initiated when $\|\boldsymbol{y} - \boldsymbol{H}\hat{\boldsymbol{s}}_{\text{int}}\| > A\sigma_w$. We refer to this simultaneous process of symbol detection and LR as incremental LR (ILR) because the lattice is gradually reduced over multiple

received vectors in a packet. In this section we formulate incremental LR specifically for CLLL-MMSE-SIC detection by considering practical modifications to this general idea.

7.2.1 Modifications for CLLL-MMSE-SIC

It is beneficial from a complexity perspective (and hardware perspective) to modify and to integrate the RA calculation with the matrices involved in the CLLL-MMSE-SIC detection process. Towards this goal we first consider an RA based on the extended system:

$$\|\bar{\boldsymbol{y}} - \bar{\boldsymbol{H}}\hat{\boldsymbol{s}}_{\text{int}}\| \leq \bar{A}\sigma_w, \tag{67}$$

where $\bar{\boldsymbol{y}} = [\boldsymbol{y}^T, \boldsymbol{0}_{1\times N_t}]^T$ and $\bar{A}$ is a positive, finite threshold parameter. Starting with the lefthand side of (67), we see that the following is true:

$$\|\bar{\boldsymbol{y}} - \bar{\boldsymbol{H}}\hat{\boldsymbol{s}}_{\text{int}}\|^2 = \|\boldsymbol{y} - \boldsymbol{H}\hat{\boldsymbol{s}}_{\text{int}}\|^2 + \sigma_w^2\|\hat{\boldsymbol{s}}_{\text{int}}\|^2 > \|\boldsymbol{y} - \boldsymbol{H}\hat{\boldsymbol{s}}_{\text{int}}\|^2,$$

which demonstrates that the RA in (67) is sufficient to guarantee full-diversity detection. Additionally, since $\hat{\boldsymbol{s}}_{\text{int}}$ only changes when a basis update occurs [21, Appendix A], the RA in (67) need only be reevaluated after each basis update (After the initial RA evaluation that occurs before the beginning of the LR processing). We then consider the following observations:

The RA can be evaluated in the z-domain. Assuming that no symbol quantization error occurs during the application of the $\mathcal{Q}_{\mathcal{S}}$ function in the CLLL-MMSE-SIC detection process, i.e., $\hat{\boldsymbol{s}} = 2\boldsymbol{T}\hat{\boldsymbol{z}} - (1+j)\,\boldsymbol{1}_{N_t\times 1}$, we can use the factorization returned by the CLLL algorithm to express the relation in (67) as

$$\left\|\tilde{\boldsymbol{R}}\hat{\boldsymbol{z}} - \frac{1}{2}\left(\left(\tilde{\boldsymbol{Q}}^{(1)}\right)^{\mathcal{H}}\boldsymbol{y} + \tilde{\boldsymbol{R}}\boldsymbol{T}^{-1}\,(1+j)\,\boldsymbol{1}_{N_t\times 1}\right)\right\|^2 \leq \frac{1}{4}\left(\bar{A}\sigma_w\right)^2. \tag{68}$$

Hence, we can evaluate the RA in (67) in the z-domain and integrate this evaluation with the SIC computation that we use to find $\hat{\boldsymbol{z}}$ in (16).

The SIC and RA computation often need only be partially (re)computed. The fully-populated $\hat{\boldsymbol{z}}$ is only required for the final computation of $\hat{\boldsymbol{s}}$. Therefore, the complexity of both the intermediate SIC and RA computation can be reduced by terminating the SIC detection in the z-domain when the RA is not satisfied. We can implement this termination efficiently by evaluating the RA as the SIC detection proceeds. We also notice that when a basis update occurs during an iteration of the CLLL algorithm for a particular k in Table 1, the subsequent SIC detection result is unaffected in dimensions higher than k (compared to SIC detection before the basis update). Therefore, we store and reuse the partially computed $\hat{\boldsymbol{z}}$ and partially computed lefthand side of (68). This modification allows us to start the joint SIC/RA computation required after each basis update at a $\hat{\boldsymbol{z}}$ element having an index that is possibly less than N_t.

The SIC and RA computation can be decoupled from basis updates on $\tilde{\boldsymbol{Q}}$. Clearly, the $N_t \times 1$ vector in the righthand side of (16) must be computed before any SIC computation may proceed. After a basis update, this vector must be recomputed before the updated $\tilde{\boldsymbol{R}}$ can be used on the lefthand side of (16) in the SIC and RA computation. Careful examination of (16) reveals that during an iteration of the CLLL algorithm for a particular k, only the $(k-1)$-th and k-th rows of this vector are affected by a basis update. We treat these two elements as a 2×1 vector. It is then apparent that the updated row elements can be found by premultiplying this 2×1 vector by the $\boldsymbol{\Theta}$ matrix computed at Line 10 of Table 1.

This modification is also utilized in [79] such that basis updates on $\tilde{\boldsymbol{Q}}$ are completely eliminated. Our motivation for this modification is different, however. We do not remove the basis updates on $\tilde{\boldsymbol{Q}}$ (and updates on $\boldsymbol{g}$ in Line 7) because then the righthand side of (16) can be computed efficiently using $\tilde{\boldsymbol{Q}}$ for subsequent received vectors in a packet. The $\tilde{\boldsymbol{Q}}$ generated during CLLL processing of a received vector is not required until the next received vector in the packet. As a result, this modification

essentially decouples the SIC and RA computation from basis updates on $\tilde{\boldsymbol{Q}}$.

7.2.2 Proposed Algorithm

The synthesis of these ideas leads to the joint LR and symbol detection algorithm listing in Table 3. We partition the code into initialization operations that occur once per packet (Line 1-2) and operations that occur for each received vector $\boldsymbol{y}$ (Line 3-33). For the first received vector in a packet, the $\tilde{\boldsymbol{Q}}$, $\tilde{\boldsymbol{R}}$, and $\boldsymbol{T}$ from the packet initialization are used in Line 3-4. For subsequent received vectors in the packet, the $\tilde{\boldsymbol{Q}}$, $\tilde{\boldsymbol{R}}$, $\boldsymbol{T}$, and $\boldsymbol{g}$ that is computed during the processing of the previous received vector in the packet are used in Line 3-4. More generally, the partially-reduced lattice is used to initialize the received vector processing after the first received vector in the packet is processed.

The received vector processing consists of repeatedly executing the CLLL_ITER function in Line 29, which computes one iteration of the CLLL algorithm (Line 3-18 in Table 1) given the current k and partially-reduced lattice. This function outputs the updated k and updated input matrices. It also outputs the *update* variable, indicating if a basis update occurs during that iteration, and the 2×2 matrix $\boldsymbol{\Theta}$ for a possible basis update. This matrix is used to update the $\bar{\boldsymbol{b}}$ vector, which stores the righthand side of (16). The $\bar{\boldsymbol{b}}$ vector is used in the partial SIC and RA computation in Line 15-23. To enable early termination of this computation, we track the current required starting index of the SIC computation in the l variable and track the partial computation of the lefthand side of (68) in the $\boldsymbol{a}$ vector. Lastly, we note that this particularly straightforward integration is enabled by incorporating the Siegel condition reevaluation in Line 7-14, which is easily implemented in hardware [21].

Table 3: Incremental CLLL-MMSE-SIC based on the Siegel Condition

(1)	$[\tilde{\boldsymbol{Q}}^{(1)}, \tilde{\boldsymbol{Q}}^{(2)}, \tilde{\boldsymbol{R}}, \boldsymbol{T}] = \text{SORTED\_QR}(\bar{\boldsymbol{H}})$	***Packet Initialization***
(2)	$\boldsymbol{g} = (1+j)\,\mathbf{1}_{N_t \times 1};\ k = 2;\ \tilde{\boldsymbol{Q}} = \tilde{\boldsymbol{Q}}^{(1)}$	
(3)	$\bar{\boldsymbol{b}} = \frac{1}{2}\left(\tilde{\boldsymbol{Q}}^{\mathcal{H}}\boldsymbol{y} + \tilde{\boldsymbol{R}}\boldsymbol{g}\right)$	
(4)	$\gamma = \frac{1}{4}\left(\bar{A}\sigma_w\right)^2;\ a_{(N_t+1)} = 0;\ l = N_t;\ update = \text{true}$	
(5)	while (1)	
(6)	if *update*	
(7)	for $n = k : 1 : N_t$	***early Siegel condition evaluation***
(8)	if $\lvert\tilde{R}_{n-1,n-1}\rvert > \sqrt{\zeta}\lvert\tilde{R}_{n,n}\rvert$	
(9)	break;	
(10)	end	
(11)	if $n == N_t$	
(12)	$\gamma = -1$	
(13)	end	
(14)	end	
(15)	for $n = l : -1 : 1$	***partial SIC and RA computation***
(16)	$\Delta = \bar{b}_n - \Sigma_{j=n+1}^{N_t} \tilde{R}_{n,j}\hat{z}_j$	
(17)	$\hat{z}_n = \text{round}(\Delta / \tilde{R}_{n,n})$	
(18)	$l = n - 1$	
(19)	if $\gamma > 0$	
(20)	$a_n = a_{n+1} + \lvert\Delta - \tilde{R}_{n,n}\hat{z}_n\rvert^2$	
(21)	if $a_n > \gamma \rightarrow$ break	
(22)	end	
(23)	end	
(24)	if $\gamma < 0 \ \ \lvert\rvert \ \ a_{l+1} \leq \gamma$	
(25)	$\hat{\boldsymbol{s}} = \mathcal{Q}_{\mathcal{S}}\left[2\boldsymbol{T}\hat{\boldsymbol{z}} - (1+j)\,\mathbf{1}_{N_t \times 1}\right]$	
(26)	if $\gamma < 0 \ \ \lvert\rvert \ \ \hat{\boldsymbol{s}} == (2\boldsymbol{T}\hat{\boldsymbol{z}} - (1+j)\,\mathbf{1}_{N_t \times 1}) \rightarrow$ break	
(27)	end	
(28)	end	
(29)	$[k, update, \boldsymbol{\Theta}, \tilde{\boldsymbol{R}}, \tilde{\boldsymbol{Q}}, \boldsymbol{T}, \boldsymbol{g}] = \text{CLLL\_ITER}(k, \tilde{\boldsymbol{R}}, \tilde{\boldsymbol{Q}}, \boldsymbol{T}, \boldsymbol{g})$	
(30)	if *update*	
(31)	$l = \max(l, k+1);\ \bar{\boldsymbol{b}}_{k:k+1,1} = \boldsymbol{\Theta}\bar{\boldsymbol{b}}_{k:k+1,1}$	
(32)	end	
(33)	end	

CHAPTER VIII

EVALUATING INCREMENTAL LR FROM ALGORITHM AND HARDWARE PERSPECTIVES

In this chapter we evaluate the effectiveness of the proposed algorithm, incremental CLLL-MMSE-SIC, at reducing the average number of basis updates. Given that the proposed algorithm requires additional overhead, we also consider the relative contribution of the overhead to the the number of arithmetic operations under the proposed method. Before either evaluation can proceed, however, we must choose $\bar{A}$, which clearly affects the operation of the proposed algorithm. When $\bar{A} = 0$, the proposed algorithm simplifies to the original CLLL-MMSE-SIC. Conversely, when $\bar{A} = \infty$, CLLL processing is never applied and the proposed algorithm simplifies to MMSE-SIC. We first determine $\bar{A}$ for the single received vector per packet case. Then we use this $\bar{A}$ to evaluate the proposed algorithm for a variety of spatial correlation cases and packet sizes.

8.1 Choosing the $\bar{A}$ Parameter

To determine a suitable $\bar{A}$, we explore the space of spatial correlation cases for the system model in (1). We apply the assumptions in Section 6.3 but do not restrict the exploration to the artificial constraint of equal transmitter and receiver spatial correlation. We determine a general guideline for which spatial correlation cases to consider by first setting $\bar{A} = 0$, fixing the $|\rho_{tx}|$ parameter and then searching for the $|\rho_{rx}|$ that results in approximately 10^{-3} BER at an SNR of 33 dB. For each $|\rho_{rx}|$ considered when $|\rho_{tx}|$ is fixed, a sufficient number of channel realizations is simulated to produce 8000 bit errors. In general the SNR required to maintain a given BER

increases as the degree of spatial correlation increases [2]. Therefore, this correlation case determination exercise provides us with the largest $|\rho_{rx}|$ that we should examine for a given $|\rho_{tx}|$.

For each spatial correlation case that we consider, we determine an $\bar{A}$ that results in at most a 10% BER degradation at 10^{-3} BER. To accelerate this $\bar{A}$ determination process, a straightforward adaptive and automated search method is employed. For each $\bar{A}$ parameter value considered in this search, a sufficient number of channel realizations is simulated to produce a minimum of four million bit errors, reducing the variance of the BER estimate. Given the large number of channel realizations simulated, we must utilize a pseudo random number generator (PRNG) that has a long period. The PRNG in [47], which has a period of $2^{19937} - 1$, is a suitable choice.

The determined $\bar{A}$ for a variety of correlation cases is contained in Figure 15. In addition, the shaded area in the figure indicates the results of the correlation case determination exercise. We obtain an approximate boundary of this shaded region by considering each $|\rho_{rx}|$ and $|\rho_{tx}|$ pair generated during the correlation case determination exercise. For each of these transmitter and receiver correlation pairs, we complete the $\bar{A}$ determination process and then plot the determined $\bar{A}$ for each $|\rho_{rx}|$ in this pair.

These results—the $\bar{A}$ determination process and correlation case determination exercise—when taken together indicate that for practical values of transmitter and receiver spatial correlation, the $\bar{A}$ parameter does not vary significantly. It is also apparent that lower $\bar{A}$ values are required for higher spatial correlation. From Figure 15, we observe that the choice $\bar{A} = 18.2$ is sufficient for the majority of correlation cases. We further validate this choice by empirically verifying that the BER to SNR curve of the proposed algorithm with $\bar{A} = 18.2$ is parallel to the BER to SNR curve of the original CLLL-MMSE-SIC algorithm for all correlation cases considered. An example plot that demonstrates this behavior for a variety of correlation cases is

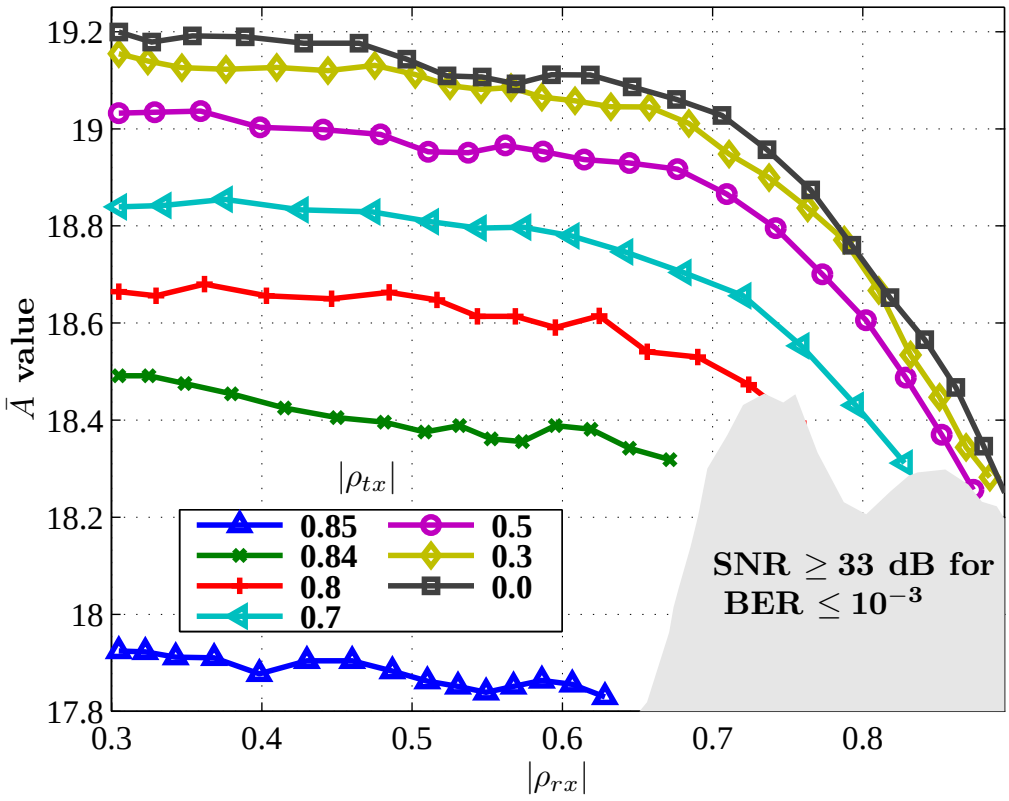

Figure 15: Determined $\bar{A}$ parameter for a variety of spatial correlation cases under the assumptions in Section 6.3. These assumptions include a 4×4 MIMO system, 64-QAM signal constellation, AoA of 45 degrees, AS of 40 degrees, and a linear array of antennas at both the receiver and transmitter.

contained in Figure 16.

8.2 Reduction in Average Number of Basis Updates

Setting $\bar{A} = 18.2$, we are now able to evaluate the effectiveness of the proposed algorithm at reducing the average number of basis updates. Our evaluation approach involves comparing the average number of basis updates of the proposed algorithm and original CLLL algorithm for identical SNR values. Then we utilize additional simulations of the proposed algorithm to determine the SNR penalty incurred for this reduction in average number of basis updates. More concretely, under the assumption of a 4×4 system with a 64-QAM signal constellation, we complete the following for

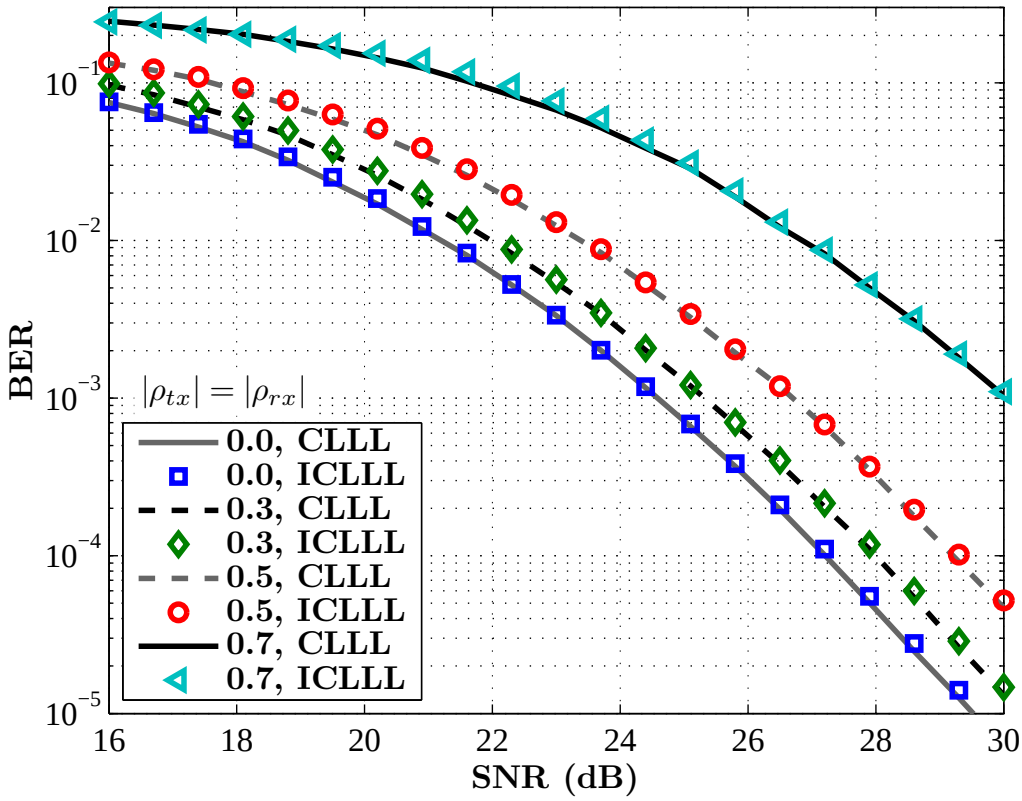

Figure 16: 4x4 64-QAM BER performance for both the proposed algorithm with $\bar{A} = 18.2$ (ICLLL) and the original CLLL-MMSE-SIC algorithm (CLLL).

a given correlation case:

1. Determine the SNR required when $\bar{A} = 0$ to achieve 10^{-3} BER;
2. Simulate the proposed algorithm at this SNR with $\bar{A} = 18.2$ and record the average number of basis updates;
3. Determine the SNR required when $\bar{A} = 18.2$ to achieve 10^{-3} BER.

For each of these tasks, we simulate a sufficient number of channel realizations to produce a minimum of 8000 bit errors.

The graphs in Figure 17 contain the results of this evaluation for each correlation case identified in Section 8.1. We additionally include results for $|\rho_{tx}| = 0.89$ with

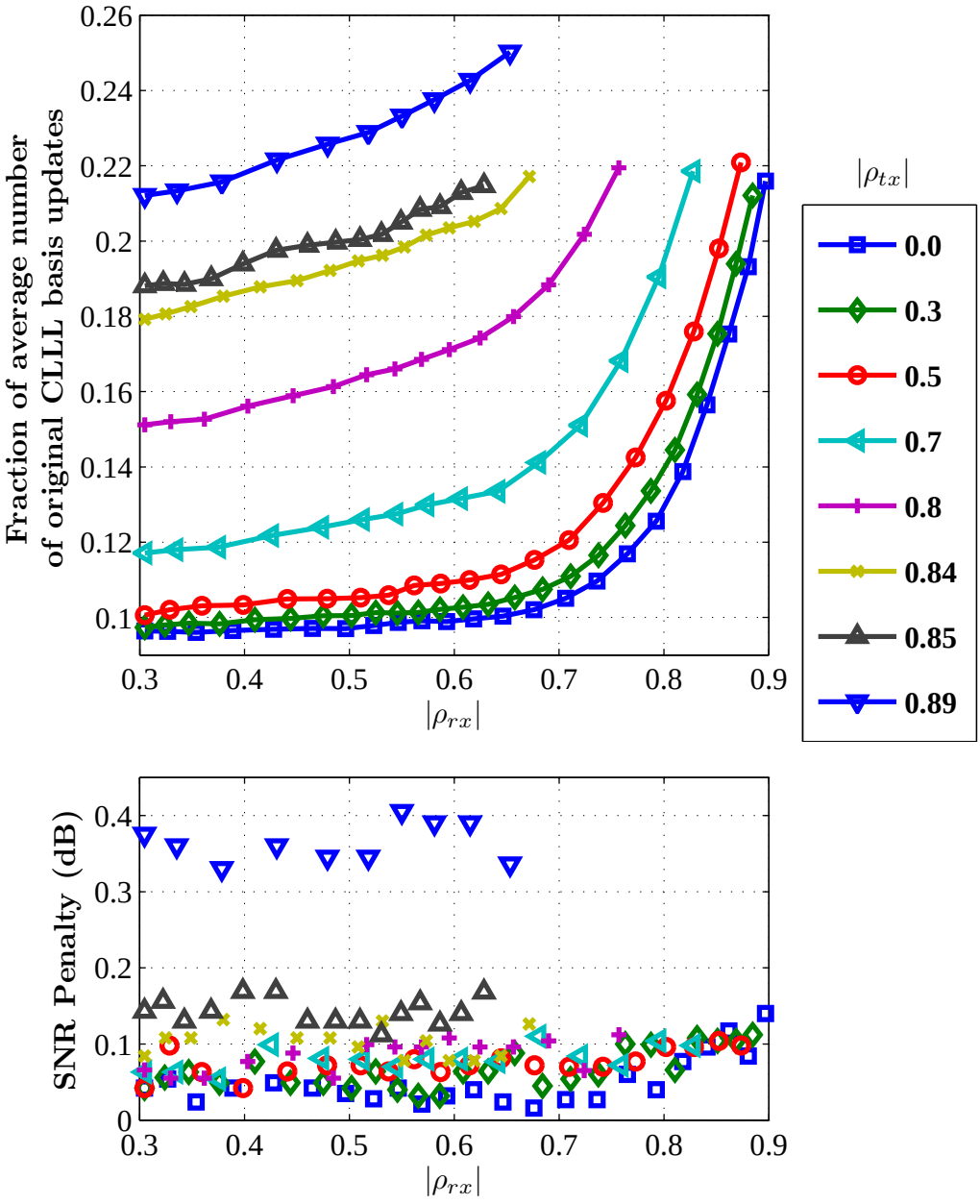

Figure 17: Average number of basis updates required by the proposed algorithm (top) and corresponding SNR penalty for a variety of correlation cases.

$|\rho_{rx}|$ between 0.3 and 0.65. These additional results allow us to evaluate the proposed algorithm for correlation cases that are not identified in Section 8.1 (an SNR greater than 33 dB is required to achieve 10^{-3} BER). We examine the ratio of the average

number of basis updates required by the proposed algorithm to the average number of basis updates required by the original CLLL algorithm and the increased SNR penalty incurred by the proposed algorithm relative to the original CLLL algorithm to achieve 10^{-3} BER. From these figures it is then apparent that for the correlation cases in Section 8.1, the proposed algorithm requires only $10\% - 25\%$ of the number of average basis updates, depending on the spatial correlation. The SNR penalty for this reduction is at most 0.17 dB. Furthermore, the performance of the algorithm for the high correlation cases appears to be reasonable; a 0.4 dB penalty in SNR is incurred in exchange for over a 75% reduction in the average number of basis updates when $|\rho_{tx}| = 0.89$.

In this evaluation, however, the AoA and AS parameters are fixed at 45 degrees and 40 degrees, respectively. Therefore, we next determine how varying these parameters affects these results. In this additional experiment, we fix both $|\rho_{tx}|$ and $|\rho_{rx}|$ at 0.7 for a given AoA and AS by adjusting the antenna spacing accordingly. Under the assumption of a 5 GHz carrier frequency, we choose AoA/AS pairs such that the smallest antenna spacing required to meet this correlation condition is 1.4 cm. From the basis update results in Figure 18, it is apparent that the proposed algorithm achieves a reduction in the average number of basis updates over a range of AoAs and ASs. We also note that in this experiment we obtain similar SNR penalty results as in Figure 17. For the AoA/AS pairs examined in this experiment, the proposed algorithm incurs a 0.2 dB maximum SNR penalty.

8.3 System Evaluation

From these results it is clear that the proposed algorithm effectively reduces the average number of basis updates when each transmitted packet only contains one symbol vector. In a communication system, however, packets have varying length depending on the channel conditions, data rate, and number of antennas at the transmitter and

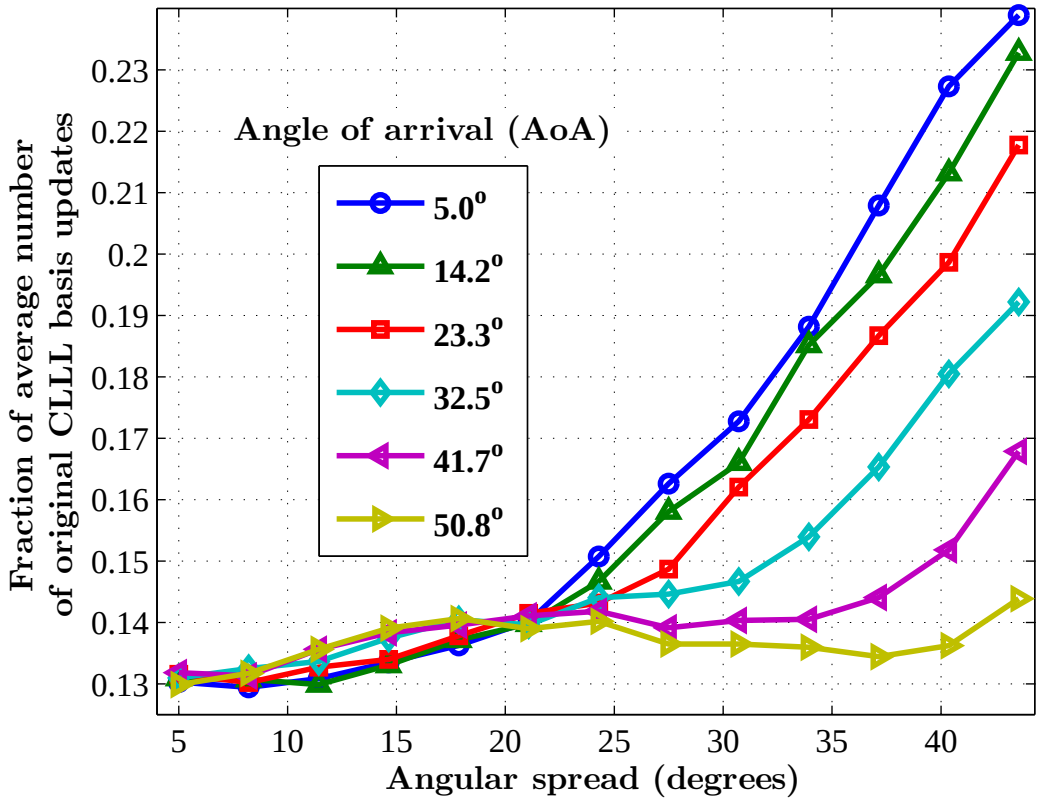

Figure 18: Proposed algorithm average number of basis updates as a fraction of original CLLL algorithm average number of basis updates for a variety of AoA and AS cases. For this experiment, both $|\rho_{tx}|$ and $|\rho_{rx}|$ are fixed at 0.7 by adjusting the antenna spacing accordingly.

receiver. For example, the 802.11n standard enables higher throughput by specifically supporting larger packet sizes than legacy standards [23]. We therefore examine the behavior of the proposed algorithm for varying packet sizes in this final evaluation.

8.3.1 Basis Updates

We follow the experimental setup in Section 8.2 but set the packet size to 20, i.e., each packet contains 20 received vectors. The processing of each received vector in a packet is initialized with the possibly partially-reduced lattice (after the first received vector is processed). As a result of this proposed algorithm feature, it is instructive to track the average number of additional basis updates executed to further reduce

the partially-reduced lattice for each received vector position.

The results of this experiment are contained in Figure 19 for a range of correlation cases.[1] From the figure we observe that the proposed algorithm distributes the LR processing over the length of the packet. For example, consider the $|\rho_{tx}| = |\rho_{rx}| = 0.7$ correlation case. For the original CLLL algorithm, the average number of basis updates is approximately 4.43. For the proposed algorithm, the average number of basis updates to complete the processing of the first received vector in the packet is 0.56. From the figure we see that the average number of additional basis updates then decreases rapidly for each subsequent received vector. Simulations of this correlation case for packet sizes greater than 100 indicate that the sum of these additional basis update averages actually converges to the average number of basis updates of the original CLLL algorithm. Through additional simulations, we confirm this behavior across all correlation cases that we consider in this article.

8.3.2 Overhead Complexity

Although this redistribution of basis update computations is desirable from a hardware realization perspective, the proposed algorithm requires additional overhead. This overhead includes the partial SIC and RA computation and the $\bar{\boldsymbol{b}}$ vector update computation (Line 31 of Table 3). To determine the complexity contribution of this overhead relative to the complexity of the proposed algorithm, we repeat the packet simulation exercise for various packet sizes and track the number of arithmetic operations in Line 3-33 of Table 3. We additionally track the number of arithmetic operations required for the overhead computations of the proposed algorithm. Here

[1]The SNR penalties that we observe in this experimental are all less than 0.01 dB. This result suggests that $\bar{A}$ could be optimized for a given packet size. To preserve the generality of our experiments, we use $\bar{A} = 18.2$, which is the choice for the single received vector per packet case.

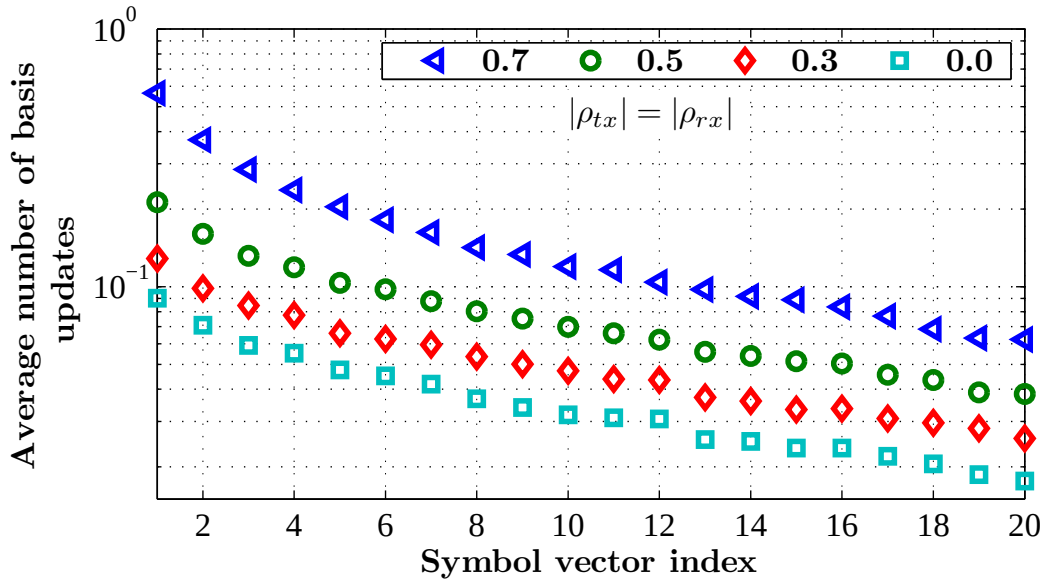

Figure 19: Average number of basis updates executed for each received vector by the proposed algorithm. In this evaluation $\bar{A} = 18.2$, the packet size is 20, and the experimental setup from Section 8.2 is utilized.

arithmetic operation refers to an addition, a multiplication, a full division, an integer-rounded division (SIC detection and size reduction operations), or a square-root operation.

The reasonableness of this complexity metric becomes apparent when we consider the results in Section 8.3.1 and how these operands are used in the proposed algorithm. Notice that the most hardware-complex operations, full division and square-root, are only required for basis updates and have the same weight as the remaining operations for the given complexity metric. Hence, for small packets sizes (less than 5 received vectors), we underestimate the complexity reduction that results from the large reduction in average number of basis updates. We further note that the integer-rounded division operation can be viewed as a reduced-precision reciprocation operation followed by a multiplication operation. Therefore assuming reciprocals are reused, which is clearly possible in the partial SIC computation, it is reasonable to treat each integer-rounded division as one multiplication.

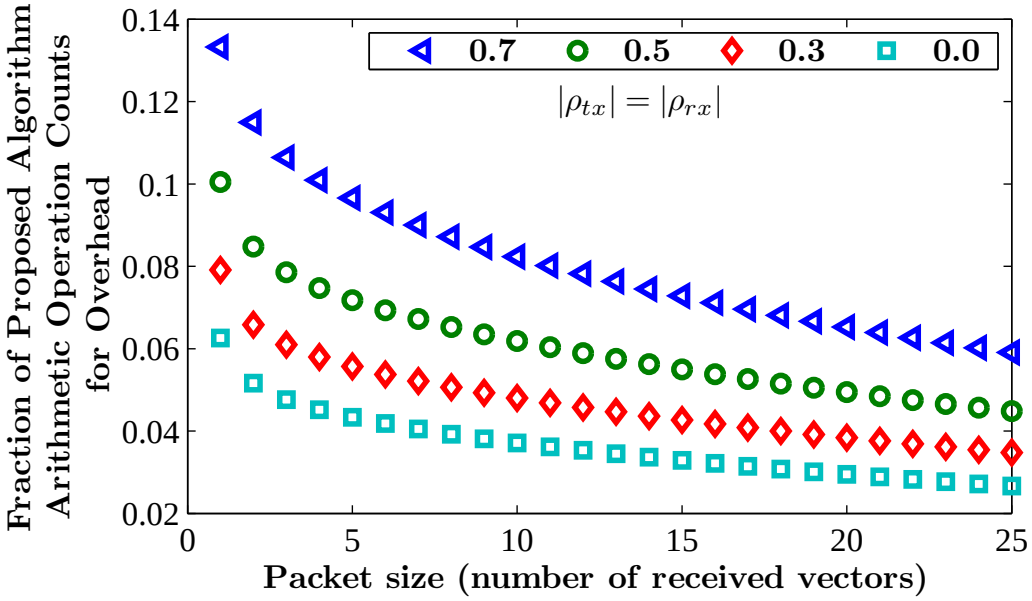

Figure 20: Contribution of overhead operations in proposed algorithm to the average number of proposed algorithm arithmetic operations. In this evaluation $\bar{A} = 18.2$ and the experimental setup from Section 8.2 is utilized.

We examine the results of this experiment in Figures 20 and 21. Figure 20 contains the complexity results of this experiment in terms of the normalized average number of proposed algorithm overhead arithmetic operations. Specifically, we normalize by the average number of proposed algorithm arithmetic operations. From the figure we observe that the overhead is less than 10% of the average number of proposed algorithm arithmetic operations for the majority of correlation and packet size cases that we examine. For the high correlation case that we examine, $|\rho_{tx}| = |\rho_{rx}| = 0.7$, this overhead is 13.3% when the packet only contains one received vector. The results in the figure also indicate that the contribution of the overhead increases with increasing spatial correlation.

Figure 21 contains the complexity results of this experiment in terms of the average number of proposed algorithm arithmetic operations that is normalized by the average number of original CLLL-MMSE-SIC algorithm arithmetics operations. From

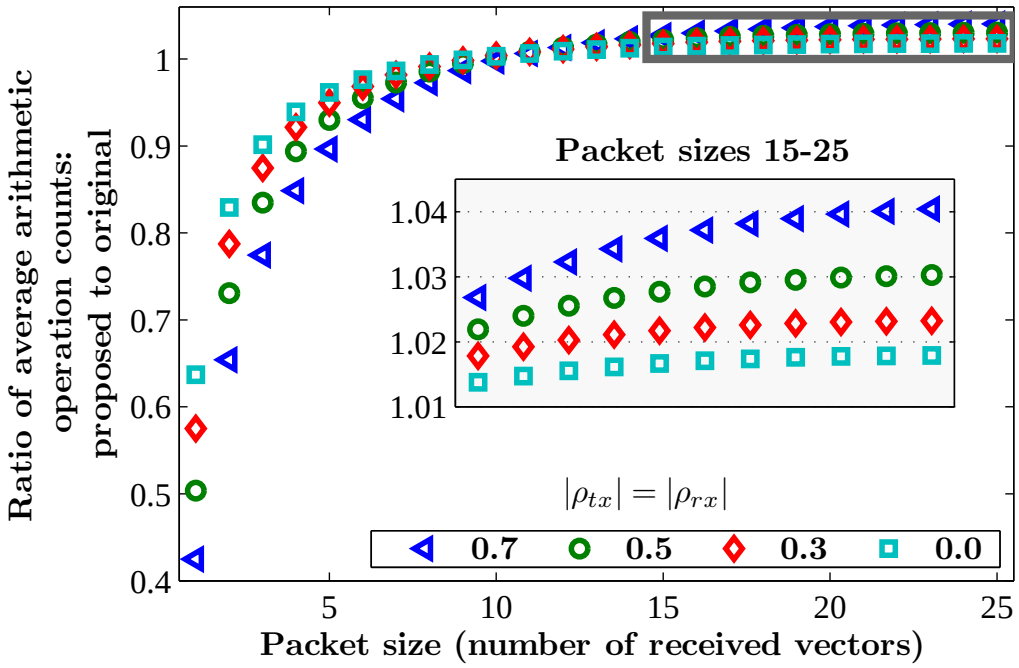

Figure 21: Average number of basis updates executed for each received vector by the proposed algorithm. In this evaluation $\bar{A} = 18.2$ and the experimental setup from Section 8.2 is utilized.

the figure we observe that the proposed algorithm achieves a reduction in average arithmetic operations for packets containing less than 5 received vectors. For packets containing more than 15 received vectors, the proposed algorithm incurs approximately less than a 4% increase in average arithmetic operations compared to the original CLLL-MMSE-SIC algorithm.

For both these figures, however, we note that we have not considered the complexity of the packet initialization operations in Table 3. Inclusion of these operations would further diminish the relative contribution of the overhead operations to the complexity of the proposed algorithm.

8.3.3 Hardware Evaluation

We could realize the proposed algorithm in hardware using a system that consists of the CLLL processor from Section 5.2 and an additional module to implement the partial SIC and RA computation. In this system, the CLLL processor outputs the $\boldsymbol{\Theta}$ matrices from basis updates, $\tilde{\boldsymbol{R}}$ matrix elements as they are updated by basis updates or size reduction operations, and $\tilde{R}_{k,k}$ reciprocals as they are computed by the NR-based integer-rounded divider from Section 4.1.1. The SIC/RA module in turn outputs a termination signal to the CLLL processor, indicating when the RA is satisfied for the current symbol vector estimate. Hardware evaluation of the proposed algorithm could then be carried out by running this system with the 802.11n example in Section 5.3.1 for various channel correlation cases.

Rather than build this complete system, we consider an abstraction that involves the CLLL processor and knowledge about when the SIC/RA module would indicate early termination (instead of the actual SIC/RA module). Specifically, we complete the following procedure for a given block of sub-carrier channel matrices in a packet containing one OFDM symbol:

1. Run the proposed algorithm on each channel matrix using the corresponding sub-carrier received vector for the RA-based early termination. Assume that the SNR is the value that results in approximately a 10^{-3} BER. Record the number of iterations required to process each channel matrix.

2. For the channel matrices that require LR processing, assume that these can be processed consecutively. Process this updated block of channel matrices using a cycle-accurate model of the CLLL processor that is set to terminate processing at the corresponding number of iterations determined in step 1 for each channel matrix. Record the total number of cycles.

3. Using the XC5VLX110-3 post-PR timing results in Table 2 and the total cycle

count in step 2, compute the total latency.

We also note here that we can alter step 1 above to accommodate the original CLLL algorithm (Set the maximum number of iterations to the value determined in Section 5.1.2) and hypothetical detector in Section 6.3. Hence, we can gather data about the three CLLL-aided detector types, CLLL, ICLLL, and hypothetical CLLL, using this extended procedure. For a given channel correlation case and CLLL-aided detector type, we run this procedure for a sufficient number of packets such that we can obtain accurate estimates of the average latency, latency where the complementary cumulative distribution function (CCDF) of the latency is equal to 0.005, and the latency where this CCDF is equal to 0.0005.

The results of this experiment for the correlation cases considered in Section 6.2 are contained in Table 4. Figure 22 contains a visualization of these results for the CLLL and ICLLL cases. From this figure, it is apparent that for all three latency metrics, the proposed algorithm results in a large reduction in required processing latency compared to the original CLLL algorithm. For example, when no correlation

Table 4: Required CLLL processor (Section 5.2) latency to process 52 channel matrices in example 802.11n system (Section 5.3.1) when each packet contains a single OFDM symbol.

		$\lvert\rho_{tx}\rvert = \lvert\rho_{rx}\rvert$			
		0.0	0.3	0.5	0.7
CLLL	Average	11.66 µs	16.11 µs	23.95 µs	44.89 µs
	CCDF = 0.005	17.30 µs	22.22 µs	30.67 µs	52.07 µs
	CCDF = 0.0005	19.85 µs	24.85 µs	32.39 µs	55.60 µs
Incremental CLLL	Average	1.82 µs	2.34 µs	3.45 µs	8.43 µs
	CCDF = 0.005	4.60 µs	5.96 µs	7.75 µs	15.83 µs
	CCDF = 0.0005	5.69 µs	7.91 µs	8.63 µs	17.22 µs
Hypothetical CLLL	Average	1.19 µs	1.80 µs	2.71 µs	7.16 µs
	CCDF = 0.005	2.71 µs	4.63 µs	6.74 µs	13.77 µs
	CCDF = 0.0005	4.09 µs	5.76 µs	7.69 µs	15.89 µs

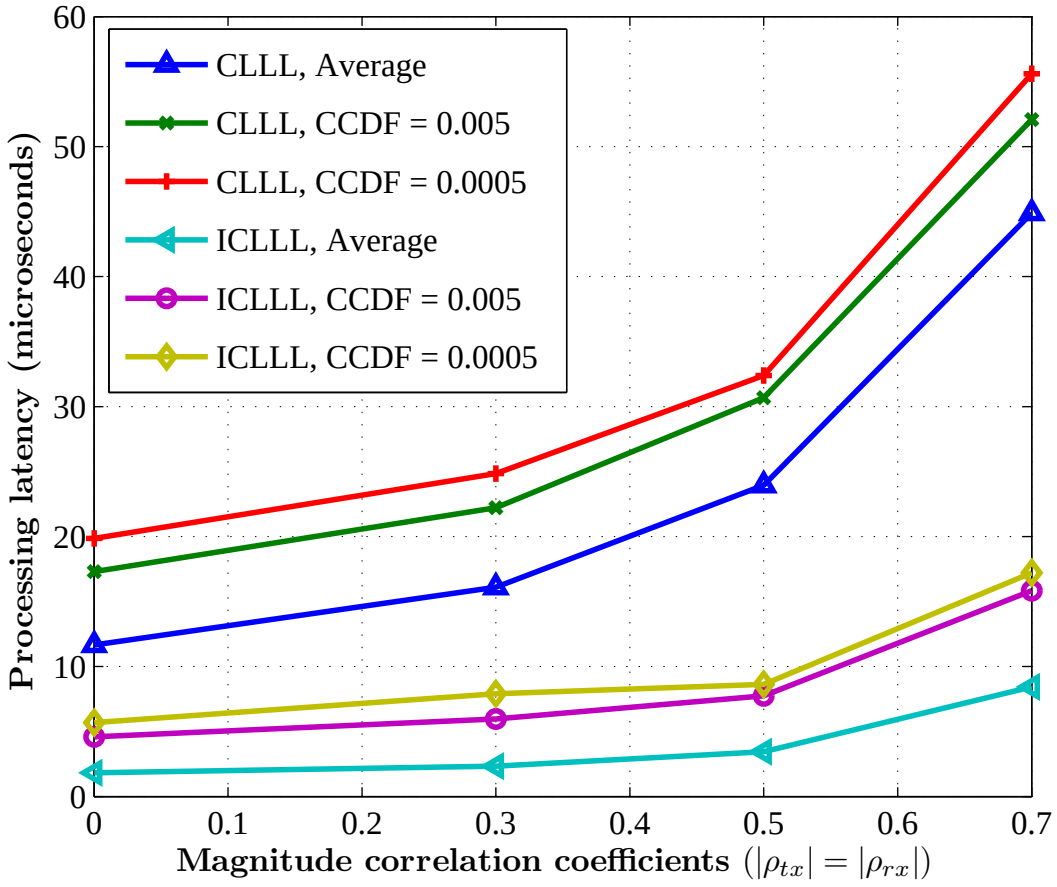

Figure 22: Visualization of experimental results in Table 4.

exists ($|\rho_{tx}| = |\rho_{rx}| = 0.0$) the CCDF = 0.0005 latency is reduced by 72%. For the high correlation case ($|\rho_{tx}| = |\rho_{rx}| = 0.7$), the CCDF = 0.0005 latency is reduced by 69%.

Given that the purpose of developing the proposed algorithm was to realize the early termination potential in Section 6.3, it is instructive to compare the ICLLL and hypothetical CLLL cases. Figure 23 contains a visualization of this comparison. In this figure we examine the ratio of the ICLLL latency to the hypothetical CLLL latency using the data from Table 4. This figure demonstrates that for the high correlation case, the required CCDF = 0.0005 latency of the proposed algorithm is less than 10% greater compared to the CLLL-aided detector with ideal early termination.

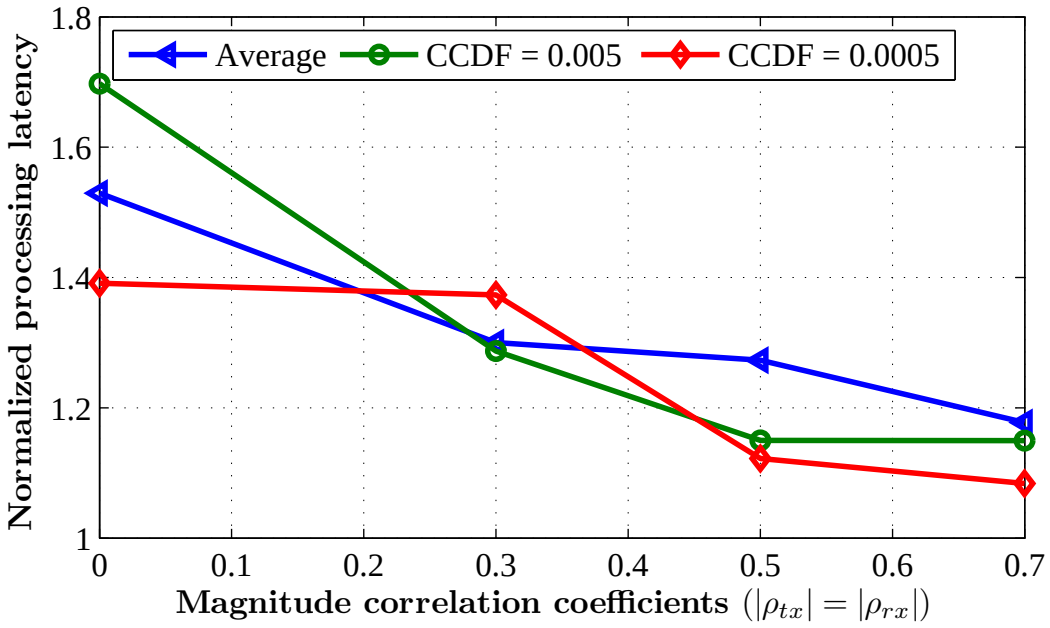

Figure 23: ICLLL data from Table 4 that has been normalized by the corresponding hypothetical CLLL results in this table.

Across all three latency metrics, this figure also demonstrates that as the correlation decreases, the required latency of the proposed algorithm relative to the hypothetical CLLL case increases.

These results are consistent with the chosen $\bar{A}$ parameter in the proposed algorithm. Recall that the $\bar{A}$ parameter sets the complexity to BER performance tradeoff. The choice of $\bar{A} = 18.2$ maintains BER performance for correlation cases $|\rho_{tx}|, |\rho_{rx}| < 0.84$. Clearly, if we restrict the correlation cases to smaller ranges, we can choose larger values for $\bar{A}$ while maintaining BER performance (based on the results in Figure 15). Hence, refining the proposed algorithm to incorporate a variable $\bar{A}$ parameter that is based on an estimate of the channel correlation should lead to the proposed algorithm latencies better matching the latencies of the hypothetical CLLL case.

CHAPTER IX

CONCLUDING REMARKS

Throughout the dissertation we have alternately examined LR-aided detection from algorithm and hardware perspectives, resulting in a unique combination of hardware and algorithm contributions. Through an understanding of the channel model employed and operation of the CLLL algorithm, we proposed, analytically and empirically justified, and implemented modifications to CLLL-aided detection algorithms. These modifications led to simplifications in the normally hardware-complex operations of the CLLL algorithm and streamlined the overall architecture development. After implementing the modified CLLL algorithm, we were then able to consider the hardware implications of a more realistic wireless channel model. Specifically, we examined the effect of spatial correlation at the receiver and transmitter on the processing latency of the proposed architecture. The large latency increase that we observed then motivated us to solve this problem of increasing latency with increasing spatial correlation. The solution to this problem involved a combination of a practical early termination condition and an intelligent merging of the LR processing and symbol detection computations, which we introduced as ILR. Finally, using a co-simulation environment, we demonstrated that the average packet processing latency can be drastically reduced when the proposed ILR scheme is coupled with the proposed architecture.

9.1 Contributions

This dual-view hardware/algorithm approach has resulted in the following specific contributions in a variety of publications:

- Elementary proof that the CLLL-MMSE-SIC symbol vector estimate is unaffected by size reduction operations until a basis update occurs during execution of the CLLL algorithm. This result naturally enables early termination of the CLLL algorithm based on the Siegel condition [21].

- Modified CLLL algorithm with relaxed size reduction condition and corresponding derivation of upper bound analysis for fixed-point implementation. This work was initially completed based on the Lovász condition [20] and then subsequently refined for the Siegel condition in [21].

- Relative error analysis and development of single Newton-Raphson iteration architecture for executing size reduction operations [16].

- Development of unrolled and pipelined Householder CORDIC architecture that enables simultaneous vectoring and rotation operations, reduced multiplexer complexity, and decreased critical path delay [20].

- Architecture and FPGA hardware realization of CLLL processor for a 4×4 CLLL-MMSE-SIC detector. This work resulted in both the first conference publication [19] and journal publications [82, 21] in the area of CLLL hardware realization.

- Analytic justification that RA-based hybrid detectors can achieve full diversity [18].

- Development of incremental CLLL-MMSE-SIC detection and application to reducing the packet processing latency of a system that uses the proposed CLLL architecture [17, 18].

- Development of a co-simulation environment for rapid hardware development and verification [15].

9.2 Suggestions for Future Research

Next generation wireless communication systems will likely exhibit extremely high spectral efficiency, require detection algorithms that exhibit improved BER performance, and require even higher packet-rate and symbol-rate processing throughput. The following extensions of the dissertation research begin to address these challenges:

- Given the general description (in terms of MIMO system size) of both the modifications to the CLLL algorithm and the CLLL processor design parameters and procedures, the analytic and empirical results in the dissertation can be applied to more complex MIMO antenna configurations, e.g., 8×8 systems.

- The proposed CLLL architecture for 4×4 systems can be used to develop a hardware realization of the CLLL-aided soft-output decoder in [81].

- The idea of incremental LR introduced in the dissertation can be applied to the most recent LR-aided detection research in [41, 38, 39]. Specifically, we note the following for a given LR-aided detection algorithm:

    1. The hypothetical experiment in Section 6.3 can be completed for the given LR-aided detection algorithm to determine if the potential for early termination exists.

    2. The proof in Appendix D is applicable to any LR-aided detection algorithm because any detection algorithm can be used for the initial estimate of the transmitted symbol vector.

    3. The correlation case determination exercise and $\bar{A}$ determination process in Section 8.1 form a straightforward methodology for determining an $\bar{A}$ given realistic SNR constraints.

    4. The final ILR algorithm based on the given LR-aided detection algorithm can evaluated using both the single symbol vector per packet and multiple

symbols per packet experiments in Section 8.2 and Section 8.3, respectively.

- Although we demonstrate the utility of ILR using a fixed $\bar{A}$ parameter, the $\bar{A}$ parameter can alternatively be determined based on the channel estimation scheme, channel matrix, estimated noise variance, and packet size. Further reduction in complexity and relaxation of hardware requirements should be possible. The hypothetical detector in Section 6.3 can be used to evaluate the effectiveness of any method for determining the $\bar{A}$ parameter.

- To create a low-power and low-latency ASIC implementation, the datapath modules of the proposed CLLL architecture can be replaced with asynchronous logic datapaths (employing the embedding technique proposed in [46]). Additionally, asynchronous logic implementations of the partial SIC and RA computation required for ICLLL-MMSE-SIC can be explored.

APPENDIX A

PROOF OF PROPOSITION 1: EFFECT OF SIZE REDUCTION ON CLLL-MMSE-SIC DETECTION

In this appendix we demonstrate that the symbol vector estimate obtained from the CLLL-MMSE-SIC detection procedure is unaffected by size reduction of the entire matrix. We first define a matrix $\boldsymbol{A}^{(i)}$ to be a unimodular matrix that has a one on each diagonal, arbitrary complex integers in the upper-off-diagonal elements of the i-th column, and zero on the remaining matrix entries. We can then consider the following lemma under the assumption that $\mathcal{Q}_{\mathcal{S}}$ in (9) is the element-wise integer-rounding operation:

Lemma 5 *Let $\hat{\boldsymbol{x}}$ be the SIC solution when an arbitrary $N_t \times N_t$ invertible upper-triangular matrix $\boldsymbol{R}$ with complex elements and an $N_t \times 1$ complex vector $\boldsymbol{b}$ are used in the (9) SIC recursion. The elements of the SIC solution $\boldsymbol{x}'$ when $\boldsymbol{R}' = \boldsymbol{R}\boldsymbol{A}^{(i)}$ and $\boldsymbol{b}$ are used in (9) are then equal to*

$$x'_n = \begin{cases} x_n & i \leq n \leq N_t, \\ x_n - a^{(i)}_{n,i} x_i & 1 \leq n \leq i-1. \end{cases} \tag{69}$$

Proof: The i-th column of $\boldsymbol{R}'$ is equal to the following:

$$\boldsymbol{R}'_i = \boldsymbol{R}_i + \sum_{j=1}^{i-1} a^{(i)}_{j,i} \boldsymbol{R}_j. \tag{70}$$

Given that only the i-th column elements of $\boldsymbol{R}'$ in rows of index less than i are affected, clearly $x'_n = x_n$ for $i \leq n \leq N_t$. We use induction to prove the $1 \leq n \leq i-1$ case of (69).

Beginning with the base case $n = i - 1$ we have the following:

$$x'_{i-1} = \left\lfloor \frac{b_{i-1} - \sum_{j=i}^{N_t} R'_{i-1,j} x'_j}{R'_{i-1,i-1}} \right\rceil = \left\lfloor \frac{b_{i-1} - \sum_{j=i+1}^{N_t} R_{i-1,j} x_j - R'_{i-1,i} x_i}{R_{i-1,i-1}} \right\rceil. \tag{71}$$

Using (70) we can apply the substitution $R'_{i-1,i} = R_{i-1,i} + a^{(i)}_{i-1,i} R_{i-1,i-1}$ to (71) and simplify:

$$\left\lfloor \frac{b_{i-1} - \sum_{j=i}^{N_t} R_{i-1,j} x_j}{R_{i-1,i-1}} \right\rceil - a^{(i)}_{i-1,i} x_i = x_{i-1} - a^{(i)}_{i-1,i} x_i.$$

Next, we assume that (69) is true for $l \leq n \leq N_t$ and show that (69) is true for $l - 1 \leq n \leq N_t$, where $2 \leq l \leq i - 2$. It therefore suffices to show that (69) is true for $n = l - 1$. We begin with the SIC solution for the $l - 1$ element of $\boldsymbol{x}'$:

$$x'_{l-1} = \left\lfloor \frac{b_{l-1} - \sum_{j=l}^{N_t} R'_{l-1,j} x'_j}{R'_{l-1,l-1}} \right\rceil. \tag{72}$$

Now letting m_{l-1} be the numerator part of the fraction in (72) such that $x'_{l-1} = \lfloor m_{l-1} / R'_{l-1,l-1} \rceil$, we can write

$$m_{l-1} = b_{l-1} - \sum_{j=l}^{i-1} R_{l-1,j} x'_j - R'_{l-1,i} x_i - \sum_{j=i+1}^{N_t} R_{l-1,j} x_j.$$

Applying (70) and the induction hypothesis, we obtain

$$m_{l-1} = b_{l-1} - \sum_{j=l}^{i-1} R_{l-1,j} \left(x_j - x_i a^{(i)}_{j,i} \right) - \sum_{j=i+1}^{N_t} R_{l-1,j} x_j - \left(R_{l-1,i} + \sum_{j=l-1}^{i-1} a^{(i)}_{j,i} R_{l-1,j} \right) x_i. \tag{73}$$

Simplifying (73) and substituting back into (72), we obtain the following:

$$x'_{l-1} = \left\lfloor \frac{b_{l-1} - \sum_{j=l}^{N_t} R_{l-1,j} x_j - a^{(i)}_{l-1,i} x_i R_{l-1,l-1}}{R_{l-1,l-1}} \right\rceil = x_{l-1} - a^{(i)}_{l-1,i} x_i. \qquad \blacksquare$$

We now use this result to prove the final lemma. Let $\tilde{\boldsymbol{Q}} \tilde{\boldsymbol{R}} = \bar{\boldsymbol{H}} \boldsymbol{T}$ be the $\bar{\boldsymbol{H}}$ (defined in (6)) factorization produced by the CLLL algorithm. Also let $\hat{\boldsymbol{z}}$ be the SIC solution to equation (16)

$$\tilde{\boldsymbol{R}} \boldsymbol{z} = \frac{1}{2} \tilde{\boldsymbol{Q}}^{\mathcal{H}} \left(\bar{\boldsymbol{y}} + \bar{\boldsymbol{H}} \left(1 + j \right) \mathbf{1}_{N_t \times 1} \right), \tag{74}$$

where we write the righthand side in an equivalent form. The symbol vector estimate (before scaling, shifting, and quantization to the nearest symbol constellation) from this z-domain solution is then $\dot{\boldsymbol{s}} = \boldsymbol{T}\hat{\boldsymbol{z}}$. Also, let $\tilde{\boldsymbol{R}}^{(i)} = \tilde{\boldsymbol{R}}^{(i-1)}\boldsymbol{B}^{(i)}$ and $\boldsymbol{T}^{(i)} = \boldsymbol{T}^{(i-1)}\boldsymbol{B}^{(i)}$ with $\tilde{\boldsymbol{R}}^{(1)} = \tilde{\boldsymbol{R}}$ and $\boldsymbol{T}^{(1)} = \boldsymbol{T}$. Let $\boldsymbol{B}^{(i)}$ be generated by running the procedure in Table 5 with $\dot{\boldsymbol{R}} = \tilde{\boldsymbol{R}}^{(i-1)}$ initially. We now consider the following lemma:

Lemma 6 *If $\tilde{\boldsymbol{R}}$ is size-reduced to produce a new upper-triangular matrix $\tilde{\boldsymbol{R}}' = \boldsymbol{R}\left(\boldsymbol{B}^{(2)}\cdots\boldsymbol{B}^{(N_t)}\right)$ and updated unimodular matrix $\boldsymbol{T}' = \boldsymbol{T}\left(\boldsymbol{B}^{(2)}\cdots\boldsymbol{B}^{(N_t)}\right)$, then the updated CLLL-MMSE-SIC symbol vector estimate $\dot{\boldsymbol{s}}'$ (before scaling, shifting, and quantization to the nearest symbol constellation) is unchanged, i.e. $\dot{\boldsymbol{s}}' = \dot{\boldsymbol{s}}$.*

Proof: Let $\hat{\boldsymbol{z}}^{(i)}$ be the SIC solution to (74) that has $\tilde{\boldsymbol{R}}$ replaced with $\tilde{\boldsymbol{R}}^{(i)}$ and $\dot{\boldsymbol{s}}^{(i)}$ be the subsequent symbol vector estimate. Since $\dot{\boldsymbol{s}}' = \dot{\boldsymbol{s}}^{(N_t)}$, we can use induction on i to prove this lemma. Beginning with the base case $i = 2$, we use Lemma 5 to show the following:

$$\dot{\boldsymbol{s}}^{(2)} = \boldsymbol{T}^{(2)}\hat{\boldsymbol{z}}^{(2)} = \begin{bmatrix} | & | & | & | \\ \boldsymbol{T}_1 & \boldsymbol{T}_2 - u_{1,2}\boldsymbol{T}_1 & \boldsymbol{T}_3 & \cdots & \boldsymbol{T}_{N_t} \\ | & | & | & | \end{bmatrix} \begin{bmatrix} \hat{z}_1 + u_{1,2}\hat{z}_2 \\ \hat{z}_2 \\ \vdots \\ \hat{z}_{N_t} \end{bmatrix}$$

$$= \sum_{j=1}^{N_t} \boldsymbol{T}_j\hat{z}_j + \boldsymbol{T}_1 u_{1,2}\hat{z}_2 - \boldsymbol{T}_1 u_{1,2}\hat{z}_2 = \boldsymbol{T}\hat{\boldsymbol{z}} = \dot{\boldsymbol{s}}.$$

Table 5: Generation of $\boldsymbol{B}^{(i)}$ matrices for Full Size Reduction on the i-th column of $\dot{\boldsymbol{R}}$

(1) $\boldsymbol{B}^{(i)} = \boldsymbol{I}_{N_t}$;
(2) for $n = i-1 : -1 : 1$
(3) $\quad u_{n,i} = \left\lfloor \dot{R}_{n,i} / \dot{R}_{n,n} \right\rceil$;
(4) $\quad \dot{\boldsymbol{R}}_i = \dot{\boldsymbol{R}}_i - u_{n,i} \cdot \dot{\boldsymbol{R}}_n; \qquad \boldsymbol{B}_i^{(i)} = \boldsymbol{B}_i^{(i)} - u_{n,i} \cdot \boldsymbol{B}_n^{(i)};$
(5) end

Next, we assume $\dot{\boldsymbol{s}}^{(i)} = \dot{\boldsymbol{s}}$ and show that this implies $\dot{\boldsymbol{s}}^{(i+1)} = \dot{\boldsymbol{s}}$. Using Lemma 5 we obtain

$$\hat{\boldsymbol{z}}^{(i+1)} = \begin{bmatrix} \hat{z}_1^{(i)} + u_{1,i+1}\hat{z}_{i+1}^{(i)} \\ \vdots \\ \hat{z}_i^{(i)} + u_{i,i+1}\hat{z}_{i+1}^{(i)} \\ \hat{z}_{i+1}^{(i)} \\ \vdots \\ \hat{z}_{N_t}^{(i)} \end{bmatrix}.$$

We also have that

$$\boldsymbol{T}^{(i+1)} = \begin{bmatrix} | & & | & | & | & & | \\ \boldsymbol{T}_1^{(i)} & \cdots & \boldsymbol{T}_i^{(i)} & \boldsymbol{T}_{i+1}^{(i+1)} & \boldsymbol{T}_{i+2}^{(i)} & \cdots & \boldsymbol{T}_{N_t}^{(i)} \\ | & & | & | & | & & | \end{bmatrix},$$

where $\boldsymbol{T}_{i+1}^{(i+1)} = \boldsymbol{T}_{i+1}^{(i)} - \sum_{j=1}^{i} u_{j,i+1}\boldsymbol{T}_j^{(i)}$. We then have

$$\begin{aligned} \dot{\boldsymbol{s}}^{(i+1)} &= \boldsymbol{T}^{(i+1)}\hat{\boldsymbol{z}}^{(i+1)} \\ &= \sum_{j=1}^{N_t} \boldsymbol{T}_j^{(i)}\hat{z}_j^{(i)} + \sum_{j=1}^{i} \boldsymbol{T}_j^{(i)} \left(\hat{z}_{i+1}^{(i)} u_{j,i+1} - \hat{z}_{i+1}^{(i)} u_{j,i+1} \right) \\ &= \boldsymbol{T}^{(i)}\hat{\boldsymbol{z}}^{(i)} = \dot{\boldsymbol{s}}^{(i)} = \dot{\boldsymbol{s}}. \end{aligned}$$

■

APPENDIX B

SOLVING CONSTRAINED MAXIMUM PROBLEM FOR UPPER BOUND ON INTERMEDIATE SIZE REDUCTION RESULTS

In this appendix, we solve the constrained maximum problem that arises in Chapter 3, which is the maximization of (38) subject to the constraint in (39) when $k > 2$. Towards this goal, we first define a $2(k-n) \times 1$ column vector

$$\boldsymbol{x} = \left[\begin{array}{cccccccc} \left|\Re[\tilde{R}_{n,k}]\right| & \cdots & \left|\Re[\tilde{R}_{k-1,k}]\right| & \left|\Im[\tilde{R}_{n,k}]\right| & \cdots & \left|\Im[\tilde{R}_{k-1,k}]\right| \end{array} \right]^T, \tag{75}$$

a $2(k-n) \times 1$ column vector

$$\boldsymbol{c} = \left[\begin{array}{ccccccccc} 1 & \alpha_{n+1,k} & \cdots & \alpha_{k-1,k} & 0 & \alpha_{n+1,k} & \cdots & \alpha_{k-1,k} \end{array} \right]^T, \tag{76}$$

and a constant $E_{\max} = B^2\left(1 + \frac{1}{2}(k-2)\right)$. Next, using these definitions, we write the righthand side of (38) as

$$\begin{aligned} &\gamma_{n,k}B + \left|\Re[\tilde{R}_{n,k}]\right| + \sum_{p=n+1}^{k-1} \alpha_{p,k}\left(\left|\Re[\tilde{R}_{p,k}]\right| + \left|\Im[\tilde{R}_{p,k}]\right|\right) \\ &\quad = \gamma_{n,k}B + c_1x_1 + \sum_{p=2}^{k-n} c_px_p + c_{k-n+1}x_{k-n+1} + \sum_{p=k-n+2}^{2(k-n)} c_px_p \\ &\quad = \gamma_{n,k}B + \boldsymbol{c}^T\boldsymbol{x}. \end{aligned} \tag{77}$$

Defining $f(\boldsymbol{x}) = \left(\boldsymbol{c}^T\boldsymbol{x}\right)^2$, we see that the constrained maximum problem in Chapter 3 can be solved by finding the $\boldsymbol{x}$ that maximizes $f(\boldsymbol{x})$ subject to the constraint $\|\boldsymbol{x}\|^2 = E_{\max}$. Letting λ be a Lagrange multiplier, we can examine partial derivatives of $f(\boldsymbol{x}) = \left(\boldsymbol{c}^T\boldsymbol{x}\right)^2 + \lambda\left(E_{\max} - \|\boldsymbol{x}\|^2\right)$ with respect to x_i:

$$\frac{\partial f}{\partial x_i} = 2c_i\boldsymbol{c}^T\boldsymbol{x} - 2\lambda x_i = 0 \tag{78}$$

Assuming that $\boldsymbol{c}^T\boldsymbol{x} \neq 0$, we see that the solution to this constrained maximization problem satisfies $c_i x_j = c_j x_i$ for any x_i and x_j. We then use this result in the original constraint equation, obtaining

$$E_{\max} = \sum_{i=1}^{2(k-n)} x_i^2$$
$$E_{\max} c_j^2 = \sum_{i=1}^{2(k-n)} (c_i x_j)^2 = x_j^2 \|\boldsymbol{c}\|^2.$$

Hence, we see that the solution to this constrained maximization problem is $x_j = \sqrt{E_{\max}} c_j / \|\boldsymbol{c}\|$. Substitution of this solution into (77) results in

$$\begin{aligned}\gamma_{n,k} B + \sqrt{E_{\max}}\|\boldsymbol{c}\| &= \gamma_{n,k} B + B\sqrt{1 + \frac{1}{2}(k-2)}\sqrt{1 + 2\sum_{p=n+1}^{k-1} \alpha_{p,k}^2} \\ &= B\left(\gamma_{n,k} + \sqrt{k\left(\frac{1}{2} + \sum_{p=n+1}^{k-1} \alpha_{p,k}^2\right)}\right).\end{aligned}$$

APPENDIX C

APPLICATION OF NEWTON-RAPHSON RELATIVE ERROR ANALYSIS IN EXAMPLE SYSTEM

To demonstrate the utility of the $M_{\gamma,a}$ upper-bound derived in Section 4, we use this upper-bound for an integer-rounded divider having parameters of $w = 12$ and $f = 13$, which mirror parameters in the work in [19].

Using the analysis in Chapter 4, we find that a 32-entry $(r_n + e)$ LUT of width 6-bits results in $M_{5,6} < 1/1024$. To demonstrate that $M_{5,6}$ is indeed an upper-bound for the relative error, we employ random simulations. Generating test cases by sampling n and d from a uniform distribution results in a quotient distribution that is highly skewed toward lower values. Instead we first sample a "quotient" q from a uniform distribution over $(0, U_{max})$, and then generate two cases from this quotient. For the first test case we sample a d from a uniform distribution over $(0, \min(2^w/q, 2^w))$ and then generate $n = qd$. For the second test case we sample an n from a uniform distribution over $(0, \min(2^w q, 2^w))$ and then generate $d = n/q$.

Figure 24 shows the maximum relative error, $(q' - q)/q$, of 20 million test cases for each set of test cases that resulted in a particular rounded quotient, where $q' = n2^{\alpha} r'_n$. The figure demonstrates that the maximum relative error is strictly less than $M_{5,6}$ and that $M_{5,6}$ is within 2% of the overall maximum relative error.

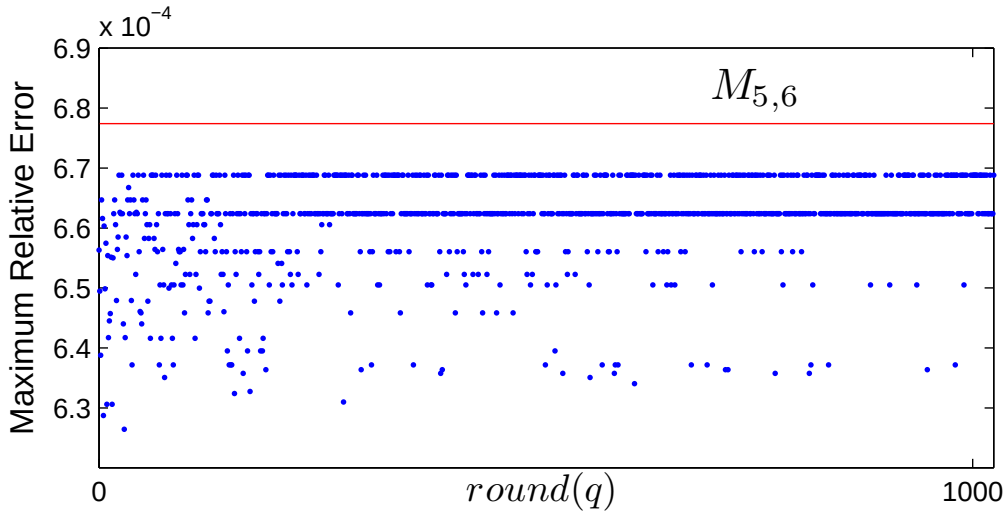

Figure 24: Maximum relative error for each set of test cases that resulted in a particular rounded q quotient. The $M_{5,6}$ bound for a 32-entry, 6-bit wide reciprocal LUT is less than the 1/1024 constraint and strictly greater than the overall maximum relative error.

APPENDIX D

PROOF OF PROPOSITION 2

We first define a random vector $\boldsymbol{v} = \boldsymbol{H}(\boldsymbol{s} - \hat{\boldsymbol{s}})$ and a random vector $\boldsymbol{m}$ to be $\boldsymbol{v}$ given that $\boldsymbol{s} \to \hat{\boldsymbol{s}}$ has occurred. We let $\boldsymbol{m}^{(I)} = \Re[\boldsymbol{m}]$ and $\boldsymbol{m}^{(Q)} = \Im[\boldsymbol{m}]$. We then first notice the following upper bound:

$$P\{\|\boldsymbol{d}\| \leq A\sigma_w \mid \boldsymbol{m}\} \leq \prod_{i=1}^{N_r} \left(P\left\{\left|m_i^{(I)} + \eta\Re[w_i]\right| \leq A\sigma_w\right\} \cdot P\left\{\left|m_i^{(Q)} + \eta\Im[w_i]\right| \leq A\sigma_w\right\}\right). \tag{79}$$

For convenience, we define $\boldsymbol{n} = \frac{\sqrt{2}}{\sigma_w}\boldsymbol{w}$, let $\sigma = \eta\sigma_w$, and let $\tilde{A} = A/\eta$. Then we focus on establishing an upper bound for one of the product terms $P\left\{\left|m_i^{(I)} + \eta\Re[w_i]\right| \leq A\sigma_w\right\}$, which we can write as $P\left\{\left|\frac{\sqrt{2}}{\sigma}m_i^{(I)} + \Re[n_i]\right| \leq \sqrt{2}\tilde{A}\right\}$.

For the case $\left|m_i^{(I)}\right| > \tilde{A}\sigma$ this probability is the area of the shaded region in Figure 25, which we upper bound using the following:

$$\begin{aligned} P\left\{\left|\frac{\sqrt{2}}{\sigma}m_i^{(I)} + \Re[n_i]\right| \leq \sqrt{2}\tilde{A}\right\} &= \frac{1}{\sqrt{2\pi}} \int_{-\sqrt{2}\tilde{A} - \frac{\sqrt{2}}{\sigma}\left|m_i^{(I)}\right|}^{\sqrt{2}\tilde{A} - \frac{\sqrt{2}}{\sigma}\left|m_i^{(I)}\right|} e^{-\frac{1}{2}x^2} dx \\ &\leq Q\left(\frac{\sqrt{2}}{\sigma}\left|m_i^{(I)}\right| - \sqrt{2}\tilde{A}\right) \\ &\leq e^{-\frac{1}{2}\left(\frac{\sqrt{2}}{\sigma}\left|m_i^{(I)}\right| - \sqrt{2}\tilde{A}\right)^2} \leq e^{-\frac{1}{2}\left(\frac{2}{\sigma^2}\left|m_i^{(I)}\right|^2 - \frac{4}{\sigma}\tilde{A}\left|m_i^{(I)}\right|\right)}. \end{aligned} \tag{80}$$

For the case $\tilde{A} < \frac{3}{8\sigma}\left|m_i^{(I)}\right|$, we can apply (80) to establish the upper bound

$$P\left\{\left|\frac{\sqrt{2}}{\sigma}m_i^{(I)} + \Re[n_i]\right| \leq \sqrt{2}\tilde{A}\right\} \leq e^{-\frac{\left(m_i^{(I)}\right)^2}{4\sigma^2}}. \tag{81}$$

We then use this result in the definition of the following function:

$$P\left(x, \tilde{A}, \sigma\right) = \begin{cases} e^{-\frac{x^2}{4\sigma^2}} & |x| > \frac{8}{3}\tilde{A}\sigma, \\ 1 & \text{otherwise.} \end{cases} \tag{82}$$

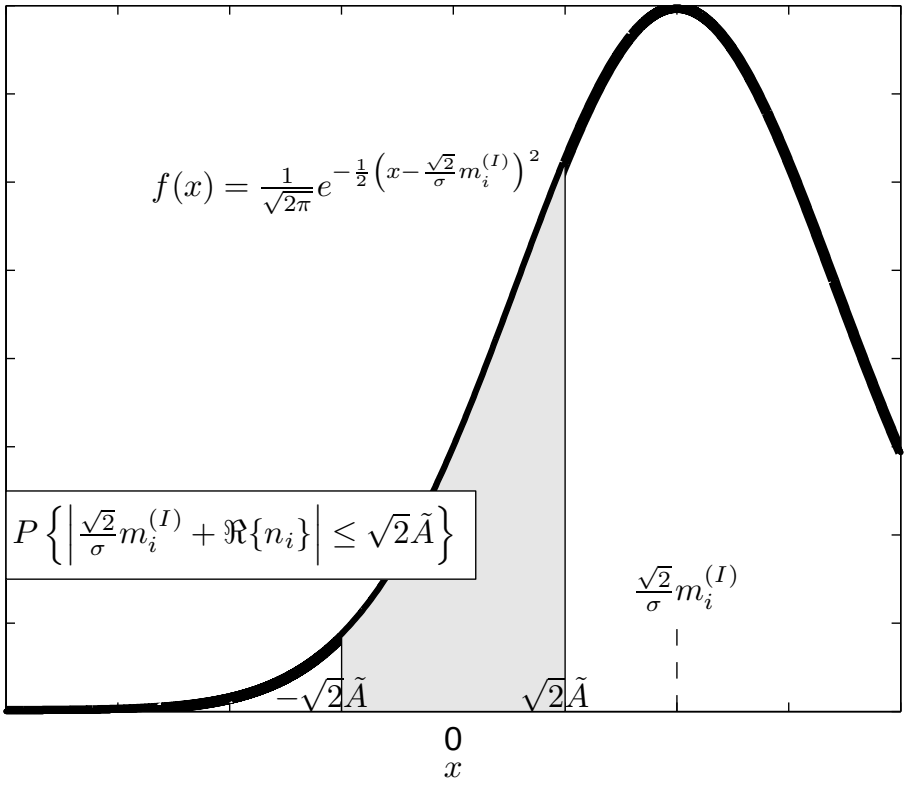

Figure 25: Visualization of $P\left\{\left|\frac{\sqrt{2}}{\sigma} m_i^{(I)} + \Re[n_i]\right| \leq \sqrt{2}\tilde{A}\right\}$ when $\left|m_i^{(I)}\right| > \tilde{A}\sigma$. Specifically, the $m_i^{(I)} > 0$ case is shown. As a result of symmetry, however, the illustration is also useful for visualizing the $m_i^{(I)} < 0$ case.

Clearly, we have

$$P\left\{\left|\frac{\sqrt{2}}{\sigma} m_i^{(I)} + \Re[n_i]\right| \leq \sqrt{2}\tilde{A}\right\} \leq P\left(m_i^{(I)}, \tilde{A}, \sigma\right), \tag{83}$$

$$P\left\{\left|\frac{\sqrt{2}}{\sigma} m_i^{(Q)} + \Im[n_i]\right| \leq \sqrt{2}\tilde{A}\right\} \leq P\left(m_i^{(Q)}, \tilde{A}, \sigma\right). \tag{84}$$

We now use these results to establish an upper bound for the expression in (85) by first finding an upper bound for the pdf of $\boldsymbol{m}$, $f(\boldsymbol{m})$. Given the definition of $\boldsymbol{H}'$, it follows that $\boldsymbol{H}'\boldsymbol{\Sigma}_{tx}^{\frac{1}{2}}(\boldsymbol{s} - \hat{\boldsymbol{s}}) \backsim \mathcal{CN}\left(\boldsymbol{0}, \left\|\boldsymbol{\Sigma}_{tx}^{\frac{1}{2}}(\boldsymbol{s} - \hat{\boldsymbol{s}})\right\|^2 \boldsymbol{I}\right)$. It then follows that $\boldsymbol{m} \backsim \mathcal{CN}\left(\boldsymbol{0}, \left\|\boldsymbol{\Sigma}_{tx}^{\frac{1}{2}}(\boldsymbol{s} - \hat{\boldsymbol{s}})\right\|^2 \boldsymbol{\Sigma}_{rx}\right)$. Using (65) we obtain $f(\boldsymbol{m}) \leq B_e$, which we use

to establish the upper bound in (86):

$$P\{\|\boldsymbol{d}\| \leq \tilde{A}\sigma_w \mid \boldsymbol{s} \to \hat{\boldsymbol{s}}\} = \int_{\boldsymbol{m}^{(I)} \in \mathbb{R}^{N_r}} \int_{\boldsymbol{m}^{(Q)} \in \mathbb{R}^{N_r}} P\{\|\boldsymbol{d}\| \leq \tilde{A}\sigma_w \mid \boldsymbol{m}\} f(\boldsymbol{m})\, d\boldsymbol{m}^{(Q)} d\boldsymbol{m}^{(I)} \tag{85}$$

$$\leq B_e \int_{\boldsymbol{m}^{(I)} \in \mathbb{R}^{N_r}} \int_{\boldsymbol{m}^{(Q)} \in \mathbb{R}^{N_r}} P\{\|\boldsymbol{d}\| \leq \tilde{A}\sigma_w \mid \boldsymbol{m}\} d\boldsymbol{m}^{(Q)} d\boldsymbol{m}^{(I)} \tag{86}$$

Next, we define the following subsets of $\mathbb{R}^{N_r}$:

$$\mathcal{T}_{0,i} = \left\{ \boldsymbol{g} \in \mathbb{R}^{N_r} \middle| |g_i| \leq \frac{8}{3}\tilde{A}\sigma \right\}, \quad \mathcal{T}_{1,i} = \left\{ \boldsymbol{g} \in \mathbb{R}^{N_r} \middle| |g_i| > \frac{8}{3}\tilde{A}\sigma \right\}. \tag{87}$$

Then we can then split (86) into 2^{2N_r} parts:

$$\sum_{i_1=0}^{1} \cdots \sum_{i_{N_r}=0}^{1} \sum_{k_1=0}^{1} \cdots \sum_{k_{N_r}=0}^{1} \mathcal{L}_{i_1,\cdots,i_{N_r},k_1,\cdots,k_{N_r}}, \tag{88}$$

where $\mathcal{L}_{i_1,\cdots,i_{N_r},k_1,\cdots,k_{N_r}}$ is defined as

$$\mathcal{L}_{i_1,\cdots,i_{N_r},k_1,\cdots,k_{N_r}} = B_e \int_{\boldsymbol{m}^I \in \cap_{l=1}^{N_r} \mathcal{T}_{i_l,l}} \int_{\boldsymbol{m}^Q \in \cap_{l=1}^{N_r} \mathcal{T}_{k_l,l}} P\{\|\boldsymbol{d}\| \leq \tilde{A}\sigma_w \mid \boldsymbol{m}\} d\boldsymbol{m}^{(Q)} d\boldsymbol{m}^{(I)}. \tag{89}$$

Before establishing an upper bound for (89), we also define the following subsets of $\mathbb{R}$:

$$\mathcal{U}_0 = \left\{ x \in \mathbb{R} \middle| |x| \leq \frac{8}{3}\tilde{A}\sigma \right\}, \quad \mathcal{U}_1 = \left\{ x \in \mathbb{R} \middle| |x| > \frac{8}{3}\tilde{A}\sigma \right\}. \tag{90}$$

We can then apply the upper bounds in (79) to obtain the following upper bound for (89):

$$B_e \left(\prod_{l=1}^{N_r} \int_{\mathcal{U}_{i_l}} P\left(x, \tilde{A}, \sigma\right) dx \right) \left(\prod_{l=1}^{N_r} \int_{\mathcal{U}_{k_l}} P\left(x, \tilde{A}, \sigma\right) dx \right). \tag{91}$$

Then we consider integrating $P\left(x, \tilde{A}, \sigma\right)$ over $\mathcal{U}_0$:

$$\int_{\mathcal{U}_0} P\left(x, \tilde{A}, \sigma\right) dx = \int_{-\frac{8}{3}\tilde{A}\sigma}^{\frac{8}{3}\tilde{A}\sigma} dx = \frac{16}{3}\tilde{A}\sigma \tag{92}$$

and $\mathcal{U}_1$:

$$\int_{\mathcal{U}_1} P\left(x, \tilde{A}, \sigma\right) dx = 2 \int_{\frac{8}{3}\tilde{A}\sigma}^{\infty} e^{-\frac{x^2}{4\sigma^2}} dx = 4\sqrt{\pi} Q\left(\frac{8\tilde{A}}{3\sqrt{2}}\right) \sigma. \tag{93}$$

Using the definition of M_A in (66) we can establish an upper bound for (89) as follows:

$$\mathcal{L}_{i_1,\cdots,i_{N_r},k_1,\cdots,k_{N_r}} \leq B_e M_A^{2N_r} \sigma^{2N_r}. \tag{94}$$

Finally, returning to (86), we obtain the following upper bound:

$$P\{\|\boldsymbol{d}\| \leq A\sigma_w \mid \boldsymbol{s} \to \hat{\boldsymbol{s}}\} \leq (2\eta M_A)^{2N_r} B_e \left(\frac{1}{\sigma_w^2}\right)^{-N_r}. \qquad \blacksquare$$

APPENDIX E

MODELSIM-MATLAB INTERFACE FOR RTL DEBUGGING AND VERIFICATION

ModelSim [51] is an optimized simulation engine for SystemC, Verilog, and VHDL descriptions of hardware systems. It contains powerful waveform viewing and simulation data exporting features that are essential for debugging and verifying digital hardware design but lacks built-in signal processing functions. Matlab, conversely, contains a multitude of signal processing functions. Therefore, a cosimulation environment that incorporates these tools is useful for both hardware development and verification. In this appendix we describe a System Verilog/C code creation and compilation system that creates a ModelSim-Matlab shared memory interface optimized for the input/output specification of the user Verilog or VHDL. This tool enables the user to indirectly control a ModelSim simulation from a Matlab test script.

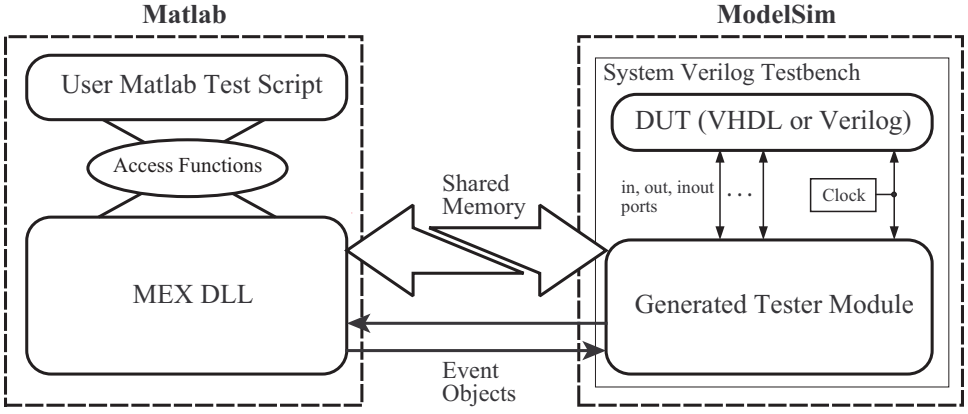

Figure 26: High-level block diagram of the proposed ModelSim-Matlab interface

E.1 Interface Features Description

The ModelSim-Matlab interface for cycle-accurate functional simulations of Verilog and VHDL code provides a simple set of access functions such that the communication and simulation control details are hidden from the user. Figure 26 shows a block diagram of the ModelSim-Matlab interface. The clock Verilog module in the System Verilog Testbench generates a DUT clock and a slightly leading and lagging clock from the DUT clock. The generated tester module drives the DUT inputs at the positive edge of the leading clock and reads the DUT outputs at the positive edge of the lagging clock. The tester module communicates with Matlab using shared memory (similar to Link for ModelSim) for data passing and a simple two-phase handshake implemented with Microsoft Event Objects for process synchronization. On the Matlab side, a mex file forms the communication entry point, with the Matlab Test Script calling this mex file to drive and to control the ModelSim simulation.

The above interface details, however, are not visible to the user. Instead the following simple set of top-level access functions handle these operations transparently:

1. open_ModelSim_connection - Opens a shared-memory connection with an open ModelSim process and returns a *DUT struct* (Matlab struct that has a field for each input, output, and bidirectional port of the DUT).

2. sendReceive_ModelSim_connection - Main interface function. The user sets the DUT (input) struct fields to a decimal number, hexadecimal number, or signal string (containing any combination of 'Z', 'X', '1', or '0' values). The function takes this DUT struct as an input, drives the inputs and bidirectional ports of the DUT using the values specified in this struct, advances the ModelSim simulation by a single clock, and finally returns a struct containing the updated DUT port states.

3. get_ModelSim_io - Takes as input a DUT struct and DUT input/output name string and returns the current value of the DUT input/output as a signal string, decimal number, or hexadecimal number. There is no apparent advancement

of the simulation time to the user.

4. pause_ModelSim_simulation - Returns control to ModelSim such that the user can use the waveform viewer and other features of ModelSim in the middle of a simulation. This is equivalent to a *$stop* Verilog command.

5. close_ModelSim_simulation - Halts a ModelSim simulation, shuts down the shared-memory connection, and closes ModelSim. This is equivalent to a *$finish* Verilog command.

The interface hides an additional execution detail from the user. If the user does not need knowledge of the DUT port states every simulation cycle (i.e. the user does not call *get_ModelSim_io*), then the *sendReceive_ModelSim_connection* function does not need to communicate to ModelSim and advance the simulation clock. Instead the function accumulates the DUT structs in a buffer until the user calls the *get_ModelSim_io* function. The interface then block transfers the DUT struct buffer to the ModelSim simulation, clocks the simulation through each DUT struct, and finally returns a DUT struct updated with the current DUT port states and returns the specified struct field value, reducing communication overhead.

E.2 Interface Generation

We accomplish automatic generation of this interface through an intelligent partitioning of the code base into static skeleton code and generated DUT-specific code, again shielding the details of this process from the user. We utilize the Direct Programming Interface (DPI) extension of the System Verilog Language. This language feature allows Verilog tasks to invoke C-functions and allow C-functions to invoke Verilog tasks. In addition we use the Matlab mex compiler to create a dynamic linked library (DLL) that the earlier mentioned access functions invoke. For simplicity there is only one user entry point for this generation process, the *DUT I/O File*. The user specifies

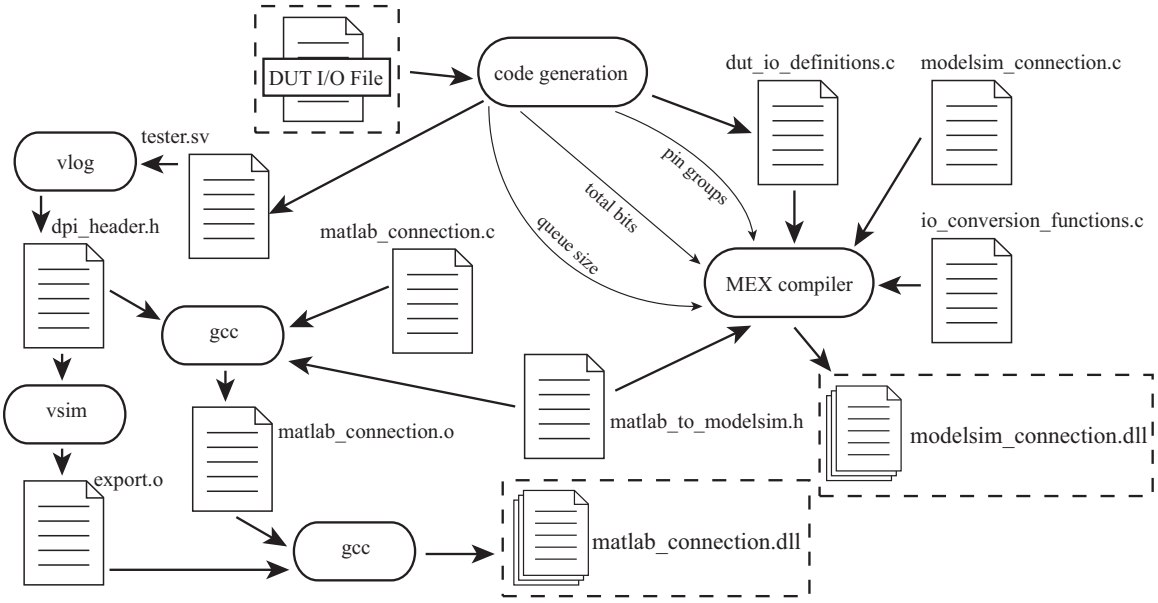

Figure 27: Interface Generation Flow-graph. The user specifies each DUT input, output, and bidirectional port name and bit-width in the DUT I/O File. The matlab_to_modelsim header file contains constants used for signaling between Matlab and ModelSim.

each DUT input, output, and bidirectional port name and bit-width in this configuration file, and then the generation script initially generates a data structure containing information about the DUT ports. It then uses this data structure to complete the steps illustrated in Figure 27.

On the ModelSim side the code generation step creates the *tester.sv* file, which is the generated tester module shown in Figure 26, using a combination of static code and generated code. The static Verilog code consists of a concurrent block that constantly calls two Verilog tasks (that have been mapped to C-functions using the DPI). One task reads an internal bus of width equal to the total number of DUT signals (*total_bits*) and sends this data to the shared-memory interface, while the other task drives this internal bus with data from the shared-memory interface. To complete these operations, the Verilog tasks call C-functions defined in *matlab_connection.c* through the System Verilog DPI. Since these functions operate on arrays of size *total_bits*, these functions need a compile parameter equal to *total_bits* but do not

need information about the DUT individual bus names or sizes. The dynamic Verilog code consists of the *tester.sv* port declaration and a set of port assignments. The port declaration mirrors the DUT inputs and outputs such that connecting the tester module to the DUT is straightforward. The port assignments consist of a listing of *assign* statements that map the port declaration to the internal bus. In both cases the generation script creates this code by using the DUT port data structure. The remaining compilation steps follow the System Verilog DPI compilation process [49] such that the final result is a DLL that links into a ModelSim simulation, enabling communication with Matlab.

The interface requirements on the Matlab side are slightly more complex since the access functions operate on DUT structs. The code generation step uses the DUT port data structure to create *dut_io_definitions.c*, which contains a DUT port name string array, a DUT port bit-width integer array, and static functions to access these arrays. This C-file allows the remaining code to be static. The functions defined in *io_conversion_functions.c* convert between the signal array that is sent over the shared memory link and the DUT structs operated on by the DUT access functions. It appears that these conversion functions should be DUT-specific, but with clever use of the information in *dut_io_definitions.c* and access to the number of DUT buses, *pin_groups*, these functions can remain generic. For example the conversion from signal array to DUT struct requires a nested loop. The outer loop processes through the listing of DUT ports using the DUT port name access function, while the inner loops processes through each bit of each port using the DUT port bit-width access function. As a result the functions in *modelsim_connection.c* provide similar functionality as the *matlab_connection.c* functions but also complete DUT struct/sign array conversion without knowledge of the DUT port names and bit-widths. In the final step the Matlab mex compiler creates a DLL from the above mentioned files.

REFERENCES

[1] Barbero, L. G., Milliner, D. L., Ratnarajah, T., Barry, J. R., and Cowan, C. F. N., "Rapid prototyping of Clarkson's lattice reduction for MIMO detection," in *Proc. of IEEE Int. Conf. on Commun.*, (Dresden, Germany), pp. 1–5, Jun. 2009.

[2] Barbero, L. G. and Thompson, J. S., "Performance of the complex sphere decoder in spatially correlated MIMO channels," *Institution of Engineering and Technology Communications*, vol. 1, pp. 122–130, Feb. 2007.

[3] Barbero, L. and Thompson, J., "FPGA design considerations in the implementation of a fixed-throughput sphere decoder for MIMO systems," in *Proc. of Int. Conf. on Field Programmable Logic and Applicat.*, (Madrid, Spain), pp. 1–6, Aug. 2006.

[4] Bruderer, L., Studer, C., Wenk, M., Seethaler, D., and Burg, A., "VLSI implementation of a low-complexity LLL lattice reduction algorithm for MIMO detection," in *Proc. of IEEE Int. Symp. on Circuits and Syst.*, (Paris, France), pp. 3745–3748, May 2010.

[5] Burg, A., Borgmann, M., Wenk, M., Zellweger, M., Fichtner, W., and Bolcskei, H., "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE J. of Solid-State Circuits*, vol. 40, pp. 1566–1577, Aug. 2005.

[6] Burg, A., Haene, S., Borgmann, M., and Baum, D., "A 4-stream 802.11n baseband transceiver in 0.13 μm CMOS," in *Proc. of Symposium on VLSI Circuits*, (Kyoto, Japan), pp. 282–283, Jun. 16-18 2009.

[7] Burg, A., Seethaler, D., and Matz, G., "VLSI implementation of a lattice-reduction algorithm for multi-antenna broadcast precoding," in *Proc. of IEEE Int. Symp. on Circuits and Syst.*, (New Orleans, USA), pp. 673–676, May 2007.

[8] Cerato, B., Masera, G., and Nilsson, P., "Hardware architecture for matrix factorization in MIMO receivers," in *Proc. of ACM Great Lakes Symposium on VLSI*, (Stresa, Italy), pp. 196–199, Mar. 11-13 2007.

[9] Choi, B., An, C., Yang, J., Jang, S., and Kim, D., "Complexity reduction for lattice reduction aided detection in MIMO-OFDM systems," in *Proc. of Comput. and Automation Eng. Conf.*, vol. 2, pp. 801 –806, Feb. 2010.

[10] Clarkson, I. V. L., *Approximation of Linear Forms by Lattice Points with Applications to Signal Processing*. PhD thesis, Australian Nat. Univ., Jan. 1997.

[11] DasSarma, D. and Matula, D., "Measuring the accuracy of ROM reciprocal tables," *IEEE Trans. Comput.*, vol. 43, pp. 932–940, Aug. 1994.

[12] Dijkstra, B. and Schumacher, L., "Description of a MATLAB implementation of the indoor MIMO WLAN channel model proposed by the IEEE 802.11 TGn channel model special committee," Jan. 2004. available online at: http://www.info.fundp.ac.be/~lsc/Research/IEEE_80211_HTSG_CMSC/distribution_terms.html.

[13] Eilert, J., Wu, D., and Liu, D., "Efficient complex matrix inversion for MIMO software defined radio," in *Proc. of IEEE Int. Symp. on Circuits and Syst.*, (New Orleans, USA), pp. 2610–2613, May 27-30 2007.

[14] Gan, Y. H. and Mow, W. H., "Complex lattice reduction algorithms for low-complexity MIMO detection," in *Proc. of IEEE Global Telecommun. Conf.*, (St. Louis, USA), pp. 2953–2957, Nov. 28-Dec. 2 2005.

[15] Gestner, B. and Anderson, D. V., "Automatic generation of modelsim-matlab interface for RTL debugging and verification," in *Proc. of IEEE Midwest Symp. on Circuits and Syst.*, (Montreal, Canada), pp. 1497–1500, 2007.

[16] Gestner, B. and Anderson, D. V., "Single Newton-Raphson iteration for integer-rounded divider for lattice reduction algorithms," in *Proc. of IEEE Midwest Symp. on Circuits and Syst.*, (Knoxville, USA), pp. 966–969, Aug. 2008.

[17] Gestner, B., Ma, X., and Anderson, D. V., "Are all basis updates for lattice-reduction-aided mimo detection necessary," *Proc. of IEEE Int. Conf. on Acoust., Speech and Signal Process.*, to be published.

[18] Gestner, B., Ma, X., and Anderson, D. V., "Incremental lattice reduction: Motivation, theory, and practical implementation," *IEEE Trans. Wireless Commun.*, to be published.

[19] Gestner, B., Zhang, W., Ma, X., and Anderson, D. V., "VLSI implementation of a lattice reduction algorithm for low-complexity equalization," in *Proc. of IEEE Int. Conf. on Circuits and Syst. for Commun.*, (Shanghai, China), pp. 643–647, May 2008.

[20] Gestner, B., Zhang, W., Ma, X., and Anderson, D. V., "VLSI implementation of an effective lattice reduction algorithm with fixed-point considerations," in *Proc. of IEEE Int. Conf. on Acoust., Speech and Signal Process.*, (Taipei, Taiwan), pp. 577–580, Apr. 2009.

[21] 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*, to be published.

[22] Golden, G. D., Foschini, G. J., Valenzuela, R. A., and Wolniansky, P. W., "Detection algorithm and initial laboratory results using V-BLAST space-time communication architecture," *Electronics Letters*, vol. 35, pp. 14–16, Jan. 7 1999.

[23] Group, I. T. W., "Joint proposal: High throughput extension to the 802.11 standard."

[24] Hassibi, B., "An efficient square-root algorithm for BLAST," in *Proc. of IEEE Int. Conf. on Acoust., Speech and Signal Process.*, (Istanbul, Turkey), pp. 737–740, Jun. 5-9 2000.

[25] Hassibi, B. and Vikalo, H., "On the sphere-decoding algorithm I. Expected complexity," *IEEE Trans. Signal Process.*, vol. 53, pp. 2806–2818, Aug. 2005.

[26] Hsiao, S. F. and Delosme, J. M., "Householder CORDIC algorithms," *IEEE Trans. Comput.*, vol. 44, pp. 990–1001, Aug. 1995.

[27] Huang, X., Liang, C., and Ma, J., "System architecture and implementation of MIMO sphere decoders on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, pp. 188–197, Feb. 2008.

[28] Jaldén, J., Seethaler, D., and Matz, G., "Worst- and average-case complexity of LLL lattice reduction in MIMO wireless systems," in *Proc. of IEEE Int. Conf. on Acoust., Speech and Signal Process.*, (Las Vegas, NV, USA), Mar. 2008.

[29] Joham, M., Barbero, L. G., Lang, T., Utschick, W., Thompson, J., and Ratnarajah, T., "FPGA implementation of MMSE metric based efficient near-ML detection," in *Proc. of IEEE 2008 Workshop on Smart Antennas*, (Germany), 2008.

[30] Joham, M., Barbero, L., Lang, T., Utschick, W., Thompson, J., and Ratnarajah, T., "FPGA implementation of MMSE metric based efficient near-ML detection," in *Proc. of Int. ITG Workshop on Smart Antennas*, (Vienna, Austria), pp. 139–146, Feb. 2008.

[31] Karkooti, M. and Cavallaro, J. R., "FPGA implementation of matrix inversion using QRD-RLS algorithm," in *Proc. of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers*, (Asilomar, USA), pp. 1625–1629, Oct. 30 - Nov. 2 2005.

[32] Kermoal, J. P., Schumacher, L., Pederson, K. I., Mogensen, P. E., and Frederiksen, F., "A stochastic MIMO radio channel model with experimental validation," *IEEE J. Sel. Areas Commun.*, vol. 20, pp. 1211–1226, Aug. 2002.

[33] Kim, H., Zhu, W., Bhatia, J., Mohammed, K., Shah, A., and Daneshrad, B., "A practical, hardware friendly MMSE detector for MIMO-OFDM-based systems," *EURASIP J. Advances Signal Process.*, vol. 2008, Jan. 2008.

[34] Kim, T. H. and Park, I. C., "Small-area and low-energy K-best MIMO detector using relaxed tree expansion and early forwarding," *IEEE Trans. Circuits Syst. I, Reg. Papers*, to be published 2010.

[35] LaMacchia, B., "Basis reduction algorithms and subset sum problems," in *Master thesis*, (MIT), May 1991.

[36] Lenstra, A. K., Lenstra, H. W., , and Lovász, L., "Factoring polynomials with rational coefficients," *Math Ann.*, vol. 261, no. 3, pp. 515–534, 1982.

[37] Ling, C. and Howgrave-Graham, N., "Effective LLL reduction for lattice decoding," in *Proc. of IEEE Int. Symp. on Inform. Theory*, (Nice, France), pp. 196–200, Jun. 2007.

[38] Ling, C. and Mow, W. H., "A unified view of sorting in lattice reduction: from V-BLAST to LLL and beyond," in *Proc. of IEEE Inf. Theory Workshop*, pp. 529–533, Jun. 2009.

[39] Liu, S., Ling, C., and Stehle, D., "Randomized lattice decoding: Bridging the gap between lattice reduction and sphere decoding," in *Proc. of IEEE Int. Symp. on Inform. Theory*, pp. 2263–2267, Jun. 2010.

[40] 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 *Proc. of IEEE Int. Symp. on Circuits and Syst.*, (New Orleans, USA), pp. 1421–1424, May 2007.

[41] Luzzi, L., Othman, G. R., and Belfiore, J., "Augmented lattice reduction for MIMO decoding," *IEEE Trans. Wireless Commun.*, vol. 9, pp. 2853–2859, Sep. 2010.

[42] Ma, X. and Zhang, W., "Fundamental limits of linear equalizers: diversity, capacity and complexity," *IEEE Trans. Inf. Theory*, vol. 54, pp. 3442–3456, Aug. 2008.

[43] Ma, X. and Zhang, W., "Performance analysis for MIMO systems with lattice-reduction aided linear equalization," *IEEE Trans. Commun.*, vol. 56, pp. 309–318, Feb. 2008.

[44] Ma, X., Zhang, W., and Swami, A., "Lattice-reduction aided equalization for OFDM sytems," *IEEE Trans. Wireless Commun.*, Feb. 2009.

[45] Ma, Z., Honary, B., Fan, P., and Larsson, E., "Stopping criterion for complexity reduction of sphere decoding," *IEEE Commun. Lett.*, vol. 13, pp. 402–404, Jun. 2009.

[46] Marr, B., Hasler, P., and Anderson, D. V., "An asynchronously embedded datapath for performance acceleration and energy efficiency," in *Proc. of IEEE Int. Symp. on Circuits and Syst.*, (Taipei, Taiwan), pp. 3046–3049, May. 24-27 2009.

[47] Matsumoto, M. and Nishimura, T., "Mersenne twister: A 623-dimensionally equidistributerd uniform pseudorandom number generator," *ACM Trans. on Modeling and Comput. Simulation*, vol. 8, pp. 3–30, Jan. 1998.

[48] Maurer, J., Matz, G., and Seethaler, D., "Low-complexity and full-diversity MIMO detection based on condition number thresholding," in *Proc. of IEEE Int. Conf. on Acoust., Speech and Signal Process.*, (Honolulu, USA), pp. 529–533, Apr. 2007.

[49] Mentor Graphics Corporation, *Questa SV/AFV User's Manual*, 2006.

[50] Milliner, D. and Barry, J. R., "A lattice-reduction-aided soft detector for multiple-input multiple-output channels," in *Proc. of IEEE Global Telecommun. Conf.*, (San Francisco, USA), Nov. 27-Dec. 1 2006.

[51] ModelTech, *ModelSim Reference Manual*, 2006.

[52] Mohammed, K. and Daneshrad, B., "A MIMO decoder accelerator for next generation wireless communications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, Accepted for Future Publication 2010.

[53] Narasimhan, R., "Finite-SNR diversity-multiplexing tradeoff for correlated rayleigh and rician MIMO channels," *IEEE Trans. Inf. Theory*, vol. 52, pp. 3965–3979, Sept. 2006.

[54] Sandell, M. and Stig, T., "Lattice-reduction-aided detection applying a modified Lenstra-Lenstra-Lovász (LLL) algorithm." Patent, Jan. 2008. WO/2008/007802.

[55] Seethaler, D. and Matz, G., "Efficient vector perturbation in multi-antenna multi-user systems based on approximate integer realtions," in *Proc. of European Signal Process. Conf.*, pp. 673–676, 2006.

[56] Seethaler, D. and Matz, G., "Efficient vector perturbation in multi-antenna multi-user systems based on approximate integer relations," in *Proc. of the European Signal Process. Conf.*, (Florence, Italy), Sept. 2006.

[57] Seethaler, D., Matz, G., and Hlawatsch, F., "Low-complexity MIMO data detection using seysen's lattice redution algorithm," in *Proc. of IEEE Int. Conf. on Acoust., Speech and Signal Process.*, (Honolulu, USA), Apr. 2007.

[58] Seysen, M., "Simultaneous reduction of a lattice basis and its reciprocal basis," *Combinatorica*, vol. 13, pp. 363–376, 1993.

[59] Shen, C. A. and Eltawil, A. M., "A radius adaptive K-best decoder with early termination: Algorithm and VLSI architecture," *IEEE Trans. Circuits Syst. I, Reg. Papers*, to be published 2010.

[60] Silvola, P., Hooli, K., and Juntti, M., "Suboptimal soft-output map detector with lattice reduction," *IEEE Signal Process. Lett.*, vol. 13, pp. 321–324, June 2006.

[61] Singh, C., Prasad, S., and Balsara, P., "A fixed-point implementation for QR decomposition," in *Proc. of IEEE Dallas/CAS Workshop on Design, Applicat., Integration and Software*, (Richardson, USA), pp. 75–78, 2006.

[62] Soler-Garrido, J., Vetter, H., Sandell, M., Milford, D., and Lillie, A., "Implementation of a reduced-lattice MIMO detector for OFDM systems," in *Proc. of Design, Automation & Test in Europe Conf. & Exhibition*, (Nice, France), pp. 1626–1631, Apr. 2009.

[63] Studer, C., Blosch, P., Friedli, P., and Burg, A., "Matrix decomposition architecture for MIMO systems: Design and implementation trade-offs," in *Proc. of the Forty-First Asilomar Conference on Signals, Systems and Computers*, (Asilomar, USA), pp. 1986–1990, Nov. 4-7 2007.

[64] Taherzadeh, M., Mobasher, A., and Khandani, A. K., "Lattice-basis reduction achieves the precoding diversity in MIMO broadcast systems," in *Proc. of 39th CISS*, (John Hopkins University, USA), Mar. 15-18 2005.

[65] Technologies Information Society IST-2000-30148 I-METRA. http://www.istimetra.org.

[66] Vetter, H., Ponnampalam, V., Sandell, M., and Hoeher, P., "Fixed complexity LLL algorithm," *IEEE Trans. Signal Process.*, vol. 57, pp. 1634–1637, Apr. 2009.

[67] Vikalo, H., Hassibi, B., and Kailath, T., "Iterative decoding for MIMO channels via modified sphere decoder," *IEEE Trans. Wireless Commun.*, vol. 3, pp. 2299–2311, Nov. 2004.

[68] Wang, F., Xiong, Y., and Yang, X., "Reliability assessment for MIMO detection and its applications," *Int. J. of Electron. and Commun.*, vol. 63, pp. 678–684, Aug. 2009.

[69] Wang, N., Biglieri, E., and Yao, K., "A systolic array for linear MIMO detection based on an all-swap lattice reduction algorithm," in *Proc. of IEEE Int. Conf. on Acoust., Speech and Signal Process.*, (Taipei, Taiwan), pp. 2461–2464, Apr. 2009.

[70] Windpassinger, C. and Fischer, R. F. H., "Low-complexity near-maximum-likelihood detection and precoding for MIMO systems using lattice reduction," in *Proc. of Inform. Theory Workshop*, pp. 345–348, Apr. 2003.

[71] Windpassinger, C., Lampe, L., and Fischer, R. F., "From lattice-reduction-aided detection towards maximum-likelihood detection in MIMO systems," in *Proc. of Int. Conf. on Wireless and Optical Commun.*, (Banff, Canada), Jul. 2003.

[72] Wu, D., Eilert, J., and Liu, D., "A programmable lattice-reduction aided detector for MIMO-OFDMA," in *Proc. of IEEE Int. Conf. on Circuits and Syst. for Commun.*, (Shanghai, China), pp. 293–297, May 2008.

[73] Wubben, D., Bohnke, R., Kuhn, V., and Kammeyer, K., "MMSE extension of V-BLAST based on sorted QR decomposition," in *Proc. of IEEE Veh. Technology Conf.*, (Orlando, FL), pp. 508–512, Oct. 2003.

[74] Wübben, D., Böhnke, R., Kühn, V., and Kammeyer, K.-D., "Mmse extension of v-blast based on sorted qr decomposition," in *Proc. of IEEE Veh. Technology Conf.*, vol. 1, (Orlando, USA), pp. 508–512, Oct. 2003.

[75] Wübben, D., Böhnke, R., Kühn, V., and Kammeyer, K.-D., "Near-maximum-likelihood detection of MIMO systems using MMSE-based lattice reduction," in *Proc. of IEEE Int. Conf. on Commun.*, vol. 2, pp. 798–802, Jun. 2004.

[76] Yang, C. and Markovic, D., "A flexible DSP architecture for MIMO sphere decoding," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, pp. 2301–2314, Oct. 2009.

[77] Yao, H. and Wornell, G. W., "Lattice-reduction-aided detectors for MIMO communication systems," in *Proc. of IEEE Global Telecommun. Conf.*, vol. 1, (Taipei, Taiwan), pp. 424–428, Nov. 2002.

[78] Yao, H. and Wornell, G. W., "Achieving the full MIMO diversity-multiplexing frontier with rotation-based space-time codes," in *Proc. of Allerton Conf. on Commun., Control and Comput.*, (IL, USA), Oct. 2003.

[79] Youssef, A., Shabany, M., and Gulak, P. G., "VLSI implementation of a hardware-optimized lattice reduction algorithm for WiMAX/LTE MIMO detection," in *Proc. of IEEE Int. Symp. on Circuits and Syst.*, (Paris, France), pp. 3541–3544, May 2010.

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

[81] 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.

[82] Zhang, W., Ma, X., Gestner, B., and Anderson, D. V., "Designing low-complexity equalizers for wireless systems," *IEEE Commun. Mag.*, vol. 47, pp. 56–62, Jan 2009.

[83] Zheng, L. and Tse, D., "Diversity and multiplexing: a fundamental tradeoff in multiple-antenna channels," *IEEE Trans. Inf. Theory*, vol. 49, pp. 1073–1196, May 2003.

VITA

Brian Gestner was born on May 18, 1981 in Minneapolis, Minnesota and graduated from Henry Sibley High School in 2000. He received his B.S. and M.S. in Electrical and Computer Engineering from Carnegie Mellon University, Pittsburgh, PA in 2004. From 2004 to 2005 he worked for Northrop Grumman Electronic Systems, developing an FPGA-based true random number generator and a direct digital synthesis application-specific integrated circuit (ASIC) for radar applications. He returned to academia in August 2005 to purse a Ph.D. degree in Electrical Engineering from the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA. While at Georgia Tech, he has worked on projects ranging from analog VLSI for audio sensing to digital VLSI realizations of lattice reduction algorithms for MIMO detection. Currently, his research involves applying an understanding of both hardware considerations and algorithm considerations toward the development of MIMO receiver algorithms and architectures.