Iterative MIMO Decoding:
Algorithms and VLSI Implementation Aspects

Christoph Studer

Iterative MIMO Decoding: Algorithms and VLSI Implementation Aspects

Hartung-Gorre Verlag Konstanz
2009

Reprint of Diss. ETH No. 18512

SERIES IN MICROELECTRONICS VOLUME 202

edited by Wolfgang Fichtner
Qiuting Huang
Heinz Jäckel
Hans Melchior
George S. Moschytz
Gerhard Tröster
Bernd Witzigmann

Bibliographic Information published by Die Deutsche Nationalbibliothek

Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the internet at: http://dnb.d-nb.de.

First edition 2009

HARTUNG-GORRE VERLAG KONSTANZ
http://www.hartung-gorre.de

ISSN 0936-5362
ISBN-10: 3-86628-288-5
ISBN-13: 978-3-86628-288-9

To Séverine ♡

Acknowledgments

First, I would like to thank my advisors Prof. Dr. W. Fichtner and Prof. Dr. H. Bölcskei for their support, for giving me the opportunity to pursue my Ph.D. Thesis on such interesting research topics, and for providing very stimulating research environments at the Integrated Systems Laboratory (IIS) and the Communication Theory Group (CTG). It is also my pleasure to express my thanks to Prof. Dr. E. Viterbo who kindly agreed to act as an external co-examiner of my Ph.D. Thesis. My very special thanks go to Prof. Dr. A. Burg and Dr. D. Seethaler for the fruitful discussions on MIMO detection and corresponding implementation aspects, their invaluable support during my Ph.D. studies, and for continuously providing me with creative ideas. Furthermore, I would like to thank the staff members of the IIS and the Design Zentrum (DZ) N. Felber, H. Kaeslin, as well as all my colleagues from the IIS and the DZ who contributed in many ways to this work, namely, C. Benkeser, F. Bürgin, M. Brändli, S. Eberli, F. Gürkaynak, S. Häne, P. Lüthi, D. Perels, C. Senning, J. Treichler, M. Wegmüller, and M. Wenk. I also want to express my thanks to my colleagues M. Borgmann, M. Gärtner, and U. Schuster from the CTG, who sparked my interest for communication theory. I finally appreciate all contributions to my Ph.D. Thesis of the following students: S. Belfanti, S. Fateh, N. Preyss, D. Riha, C. Roth, and S. Schläpfer.

Abstract

The use of multiple antennas at both ends of the wireless link is known as multiple-input multiple-output (MIMO) wireless technology and enables to transmit multiple data streams concurrently and within the same frequency band. This method is known as spatial multiplexing (SM) and improves the spectral efficiency and link reliability of wireless communication systems without requiring additional transmit power. Channel coding can be deployed to further improve the reliability of SM, which is one of the most promising technologies to meet the demands of future wireless communication systems for higher data-rates and improved quality of service.

Joint MIMO detection and channel decoding (JDD) is the optimum method for data detection (in terms of error-rate performance) in systems employing channel coding and SM. The (often) prohibitive computational complexity associated with JDD inhibits practical implementation. Iterative MIMO decoding requires significantly less computational complexity (compared to that of JDD) and was shown by Hochwald and ten Brink, *IEEE Trans. Comm.*, 2003, to enable near-optimum detection performance. The main idea underlying this approach is to separate MIMO detection from channel decoding and to iteratively exchange reliability information (i.e., soft-information) between a soft-input soft-output (SISO) detector for MIMO systems and a SISO channel decoder. So far, not much is known about the very-large-scale integration (VLSI) implementation complexity associated with iterative MIMO decoding.

In this thesis, we design and optimize algorithms for iterative MIMO decoding and investigate the associated performance and VLSI implementation aspects.

First, we optimize the detection algorithm developed by Wang and Poor, *IEEE Trans. Comm.*, 1999, for low-complexity soft-input soft-output MIMO detection. To this end, we propose a novel method that substantially reduces the computational complexity, without sacrificing performance. We design a corresponding high-throughput VLSI architecture and provide application-specific integrated circuit (ASIC) implementation results. This reference design demonstrates that SISO detection for iterative MIMO decoding is feasible in practice.

Secondly, we present a low-complexity SISO detection algorithm based on the principles of sphere decoding (SD), initially developed by Pohst, *ACM SIGSAM*, 1981. Our algorithm resorts to the single tree-search (STS) paradigm and incorporates clipping of soft-information into the tree-search, which results in significant complexity savings and allows to cover a large performance/complexity tradeoff region. In order to further reduce the complexity and to improve the performance of SISO STS-SD, we deploy a variety of techniques. The resulting algorithm clearly outperforms state-of-the-art SISO detection schemes for iterative MIMO decoding. Moreover, reference VLSI implementation results of soft-output STS-SD show that the proposed algorithm is well-suited for high-performance MIMO detection in practical systems.

Finally, we study the performance and VLSI implementation complexity associated with SISO channel decoding. To this end, we develop high-performance VLSI architectures for SISO decoding of convolutional codes, quasi-cyclic low density parity-check codes, and turbo codes. Corresponding ASIC implementation results demonstrate that high-throughput SISO channel decoding is feasible in practical systems.

Based on the results obtained throughout this thesis, we show that iterative MIMO decoding is feasible in practical systems and provide estimates of the silicon-complexity required for iterative MIMO decoding. In particular, we demonstrate that the silicon area of iterative MIMO decoding grows approximately linear in the number of iterations, while even a very small number of iterations is sufficient to approach the fundamental performance limits of MIMO wireless communication systems.

Zusammenfassung

Die MIMO (multiple-input multiple-output)-Technologie verwendet mehrere Antennen an beiden Seiten einer drahtlosen Verbindung und ermöglicht es, mehrere Datenströme gleichzeitig und im selben Frequenzband zu übertragen. Diese Technik —bekannt als räumliches Multiplexen— verbessert die spektrale Effizienz sowie die Zuverlässigkeit der Übertragung ohne die Sendeleistung zu erhöhen. Zusätzlich kann Kanalcodierung verwendet werden, um die Zuverlässigkeit der Übertragung weiter zu steigern. Räumliches Multiplexen im Verbund mit Kanalcodierung ist eine der wichtigsten Technologien, um den Ansprüchen von zukünftigen drahtlosen Kommunikationsystemen auf höheren Durchsatz und bessere Übertragungsqualität gerecht zu werden.

MIMO-Detektion mit gleichzeitiger Kanaldecodierung ist die optimale Methode um die Fehlerrate in solchen Systemen zu minimieren. Die benötigte Rechenleistung erlaubt es jedoch nicht, diese Methode in praktischen Systemen zu verwenden. Methoden, welche auf iterativer MIMO-Decodierung basieren, benötigen signifikant weniger Rechenleistung und erreichen dabei, wie Hochwald und ten Brink, *IEEE Trans. Comm.,* 2003, gezeigt haben, fast gleichwertige Fehlerraten. Die Kernidee dieser Verfahren besteht darin, MIMO-Detektion und Kanaldecodierung zu trennen und Zuverlässigkeitsinformation (soft-information) zwischen den beiden Komponenten auf iterative Weise auszutauschen; dies benötigt einen sogenannten SISO (soft-input soft-output)-MIMO-Detektor sowie einen SISO-Kanaldecoder. Bis Heute ist jedoch nicht viel über die Komplexität von VLSI (very-large scale integration)-Schaltungen für iterative MIMO-Decodierung bekannt.

Im Rahmen dieser Arbeit werden Algorithmen für iterative MIMO-Decodierung entwickelt und optimiert, sowie deren Leistungsfähigkeit und VLSI-Implementationskomplexität analysiert.

Zuerst wird der Detektionsalgorithmus von Wang und Poor, *IEEE Trans. Comm.,* 1999 untersucht und für bessere Effizienz optimiert. Dabei wird eine neue Methode angewendet, welche es ermöglicht die Komplexität des Algorithmus signifikant zu reduzieren ohne die Leistungsfähigkeit zu verringern. Basierend auf den in dieser Arbeit optimierten Algorithmus wurde eine VLSI-Schaltung implementiert, welche sehr hohe Effizienz erreicht und damit beweist, dass SISO-Detektion für iterative MIMO-Decodierung tatsächlich in die Praxis umgesetzt werden kann.

Als zweiten Beitrag dieser Arbeit präsentieren wir einen neuen SISO-Detektionsalgorithmus für MIMO-Systeme, welcher auf dem Prinzip von Sphere-Decoding (SD) —entwickelt von Pohst, *ACM SIGSAM,* 1981— aufbaut. Dieser Algorithmus benutzt das STS (single tree-search) Verfahren und bezieht das Limitieren von Zuverlässigkeitsinformation in die Baumsuche mit ein. Diese Technik führt zu einer massiven Komplexitätsreduktion und ermöglicht einen umfassenden Abtausch zwischen Leistungsfähigkeit und Komplexität. Ausserdem zeigt der resultierende Algorithmus bessere Leistungsfähigkeit und geringere Komplexität als bestehende SISO-Detektionsalgorithmen für iterative MIMO-Decodierung. Weiterhin zeigen wir VLSI-Implementationsresultate einer soft-output-Variante vom STS-SD Algorithmus und demonstrieren damit, dass diese Methode optimale Leistungsfähigkeit in praktischen Systemen erreichen kann.

Zuletzt wird der VLSI-Schaltungsaufwand und die Leistungsfähigkeit von verschiedenen (state-of-the-art) SISO-Kanaldecodern analysiert. Zu diesem Zweck wurden dedizierte VLSI-Implementationen für die SISO-Decodierung von Faltungscodes, LDPC (low-density parity check)-Codes, sowie Turbocodes entwickelt, welche einen hohen Durchsatz erreichen. Die daraus resultierenden Implementationsresultate zeigen, dass effiziente SISO-Decodierung für alle drei Codierungsverfahren in der Praxis möglich ist.

Basierend auf den Resultaten, die in dieser Arbeit erhalten wurden, zeigen wir, dass iterative MIMO-Decodierung in praktische Systeme umgesetzt werden kann. Zudem schätzen wir den entsprechenden

Flächenverbrauch ab und zeigen, dass die benötigte Schaltungsfläche ungefähr linear mit der Anzahl Iterationen ansteigt, wobei jedoch nur eine sehr geringe Zahl von Iterationen benötig wird, um die optimale Leistungsfähigkeit von drahtlosen MIMO-Kommunikationssystemen näherungsweise zu erreichen.

Contents

Chapter 1

Introduction

During the last decade, many wired communication systems are being replaced by corresponding wireless services. With the increasing availability of portable computers and personal digital assistants, for example, wireless services have shifted from voice-based to multimedia-oriented applications. Such services often tend to require even higher data rates. The trend towards throughput-intensive applications is summarized by Edholm's Law, which states that data rates of wired and wireless communication systems double every 18 months [1]. Novel technologies, the evolution of wireless communication standards, and corresponding low-cost devices, are key to follow this trend, achieving better quality of service (QoS) and supporting a large amount of users that communicate simultaneously. The recent development of the IEEE 802.11n wireless local area network (WLAN) standard [2], for example, indicates that future wireless systems will be able to support peak data rates in the range of several hundred megabits up to gigabits per second, while offering the same reliability and data rates as their corresponding wired counterparts.

Simply allocating more bandwidth or increasing the transmit power are *not* viable solutions to keep up with the growing demand for higher data rates and a large number of users, while meeting stringent QoS requirements (such as link reliability, coverage, and range). As bandwidth has become an extremely scarce (and hence expensive) resource, simply increasing the bandwidth is neither feasible nor eco-

nomic. Increasing the transmit power is not practicable as well, since the maximum transmit power is regulated in most of the available frequency bands, e.g., in order to prevent health hazards. In addition, a high transmit power would significantly reduce the battery lifetime of portable devices and cause interference to other users communicating nearby, which reduces the potential of re-using of frequency bands.

In order to meet the contradictory requirements for higher data rates, better QoS, and a large number of users, while maintaining the transmit power and bandwidth, novel technologies need to be considered. Multiple-input multiple-output (MIMO) wireless communication is believed to be the key technology to meet these demands, because it improves both, the data rate for a given bandwidth (which is also known as spectral efficiency) *and* the QoS of wireless communication systems.

1.1 MIMO Wireless Technology

Wireless channels suffer from signal fading, caused by destructive interference of multi-path propagation, and from interference caused by other services sharing the same frequency band. The use of multiple antennas at the transmitter and/or the receiver enables significant improvements in terms of link reliability, range, and spectral efficiency compared to that of single-antenna systems. However, fully exploiting these gains comes, in general, at the cost of significantly increased signal-processing complexity, especially in the receiver.[1]

1.1.1 MIMO Gains

Employing multiple antennas only at the transmitter *or* the receiver, is known as multiple-input single-output (MISO) and single-input multiple-output (SIMO), respectively. Communication with multiple antennas at *both* sides of the wireless link is known as multiple-input multiple-output (MIMO) technology. The gains enabled by multi-antenna technologies can be categorized as follows (see, e.g., [3–5]).

[1] Note that it is not always feasible to obtain *all* gains at once. The performance improvements of multi-antenna technology rather depends on the underlying signaling schemes.

Diversity Gain The main idea underlying diversity corresponds to transmitting the same signal over independently fading links (also known as diversity branches). Since not all links fade concurrently with high probability, combining all versions of the transmit signal in the receiver mitigates fading effects. Note that the concept of diversity is strongly related to link reliability: For a higher number of independently fading diversity branches, fading becomes less likely.

There are three main sources of diversity that can be exploited in single-user wireless systems: i) temporal diversity, which is caused by the delay spread of the signal, ii) frequency diversity, which is caused by the Doppler spread, and iii) spatial diversity (also known as antenna diversity). While single-antenna systems offer temporal and frequency diversity (i.e., signals can be transmitted at different time instances or over different frequencies), multiple antennas at the transmitter and/or the receiver enables spatial diversity, i.e., the signal can be transmitted over different paths in space. This technique has become popular in wireless communication systems since it improves link reliability—in contrast to time or frequency diversity—without reducing the data rate or increasing the bandwidth. In MIMO systems, the maximum amount of spatial diversity corresponds to the number of transmit antennas times the number of receive antennas (if all paths fade independently). Thus, MIMO technology is able to significantly reduce fading effects and it offers to improve the QoS of wireless communication systems.

For portable low-complexity and battery-powered wireless devices it is often not feasible to employ multiple antennas, due to stringent space and power limitations. For such systems, multiple antennas are often employed only at the base-station (where power consumption and space is not an issue) and space-time coding is employed; this technique enables to obtain transmit diversity and it, therefore, improves link reliability without the need for channel-state information at the transmitter. In particular, space-time block codes (STBCs) [6,7], such as the Alamouti scheme [8], have emerged as promising means to offer transmit diversity in practical systems.

Array Gain refers to an increase in average receive signal-to-noise ratio (SNR) by the coherent combination (i.e., by alignment of the

phases using channel knowledge at the receiver) of all signals picked up at multiple receive antennas.[2] The increase in terms of average SNR grows proportionally to the number of receive antennas; simply speaking: "more receive antennas pick up more signal energy." Hence, array gain as well improves the QoS of wireless communication systems.

Multiplexing Gain Array gain and spatial diversity can be obtained through multiple antennas at the transmitter *or* at the receiver. Employing multiple antennas at *both ends* of the wireless link (i.e., MIMO technology) permits the concurrent transmission of multiple data streams within the same frequency band. This technique is known as spatial multiplexing (SM) and it is able to yield a linear increase (in the minimum number of transmit or receive antennas) in terms of system capacity [9]. It is important to note that the SM gain improves the spectral efficiency and comes at no expense of increased transmit power. Since no additional transmit power is necessary, the potential for bandwidth re-using by users communicating nearby is improved as well. Hence, SM is key to achieve high data rates, which is the main reason for the adoption of this technology by many modern wireless communication standards, such as IEEE 802.11n [2], IEEE 802.16e [10], and the third-generation partnership project (3GPP) consortium long-term evolution (LTE) standard [11].

1.1.2 The Role of MIMO Decoding

The physical layer of a receiver for MIMO wireless systems consist of several parts, such as radio frequency (RF) components, synchronization circuitry, channel estimation, and the MIMO decoder. The MIMO decoder consists of a MIMO detector to separate the spatially multiplexed data streams and a channel decoder, which computes estimates of the transmitted information bits. The performance of MIMO systems is heavily affected by the performance of the MIMO decoder. In order to fully exploit the gains offered by MIMO technology, it

[2] Note that the array gain can also be obtained by employing multiple antennas at the transmitter, which requires transmit-side channel knowledge.

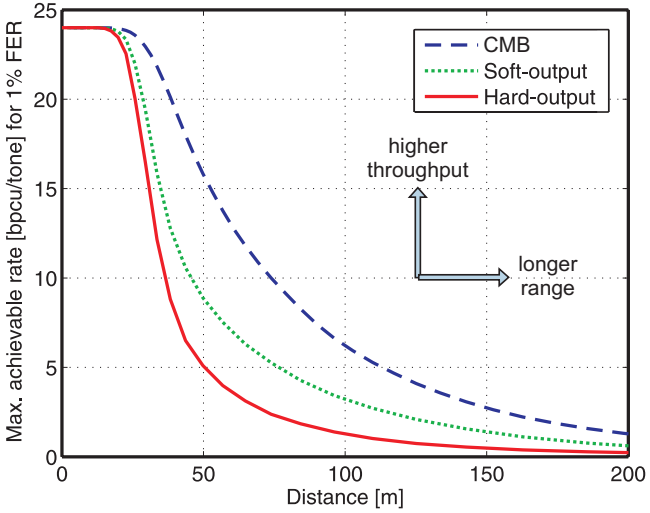


Figure 1.1: Distance (between transmitter and receiver) versus maximum achievable rates for 64-QAM data transmission in a 4-stream MIMO-OFDM system.

is essential to employ high-performance MIMO decoding algorithms. However, such algorithms often entail an extremely high amount of signal processing complexity. The tradeoff between performance and complexity inherently present in MIMO decoding is briefly illustrated below.

Performance/Complexity Tradeoff

Performance Figure 1.1 shows maximum achievable rates at a given distance (between transmitter and receiver) for two different MIMO detection algorithms (i.e., optimum hard-output detection and max-log optimal soft-output detection) as well as the theoretical limit of the considered system, referred to as coded-modulation bound (CMB).[3] We can see that hard-output MIMO detection re-

[3]This simulation shows the maximum achievable rate in bits per channel use (bpcu) per OFDM tone for a target frame error-rate (FER) of 1%. We consider an

sults in worse performance (in terms of achievable rates and range) than soft-output MIMO detection. It is important to note that soft-output MIMO detection requires, in general, more sophisticated (and hence, more complex) algorithms compared to hard-output MIMO detection (see Section 2.2). Hence, more sophisticated MIMO detection schemes allow an increased achievable rate at a given distance to the transmitter, or an increased range (i.e., the maximum distance where a given target rate is supported). We emphasize that there is still a substantial gap between soft-output MIMO detection and the theoretical performance limit. Hence, employing even more sophisticated MIMO detection schemes might further approach the CMB.

Complexity Figure 1.2 illustrates the computational complexity associated with various algorithms employed in MIMO decoders.[4] The algorithms and implementations provided in this thesis have been highlighted in this figure; they are able to approach the theoretical optimum (i.e., the CMB). We can see that more sophisticated algorithms (such as soft-output sphere decoding or SISO MMSE PIC) require higher computational complexity compared to that of low-complexity algorithms for MIMO decoding (e.g., soft-output MMSE detection or hard-output MIMO detection using the sphere decoder). Hence, we conclude that employing MIMO decoding schemes that are able to approach the theoretical performance limit, entail—as it

IEEE 802.11n-like [2] MIMO-OFDM system, with four transmit and four receive antennas, 64 OFDM tones, 64-QAM modulation with Gray mapping. The distance is measured in meters and has been computed from the average receive SNR according to the TGn type C path-loss model (for 20 MHz bandwidth and 2.4 GHz carrier frequency) [12]. We consider an implementation loss of 4 dB SNR to obtain more realistic results, i.e., the average receive SNR has been reduced by 4 dB. The curves associated with "hard-output" and "soft-output" correspond to that of hard-output maximum-likelihood (ML) detection and max-log-optimal soft-output a posteriori probability (APP) performance, respectively. The corresponding rates are obtained through Monte-Carlo simulations of the mutual information between the transmitted bits and the log-likelihood ratios or the hard-outputs computed by the MIMO detectors. The curve associated with the CMB refers to the maximum achievable rate when using 64-QAM modulation (see [13]), i.e., corresponds to an upper bound on the system performance.

[4] Figure 1.2 has been adapted from [14, Fig. 2.2] and [15, Fig. 1.1] using results of [16]. The algorithms and implementation results described in this thesis have been highlighted. The computational complexity is in operations per second (ops).

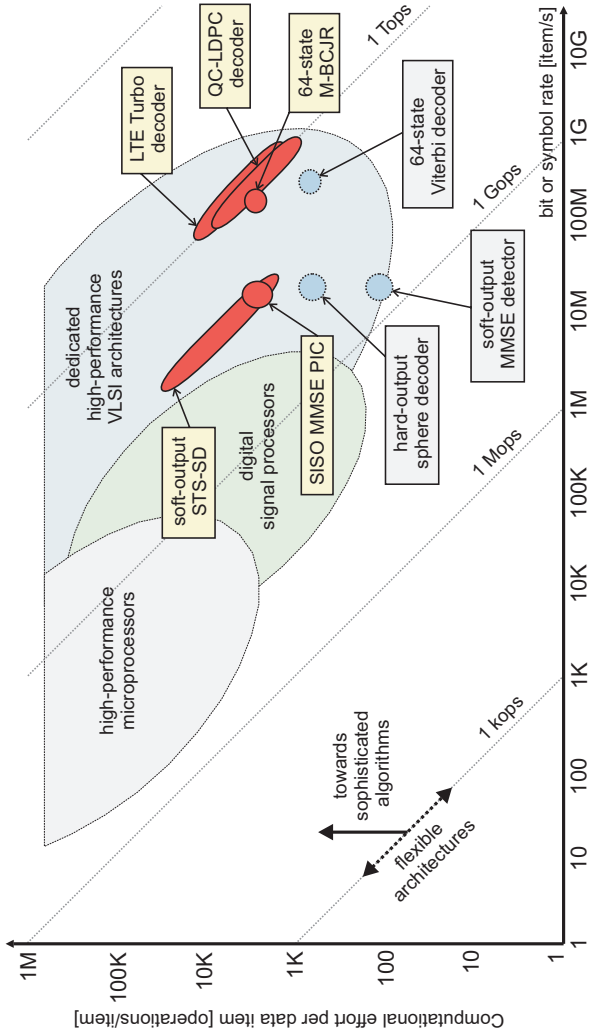


Figure 1.2: Computational needs for different MIMO detection and channel decoding algorithms. Implementations presented in this thesis are highlighted.

will be shown in the remainder of this thesis—an (often significant) increase in computational complexity.

Furthermore, we can see from Figure 1.2 that the algorithms considered in this thesis require dedicated high-performance very-large-scale integration (VLSI) architectures to meet the computational requirements. As it was shown in [15], MIMO decoding contributes substantially to the VLSI implementation complexity of the total receiver and strongly affects the corresponding costs. The ultimate MIMO receiver must, therefore, achieve best decoding performance at lowest possible VLSI implementation complexity. This goal can only be achieved by jointly considering algorithm and VLSI implementation aspects and by performing careful investigations of the underlying performance/complexity tradeoffs.

The State-of-the-Art in MIMO Decoding

Almost all recent MIMO receiver designs that can be found in the open literature employ linear MIMO detection schemes, see e.g., [16–18]. Note that linear detection offers acceptable error-rate performance while requiring low computational complexity (cf. Figure 1.2). Hence, most of the currently available implementations lie on the low-complexity low-performance end of the performance/complexity tradeoff.

On the algorithmic side, state-of-the-art MIMO decoders employ non-linear MIMO detection schemes based on sphere decoding (SD). Moreover, such decoders often perform iterative MIMO detection and channel decoding, which is known as *iterative MIMO decoding* in the literature [19]. This approach is believed to be a key technology to achieve near-optimum performance in MIMO wireless systems. However, the associated computational complexity is significant and—up to now—not much is known about the performance and complexity of corresponding VLSI implementations.

1.2 Contributions of this Work

The goal of this thesis is to design and optimize algorithms for iterative MIMO decoding and to investigate the associated performance

and VLSI implementation aspects. To this end, we analyze and improve soft-input soft-output MIMO detection algorithms for iterative MIMO decoding and present novel low-complexity solutions. In addition, we describe corresponding VLSI architectures and provide reference application-specific integrated circuit (ASIC) implementation results. Moreover, we compare the performance and VLSI implementation complexity associated with SISO channel decoding. Finally, all results obtained throughout this work allow for a detailed analysis of the performance/complexity tradeoffs associated with iterative MIMO decoding. Based on this analysis, we describe best-practices for hardware-efficient iterative MIMO decoding.

In summary, this thesis provides novel high-performance and low-complexity solutions for iterative MIMO decoding and an analysis of the associated performance/complexity trade-offs. To the best of the authors knowledge, all presented VLSI implementations are currently ranked among the best performing. The contributions of this thesis are detailed in the following.

SISO MMSE Parallel Interference Cancellation

In Chapter 3, we describe the first VLSI implementation of a soft-input soft-output (SISO) detector for iterative MIMO decoding. The algorithm underlying our VLSI implementation is known as SISO minimum mean-square error (MMSE) parallel interference cancellation (PIC) and was invented by Wang and Poor [20] for iterative detection in multi-user code division multiple access (CDMA) systems.

The algorithm in its original form exhibits prohibitive computational complexity due to the requirement of *multiple* matrix inversions at symbol-rate. For economic hardware implementation, we describe a novel method which requires only one matrix inversion at symbol-rate, without affecting the algorithm's performance. The (remaining) matrix inversion task is performed by a novel high-performance architecture based on the LU-decomposition. In addition, a variety of tricks on algorithmic and architectural level have been employed in order to attain a low-complexity and high-performance VLSI architecture of the optimized SISO MMSE PIC algorithm.

The resulting architecture has been integrated in 90 nm CMOS technology. The ASIC implementation results demonstrate that SISO

detection based on the proposed SISO MMSE PIC algorithm enables high-throughput soft-input soft-output MIMO detection in practice.

SISO Single Tree-Search Sphere Decoding

Chapter 4 summarizes and extends our results in [21–27]. We describe a SISO single tree-search (STS) sphere decoding (SD) algorithm that is tunable between max-log optimal SISO and hard-output maximum a posteriori (MAP) detection performance. We develop a max-log optimal a priori information processing method, which significantly reduces the tree-search complexity compared to [28–32] and avoids the computation of transcendental functions. The basic idea for complexity reduction and tunability of the algorithm is to incorporate clipping of the extrinsic log-likelihood ratios (LLRs) into the tree search.

We furthermore propose a method for self-interference compensation in the LLRs —caused by channel-matrix regularization— directly in the tree search. Due to prohibitively high worst-case complexity of SD, we propose early termination based on a novel scheduling technique, which is well-suited for implementation in practical systems, while only slightly degrading the algorithm's performance. In addition, we describe a new method to correct approximate LLRs from sub-optimal detectors, which (often significantly) improves detection performance at low additional computational complexity. Simulation results show that the resulting SISO STS-SD algorithm operates close to outage capacity at remarkably low computational complexity. In addition, the proposed algorithm clearly outperforms state-of-the-art SISO detection schemes for iterative MIMO systems.

We describe the first VLSI implementation of the soft-output (SO) STS-SD algorithm, based on the one-node-per-cycle VLSI implementation for hard-output Schnorr-Euchner SD developed in [33]. Corresponding VLSI implementation results demonstrate that the proposed algorithm is suitable for low-complexity and high-performance MIMO detection in practical systems.

Soft-Input Soft-Output Channel Decoding

In Chapter 5, we describe three different high-throughput channel decoder implementations for SISO decoding of convolutional codes

(CCs), low-density parity check (LDPC) codes, and turbo codes:

- For SISO decoding of CCs, we design a high-throughput architecture for the M-BCJR algorithm [34] and we derive corresponding VLSI implementations that support different constraint lengths (ranging from three to seven). Hardware-level optimizations yield significant improvements in circuit area and decoding throughput. The resulting M-BCJR architectures have been implemented in 180 nm CMOS technology. For constraint length seven, the architecture is compliant with IEEE 802.11n [2] and is the first of its kind described in the open literature.

- For SISO decoding of LDPC codes, we summarize our results presented in [35]. We develop a high-throughput architecture that is able to decode quasi-cyclic (QC) LDPC codes. The design of a novel permutation network and a new method to combine control signals with the information contained in the parity-check matrix enable re-configurability of the decoder (at run-time) for virtually all QC-LDPC codes that fit into the allocated memories. Corresponding VLSI implementation results in 180 nm CMOS technology demonstrate that the performance, energy-efficiency, and area are comparable to that of dedicated (i.e., non re-configurable) architectures. The resulting decoder implementation is compliant with the IEEE 802.11n standard.

- We describe a turbo decoder for the 3GPP LTE standard [11]. Our architecture has been optimized for high throughput by using an optimized radix-4 M-BCJR architecture in combination with a novel interleaver architecture. Corresponding VLSI implementation results in 130 nm CMOS technology show that the achieved throughput is 15 to 25 times higher than that of reference implementations, e.g., [36, 37], while being more efficient in terms of area per throughput.

Finally, we compare the three SISO decoders in terms of VLSI implementation complexity and error-rate performance and analyze the underlying tradeoffs.

Performance/Complexity Tradeoff Investigation

Chapter 6 compares the VLSI implementation complexity of iterative MIMO decoding (i.e., of soft-input soft-output MIMO detection and SISO channel decoding) with the associated (error-rate) performance. The performance/complexity evaluation is based on a method called throughput matching, which simplifies the tradeoff analysis and enables to provide an estimate of the silicon complexity required for an iterative MIMO decoder.

We compare the performance/complexity tradeoffs of linear MIMO detection and SISO MMSE PIC using CCs, LDPC codes, and turbo codes. Moreover, we compare the performance/complexity tradeoff realized by the SISO MMSE PIC to that of the SO STS-SD implementation. Finally, we propose guidelines for the design of hardware-efficient high-performance MIMO decoders.

1.3 Notation

Matrices are set in boldface capital letters, vectors in boldface lowercase letters. The superscripts ^{T}, ^{H}, and * stand for transpose, conjugate transpose, and (element-wise) complex conjugation, respectively. We write $A_{i,j}$ for the entry in the ith row and jth column of the matrix $\mathbf{A}$ and b_i for the ith entry of the vector $\mathbf{b} = [\, b_1 \ \cdots \ b_N \,]^T$. $\mathbf{I}_N$ and $\mathbf{0}_{M \times N}$ denote the $N \times N$ identity matrix, and the $M \times N$-dimensional all-zero matrix, respectively. $\mathbf{1}_M$ is the all-ones vector of dimension M. Slightly abusing common terminology, we call an $N \times M$ matrix $\mathbf{A}$, where $N \geq M$, satisfying $\mathbf{A}^H\mathbf{A} = \mathbf{I}_M$, unitary. The ℓ^2-norm of a vector $\mathbf{b}$ is denoted by $\|\mathbf{b}\|$. The probability of an event $\mathcal{Z}$ is denoted by $\mathrm{P}[\mathcal{Z}]$, the probability density function (PDF) of a continuous random variable (RV) z is denoted by $\mathrm{p}(z)$. $\mathbb{E}[Z]$ and $\mathrm{Var}[Z]$ stand for the expectation and variance of the RV Z, respectively. The real and imaginary part of $x \in \mathbb{C}$ is denoted by $\Re\{x\}$ and $\Im\{x\}$, respectively. The binary complement of $x \in \{+1, -1\}$ is $\overline{x} = -x$. The round, ceil, floor, and signum operations of $x \in \mathbb{R}$ are denoted by $\lceil x \rfloor$ $\lceil x \rceil$, $\lfloor x \rfloor$, and $\mathrm{sign}(x)$, respectively. $\mathbb{CZ}$ stands for the set of Gaussian integers, i.e., $\mathbb{CZ} = \mathbb{Z} + \sqrt{-1}\,\mathbb{Z}$, and $|\mathcal{O}|$ designates the cardinality of the set $\mathcal{O}$.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 describes the system model, introduces iterative MIMO decoding, and briefly reviews the basics of hard-output and (soft-input) soft-output MIMO detection. We conclude this chapter by a performance comparison of the reference detector with some of the most prominent sub-optimal low-complexity MIMO detection schemes. In Chapter 3, a SISO detection algorithm for iterative MIMO decoding is optimized for VLSI implementation. Finally, we show corresponding ASIC implementation results of the first SISO detector for iterative MIMO decoding reported in the literature. A novel high-performance low-complexity SISO detection algorithm based on SD is described in Chapter 4. A variety of techniques are employed to further reduce its computational complexity. Finally, we propose reference VLSI implementation results of a soft-output variant of the proposed algorithm. Chapter 5 describes VLSI implementations for SISO decoding of convolutional, LDPC, and turbo codes. A detailed tradeoff comparison in terms of performance and complexity concludes this chapter. In Chapter 6, we provide—based on implementation results provided in the previous three chapters—performance/complexity tradeoff results for iterative MIMO decoding (including the MIMO detector and the channel decoder). We conclude in Chapter 7 and outline open research topics.

Chapter 2

Iterative MIMO Decoding

Optimum detection performance in coded MIMO systems in terms of minimizing the error-rate performance, is obtained by joint detection and decoding (JDD). The computational complexity associated with JDD is, in general, prohibitively high, even for short code-blocks (see [38]). *Iterative MIMO decoding* is believed to be the most promising approach for low-complexity and near-optimum data detection in coded MIMO systems. The main idea underlying iterative detection and decoding is the assumed *independence* of MIMO detection and channel decoding. This assumption enables to solve both tasks *separately*, which entails, in general, significantly less computational complexity compared to that of JDD. Since MIMO detection and channel decoding have been separated, reliability information on the coded bits is exchanged between the two components in an iterative fashion. When the iteration process is stopped, the channel decoder produces estimates of the transmitted bits.

Iterative decoding reaches back to 1963, when Gallager proposed an iterative decoding method for LDPC codes in his visionary Ph.D. thesis [39]. Three decades later, iterative decoding has been rediscovered by Berrou *et al.* for decoding of turbo codes [40]. The exceptional performance achieved by iterative decoding of turbo codes,

and the additional generalization of iterative decoding of virtually any linear block-code by Hagenauer *et al.* in 1996 [41], sparked tremendous research activities. In 1999, the concept of iterative detection and decoding has been developed by Wang and Poor for data detection in multi-user (MU) CDMA systems [20]. In 2002, the idea has been adapted for detection in inter-symbol interference (ISI) channels by Tüchler *et al.* [42]. In the same year, iterative MIMO decoding has been proposed for MIMO wireless communication systems by Witzke *et al.* [43]. In 2003, Hochwald and ten Brink demonstrated that iterative MIMO decoding is able to achieve near channel-capacity when using a SD-based soft-input soft-output MIMO detection algorithm in combination with turbo codes [19].

In the remainder of this chapter, we introduce the system model and describe the concept of iterative MIMO decoding (Section 2.1). We outline some of the most relevant hard-output and soft-output MIMO detection algorithms in Section 2.2 and conclude with corresponding simulation results that demonstrate the potential of iterative MIMO decoding.

2.1 System Model

Consider an iterative MIMO system with M_{T} transmit and M_{R} receive antennas. In the remainder of this thesis, we assume $M_{\mathrm{R}} \geq M_{\mathrm{T}}$. The MIMO system is depicted in Figure 2.1 and consists of three main components: i) the transmitter, ii) the MIMO channel, and iii) the iterative MIMO decoder; all components are described in the following.

2.1.1 Transmitter

The MIMO transmitter obtains a sequence of information bits, denoted by the B-dimensional binary-valued vector $\mathbf{b}$. This vector is referred to as data frame in the remainder of this thesis. In order to enable reliable transmission, bit-interleaved coded modulation (BICM) is performed [44, 45]. To this end, the channel encoder introduces redundancy in the data frame and performs interleaving of the coded bits. The resulting coded and interleaved bit-stream is denoted by the

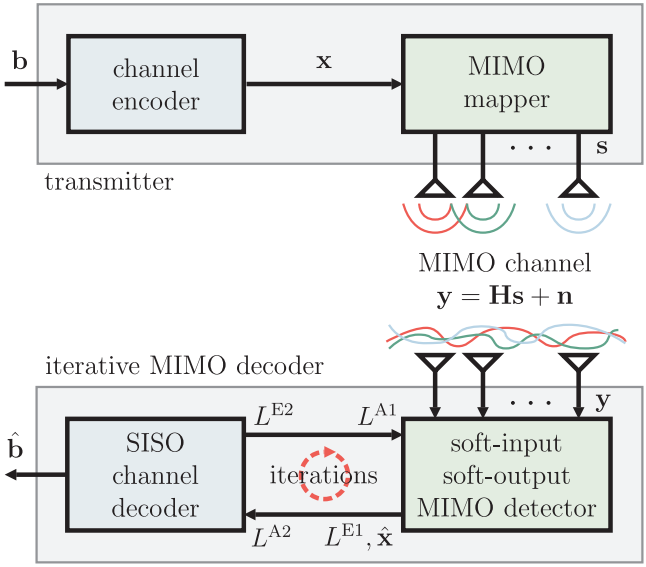


Figure 2.1: MIMO wireless communication system employing BICM, spatial multiplexing (SM), and iterative MIMO decoding [19].

binary-valued vector $\mathbf{x}$ of dimension B/R, where $0 < R \leq 1$ denotes the rate of the employed channel code.[1]

The coded bit-stream is mapped to a sequence of M_{T}-dimensional symbol vectors $\mathbf{s}[k] \in \mathcal{O}^{M_{\mathrm{T}}}$, where $\mathcal{O}$ denotes the underlying complex scalar constellation set with $|\mathcal{O}| = 2^Q$ and $k = 1, \ldots, N$. The index k corresponds to the kth transmitted symbol vector.[2] Each symbol vector $\mathbf{s}[k]$ is associated with $M_{\mathrm{T}}Q$ bits, denoted by $x_{i,b}[k]$ (for $i = 1, \ldots, M_{\mathrm{T}}$ and $b = 1, \ldots, Q$); the indices i and b refer to the bth bit in the binary label of the ith entry of the symbol vector $\mathbf{s}[k]$. The bits are chosen from the set $\{+1, -1\}$ where the null element (0 in binary logic) of GF(2) corresponds to $+1$. The bijective mapping between bits $x_{i,b}[k]$ $(\forall i, b)$ and entries of the symbol vector $\mathbf{s}[k] =$

[1] Zero-padding is employed if B is not an integer multiple of $1/R$.

[2] In orthogonal frequency division multiplexing (OFDM) systems, k stands for the kth OFDM tone and N denotes the maximum number of used tones.

$\left[s_1[k] \ \cdots \ s_{M_\mathrm{T}}[k] \right]^T$ is

$$s_i[k] = \mathrm{map}\left(x_{i,b}[k], b = 1, \ldots, Q\right). \tag{2.1}$$

The inverse mapping of (2.1) is denoted by $\left[s_i[k]\right]_b = x_{i,b}[k]$. The total number of transmitted information bits per data frame corresponds to $B = RNM_\mathrm{T}Q$ bit.

2.1.2 MIMO Channel

The symbol vectors $\mathbf{s}[k]$ ($\forall k$) are transmitted over N MIMO channels. The complex baseband input-output relation of the (frequency-flat) MIMO channel is given by[3]

$$\mathbf{y}[k] = \mathbf{H}[k]\mathbf{s}[k] + \mathbf{n}[k] \tag{2.2}$$

where $\mathbf{H}[k]$ stands for the kth complex-valued $M_\mathrm{R} \times M_\mathrm{T}$ channel matrix, $\mathbf{y}[k]$ denotes the M_R-dimensional received signal vector, and $\mathbf{n}[k]$ is an i.i.d. circularly symmetric complex Gaussian distributed M_R-dimensional noise vector with variance N_o per complex entry, i.e., $\mathbf{n}[k] \sim \mathcal{CN}(0, N_\mathrm{o}\mathbf{I}_{M_\mathrm{R}})$, $\forall k$. The element $H_{i,j}[k]$ of the MIMO channel matrix represents the complex-valued transfer function from the jth transmit to the ith receive antenna. Different transmit powers on the individual transmit antennas are assumed to be absorbed in the channel matrices $\mathbf{H}[k]$, which—including the corresponding scaling factors—will be referred to as the physical MIMO channels. In the remainder of this thesis we assume that the transmitter has no channel state information, whereas the receiver is assumed to have perfect channel state information, i.e., the $\mathbf{H}[k]$ ($\forall k$) and N_o are perfectly known to the receiver.[4]

The elements of the N channel matrices are usually assumed to be i.i.d. (across space and index k) circularly symmetric complex Gaussian distributed with unit variance (e.g., [9]). This assumption is,

[3] Note that the input-output relation given in (2.2) does not only model MIMO wireless communication using SM, but it can also represent MIMO channels including linear STBCs (e.g., [46]), or other communication channels, such as ISI channels.

[4] Pilot symbols are employed in practice to estimate the channel state in the receiver. However, perfect channel state information is not possible, since the so obtained estimates are disturbed by noise [47].

however, optimistic, since real-world wireless channels usually exhibit correlation across space, time, or frequency. To this end, the IEEE 802.11 task group N (TGn) specified typical channel models for WLAN applications [12]. These models enable to obtain simulation results that are more relevant in practice. Unless specified otherwise, the TGn type C channel model—corresponding to a typical residential or small office environment with 30 ns root mean square (RMS) delay spread [12]—is used throughout this thesis.

Transmit symbols are normalized such that $\mathbb{E}\left[|s_i|^2\right] = E_\mathrm{s}$ $(\forall i)$ and the SNR definition used in this thesis refers to the average SNR per receive antenna defined as $\mathsf{SNR} = M_\mathrm{T} E_\mathrm{s}/N_\mathrm{o}$. For the sake of simplicity of exposition, the symbol-vector index k is omitted in the remainder of this thesis.

2.1.3 Iterative MIMO Decoder

The structure of an iterative MIMO decoder is depicted in Figure 2.1 and consists of a SISO detector for MIMO systems and a SISO channel decoder. The detector and the decoder iteratively exchange reliability information for each coded bit. The iterative detection process is stopped if a maximum number of iterations is reached. Then, the SISO channel decoder produces (binary-valued) estimates of the data bits $\hat{\mathbf{b}}$.

Soft-Input Soft-Output MIMO Detector

The task of the soft-input soft-output MIMO detector is to undo the MIMO mapping and the influence of the MIMO channel, which is accomplished on the basis of the received vector $\mathbf{y}$, the channel state information (i.e., $\mathbf{H}$ and N_o), and soft-inputs (i.e., a priori reliability information) in the form of LLRs, denoted by L^{A1} (see Figure 2.1). The MIMO detector either produces hard decisions for each transmitted bit $\hat{x}_{i,b}$ $(\forall i, b)$ or it computes corresponding a posteriori soft-outputs in the form of *extrinsic* LLRs, denoted by $L^{\mathrm{E1}}_{i,b}$. The vector containing all a posteriori extrinsic LLR values is denoted by L^{E1} (see Figure 2.1).

MIMO detectors that compute soft-outputs on the basis of soft-

inputs are referred to as soft-input soft-output (SISO) detectors for MIMO systems. Detectors that compute hard-output estimates (possibly based on a priori information) are referred to as hard-output MIMO detectors. The most prominent hard-output and SISO detection schemes are outlined in Section 2.2.

Soft-Input Soft-Output Channel Decoder

In iterative MIMO decoding, the SISO channel decoder serves two purposes: i) it computes new a posteriori soft-outputs (in the form of extrinsic LLRs), denoted by L^{E2}, on the basis of the a priori LLRs L^{A2} delivered by the MIMO detector (see Figure 2.1) and ii) the SISO channel decoder computes hard-output estimates of the transmitted information bits $\hat{\mathbf{b}}$ at the end of the iteration process.[5] It is important to note that the a posteriori LLRs of the channel decoder L^{E2} will get a priori LLRs L^{A1} of the MIMO detector (i.e., $L^{\mathrm{E2}} = L^{\mathrm{A1}}$) and the extrinsic a posteriori LLRs L^{E1} will be a priori LLR-values L^{A2} of the channel decoder, i.e., $L^{\mathrm{E1}} = L^{\mathrm{A2}}$ (see Figure 2.1).

Different types of channel codes and corresponding SISO decoding algorithms exist in the literature. The most prominent variants are reviewed in Chapter 5 together with corresponding VLSI implementation results.

2.2 The Basics of MIMO Detection

In Section 2.2.1, we review hard-output MIMO detection algorithms. The basics of soft-input soft-output MIMO detection algorithms are described in Section 2.2.2.

2.2.1 Hard-Output MIMO Detection

Coherent hard-output MIMO detection algorithms compute estimates of the transmitted symbol vector $\hat{\mathbf{s}}$. Estimates of the transmitted bits are obtained by remapping the computed symbol-vector estimate $\hat{\mathbf{s}}$ to

[5] Note that if interleaving is used, the LLRs need to be de-interleaved at the input and interleaved at the output of the channel decoder.

its corresponding bit-labels according to

$$\hat{x}_{i,b} = [\hat{s}_i]_b, \quad i = 1, \ldots, M_{\mathrm{T}}, \quad b = 1, \ldots, Q$$

where $[\hat{s}_i]_b$ stands for the bth bit in the label of $\hat{s}_i$. In the following, hard-output MIMO detection schemes (ranging from optimum (error-rate) performance to low computational complexity) are reviewed and the associated performance is compared based on numerical simulation results.

Maximum a Posteriori and Maximum-Likelihood Detection

The optimal hard-output MIMO detector (in terms of minimizing the probability that the computed estimate does not correspond to the transmitted symbol vector $\mathbf{s}'$) is referred to as the maximum a posteriori (MAP) detector and corresponds to

$$\hat{\mathbf{s}}^{\mathrm{MAP}} = \underset{\mathbf{s} \in \mathcal{O}^{M_{\mathrm{T}}}}{\arg\max}\, \mathrm{P}[\mathbf{s}' = \mathbf{s} \,|\, \mathbf{y}, \mathbf{H}] \tag{2.3}$$

where $\hat{\mathbf{s}}^{\mathrm{MAP}}$ is referred to as the MAP estimate. Bayes's theorem applied to (2.3) leads to the following equivalent formulation of the MAP detection rule

$$\begin{aligned} \hat{\mathbf{s}}^{\mathrm{MAP}} &= \underset{\mathbf{s} \in \mathcal{O}^{M_{\mathrm{T}}}}{\arg\max} \left\{ \mathrm{p}(\mathbf{y} \,|\, \mathbf{s}' = \mathbf{s}, \mathbf{H}) \, \frac{\mathrm{P}[\mathbf{s}' = \mathbf{s}]}{\mathrm{p}(\mathbf{y})} \right\} \\ &= \underset{\mathbf{s} \in \mathcal{O}^{M_{\mathrm{T}}}}{\arg\max} \left\{ \mathrm{p}(\mathbf{y} \,|\, \mathbf{s}' = \mathbf{s}, \mathbf{H}) \, \mathrm{P}[\mathbf{s}' = \mathbf{s}] \right\} \end{aligned} \tag{2.4}$$

where the second equality results from the fact that $\mathrm{p}(\mathbf{y})$ does not depend on $\mathbf{s}$. Due to the assumptions made on the noise statistics (see Section 2.1.2), the PDF $\mathrm{p}(\mathbf{y} \,|\, \mathbf{s}' = \mathbf{s}, \mathbf{H})$ in (2.4) corresponds to the joint probability density function of a multi-variate complex Gaussian with i.i.d. circularly symmetric components (each having variance N_{o}) with mean $\mathbb{E}[\mathbf{y}] = \mathbf{H}\mathbf{s}$, i.e.,

$$\mathrm{p}(\mathbf{y} \,|\, \mathbf{s}' = \mathbf{s}, \mathbf{H}) = \frac{1}{(\pi N_{\mathrm{o}})^{M_{\mathrm{R}}}} \exp\left(- \frac{\|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2}{N_{\mathrm{o}}} \right).$$

Since $\log(x)$ is monotonically increasing in x, maximizing the logarithm of (2.4) is equivalent; this leads to the well-known MAP detection rule for MIMO systems [28–32, 48]

$$\hat{\mathbf{s}}^{\mathrm{MAP}} = \arg\min_{\mathbf{s}\in\mathcal{O}^{M_{\mathrm{T}}}} \left\{ \frac{\|\mathbf{y}-\mathbf{H}\mathbf{s}\|^2}{N_{\mathrm{o}}} - \log \mathrm{P}[\mathbf{s}'=\mathbf{s}] \right\}. \tag{2.5}$$

If all transmit symbol vectors are equally likely, the MAP solution (2.5) coincides with the maximum likelihood (ML) solution of the MIMO detection problem [3–5]

$$\hat{\mathbf{s}}^{\mathrm{ML}} = \arg\min_{\mathbf{s}\in\mathcal{O}^{M_{\mathrm{T}}}} \|\mathbf{y}-\mathbf{H}\mathbf{s}\|^2 \tag{2.6}$$

which is a closest-vector problem (CVP); solving this problem is known to require high computational complexity [49]. Straightforward MIMO detection according to (2.5) or (2.6), by performing an exhaustive search over all possible transmit vectors, can lead to prohibitively high computational complexity. For example, MAP or ML detection in a MIMO system with $M_{\mathrm{T}} = 4$ and 64-QAM modulation alphabet requires a comparison of $|\mathcal{O}|^{M_{\mathrm{T}}} \approx 16.7 \cdot 10^6$ candidates. In order to avoid the prohibitive computational complexity associated with solving of (2.5) or (2.6), a variety of low-complexity MIMO detection schemes have been proposed in the past.

Linear MIMO Detection

The most prominent low-complexity algorithms for MIMO detection belong to the class of linear detection (LD) schemes [3]. The main idea underlying LD is to invert the effect of the MIMO channel matrix. To this end, the MIMO detection problem is decomposed into M_{T} single-antenna detection problems that can be solved with (often significantly) reduced complexity compared to that of MAP or ML detection. However, the decomposition in M_{T} (independent) detection tasks leads, in general, to a significant performance loss. Corresponding simulation results are shown below.

Zero-Forcing Detection One of the simplest approaches to low-complexity MIMO detection is zero-forcing (ZF), which corresponds

to left-multiplying the Moore-Penrose pseudo-inverse of the channel matrix $\mathbf{H}^{\dagger} = (\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$ to the received vector $\mathbf{y}$ such that

$$\mathbf{H}^{\dagger}\mathbf{y} = \hat{\mathbf{y}}^{\mathrm{ZF}} = \mathbf{s} + \mathbf{H}^{\dagger}\mathbf{n}. \tag{2.7}$$

By ignoring the influence of colored noise in (2.7), detection can be performed *separately* for each transmit stream according to

$$\hat{s}_i^{\mathrm{ZF}} = \left\lceil \hat{y}_i^{\mathrm{ZF}} \right\rfloor_{\mathcal{O}}, \quad i = 1, \ldots, M_{\mathrm{T}}$$

where $\lceil y \rfloor_{\mathcal{O}}$ denotes mapping of $y \in \mathbb{C}$ to the nearest constellation point in $\mathcal{O}$, i.e.,

$$\lceil y \rfloor_{\mathcal{O}} = \arg\min_{s \in \mathcal{O}} |y - s|.$$

Computing the pseudo-inverse and slicing are both of polynomial complexity (in M_{T}) and hence, ZF is a polynomial-time algorithm for MIMO detection. Note that in (2.7), interference from other streams is perfectly removed, i.e., ZF decomposes the MIMO detection problem in M_{T} parallel single-stream problems. However, if the channel matrix is ill-conditioned, ZF leads to noise enhancement since the equalized noise $\mathbf{H}^{\dagger}\mathbf{n}$ in (2.7) can get arbitrarily large [3]; this is the main cause of the poor error-rate performance realized by ZF detection.

MMSE Detection LD based on the MMSE criterion is a solution to trade noise enhancement for interference suppression. Instead of using the pseudo-inverse of the channel matrix, an $M_{\mathrm{T}} \times M_{\mathrm{R}}$ estimator matrix $\mathbf{M}$ that fulfills the MMSE criterion

$$\mathbf{M} = \arg\min_{\widetilde{\mathbf{M}} \in \mathbb{C}^{M_{\mathrm{T}} \times M_{\mathrm{R}}}} \mathbb{E}\left[\|\widetilde{\mathbf{M}}\mathbf{y} - \mathbf{s}\|^2 \right] \tag{2.8}$$

is used for equalization. Note that expectation in (2.8) is over the noise and the symbol vectors. The solution of (2.8) corresponds to the well-known MMSE estimator matrix [3]

$$\mathbf{M} = \left(\mathbf{H}^H\mathbf{H} + \frac{M_{\mathrm{T}}}{\mathsf{SNR}}\mathbf{I}_{M_{\mathrm{T}}} \right)^{-1} \mathbf{H}^H. \tag{2.9}$$

Equalization of the MIMO input-output relation with (2.9) leads to

$$\mathbf{My} = \widetilde{\mathbf{H}}\mathbf{s} + \mathbf{Mn}$$

with $\widetilde{\mathbf{H}} = \mathbf{MH}$. As noted in [50], the MMSE estimator in (2.9) is biased, i.e., the entries on the main-diagonal of $\widetilde{\mathbf{H}}$ are, in general, not equal to one. An unbiased MMSE estimator can be obtained by extraction of the main-diagonal entries of $\widetilde{\mathbf{H}}$ into a $M_{\mathrm{T}} \times M_{\mathrm{T}}$ diagonal matrix

$$\mathbf{D} = \mathrm{diag}\big(\tilde{H}_{1,1}, \ldots, \tilde{H}_{M_{\mathrm{T}},M_{\mathrm{T}}}\big)$$

and by using the *unbiased* MMSE estimator matrix $\widetilde{\mathbf{M}} = \mathbf{D}^{-1}\mathbf{M}$ instead of (2.9).

Analogous to ZF, linear (and unbiased) MMSE detection corresponds to the application of the unbiased MMSE estimator matrix $\widetilde{\mathbf{M}}$ to the received vector $\mathbf{y}$ according to

$$\widetilde{\mathbf{M}}\mathbf{y} = \hat{\mathbf{y}}^{\mathrm{MMSE}} = \mathbf{s} + \tilde{\mathbf{n}} \tag{2.10}$$

and detection is carried out by slicing $\hat{y}_i^{\mathrm{MMSE}}$ to the nearest constellation point in $\mathcal{O}$ for $i = 1, \ldots, M_{\mathrm{T}}$. The vector $\tilde{\mathbf{n}}$ in (2.10) corresponds to the effective noise-plus-(self)-interference (NPI)

$$\tilde{\mathbf{n}} = \big(\widetilde{\mathbf{M}}\mathbf{H} - \mathbf{I}_{M_{\mathrm{T}}}\big)\mathbf{s} + \widetilde{\mathbf{M}}\mathbf{n}. \tag{2.11}$$

It is important to note that the NPI-term in (2.11) now contains self-interference, but suffers less from noise enhancement than ZF detection. The reason for reduced noise enhancement is due to the fact that the inversion in (2.9) is based on a regularized version of the Gram matrix $\mathbf{H}^H\mathbf{H}$, which is usually better conditioned than computation of the pseudo-inverse (as required for ZF). Thus, LD based on the MMSE criterion achieves, in general, better performance than ZF, while requiring a similar amount of computational complexity.

Successive Interference Cancellation

Similar to LD, successive interference cancellation (SIC) decomposes the MIMO detection problem into M_{T} single-stream detection problems. The key difference is that SIC performs detection in a sequential

manner, i.e., one stream is detected after another while the result of the previously detected stream is used to cancel out interference in the subsequent detection steps. The detection process can be described conveniently using the QR-decomposition (QRD) of the MIMO channel matrix. In the sequel, the general case of SIC, i.e., including layer sorting and regularization, is described.

SIC starts by performing a column-sorted QRD of a regularized version of the channel matrix

$$\begin{bmatrix} \mathbf{H} \\ \alpha \mathbf{I}_{M_\mathrm{T}} \end{bmatrix} \mathbf{P} = \mathbf{Q}\mathbf{R} \tag{2.12}$$

where α is a suitably chosen regularization parameter, $\mathbf{P}$ is a $M_\mathrm{T} \times M_\mathrm{T}$ permutation matrix, $\mathbf{Q}$ is a unitary $(M_\mathrm{R}+M_\mathrm{T}) \times M_\mathrm{T}$ matrix and $\mathbf{R}$ is of dimension $M_\mathrm{T} \times M_\mathrm{T}$ and upper-triangular with non-negative real-valued entries on the main diagonal. Partitioning $\mathbf{Q}$ according to $\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_a^T & \mathbf{Q}_b^T \end{bmatrix}^T$, where $\mathbf{Q}_a$ is of dimension $M_\mathrm{R} \times M_\mathrm{T}$ and $\mathbf{Q}_b$ is of dimension $M_\mathrm{T} \times M_\mathrm{T}$, and left multiplication of the received vector by $\mathbf{Q}_a^H$, leads to the input-output relation

$$\mathbf{Q}_a^H \mathbf{y} = \hat{\mathbf{y}}^{\mathrm{SIC}} = \mathbf{R}\tilde{\mathbf{s}} + \tilde{\mathbf{n}} \tag{2.13}$$

where $\mathbf{P}\tilde{\mathbf{s}} = \mathbf{s}$ and the effective NPI vector is

$$\tilde{\mathbf{n}} = -\alpha \mathbf{Q}_b^H \mathbf{s} + \mathbf{Q}_a^H \mathbf{n}.$$

Note that $\mathbf{Q}$ is unitary, but $\mathbf{Q}_a$ and $\mathbf{Q}_b$ will, in general, not be unitary. Hence, the effective NPI vector is no longer i.i.d. circularly symmetric complex Gaussian distributed with variance N_o per complex entry. In addition, since $\mathbf{Q}_b$ is not an all-zero matrix if $\alpha \neq 0$, the NPI vector contains self-interference.

MIMO detection based on SIC now amounts to detect the M_Tth stream according to

$$\hat{s}_{M_\mathrm{T}}^{\mathrm{SIC}} = \left\lceil \frac{\hat{y}_{M_\mathrm{T}}^{\mathrm{SIC}}}{R_{M_\mathrm{T},M_\mathrm{T}}} \right\rfloor_{\mathcal{O}}$$

followed by successively detecting the remaining streams

$$\hat{s}_i^{\mathrm{SIC}} = \left\lceil \frac{1}{R_{i,i}} \left(\hat{y}_i^{\mathrm{SIC}} - \sum_{j=i+1}^{M_\mathrm{T}} R_{i,j} \hat{s}_j^{\mathrm{SIC}} \right) \right\rfloor_{\mathcal{O}}, \quad i = M_\mathrm{T}-1, \ldots, 1.$$

The procedure described above resembles the solution of linear equations using back-substitution, with the key difference that the substituted variables are being sliced to the nearest constellation symbols.

Various flavors of SIC have been proposed in the literature. The (error-rate) performance mainly depends on the choice of the regularization parameter α and the column-sorting strategy. The best-performing variant is the vertical Bell Laboratories layered space-time (V-BLAST) algorithm [51, 52], which originally employs multiple matrix inversions instead of a QRD. The original V-BLAST algorithm [51] does not use any regularization (i.e., $\alpha = 0$) and it performs column sorting such that the layer with largest post-equalization SNR is detected first. An improved algorithm (in terms of performance and complexity) was proposed in [53] and employs regularization according to the MMSE criterion, i.e., $\alpha = \sqrt{M_{\mathrm{T}}/\mathsf{SNR}}$, and it processes spatial streams with highest post-equalization signal-to-noise-and-interference ratio (SINR) first.

A low-complexity alternative that approximates V-BLAST layer sorting is based on the sorted QR-decomposition (SQRD) algorithm as described in [54]. The performance of this algorithm can be improved by using regularization [55] as well. Note that SQRD-based SIC requires lower computational complexity compared to inversion-based V-BLAST algorithms (see, e.g., [56]).

Lattice-Reduction-Aided MIMO Detection

A promising approach for high-performance and low-complexity hard-output MIMO detection is based on a mathematical tool known as lattice reduction (LR). The general idea of LR-aided MIMO detection is to relax the ML detection problem to a CVP on the infinite lattice. Then, a CVP is solved on a "more orthogonal" lattice basis, which leads to performance improvements in combination with low-complexity detection methods, such as LD or SIC. The basic idea of LR-aided MIMO detection is summarized below.

Transformation to Lattices In order employ techniques from lattice theory to MIMO detection, we start by mapping the elements $s \in \mathcal{O}$ to elements $x \in \mathbb{C}\mathbb{Z}$ using the transformation $x = as + c$. The constants $a \in \mathbb{R}$ and $c \in \mathbb{C}$ with $a > 0$ are independent of s and chosen

such that $x \in \mathcal{X} \subset \mathbb{CZ}$ with $|\mathcal{X}| = |\mathcal{O}|$ and

$$\mathcal{X} = \big\{x \in \mathbb{CZ} \,|\, (k_{\min} \leq \Re\{x\} \leq k_{\max}) \\ \wedge (k_{\min} \leq \Im\{x\} \leq k_{\max})\big\} \tag{2.14}$$

where $k_{\min}, k_{\max} \in \mathbb{Z}$. Note that (2.14) can be used to check whether $x' \in \mathbb{CZ}$ is in $\mathcal{X}$ by performing separate boundary checks for the real and imaginary part of x'.[6] The transmit vectors $\mathbf{s} \in \mathcal{O}^{M_{\mathrm{T}}}$ can be mapped to vectors $\mathbf{x} \in \mathcal{X}^{M_{\mathrm{T}}} \subset (\mathbb{CZ})^{M_{\mathrm{T}}}$ according to

$$\mathbf{x} = a\mathbf{s} + \mathbf{c} \tag{2.15}$$

where $\mathbf{c} = c\mathbf{1}_{M_{\mathrm{T}}}$. The inverse transformation associated with (2.15) is given by $\mathbf{s} = a^{-1}(\mathbf{x} - \mathbf{c})$. The input-output relation in (2.2) can now be transformed into

$$\mathbf{r} = \mathbf{G}\mathbf{x} + \mathbf{n} \tag{2.16}$$

where $\mathbf{G} = a^{-1}\mathbf{H}$ and $\mathbf{r} = \mathbf{y} + \mathbf{G}\mathbf{c}$ is a translated version of the received vector $\mathbf{y}$. The essence of the transformation of (2.2) into (2.16) is that now the received vector $\mathbf{r}$ can be interpreted as a lattice point $\mathbf{u} \in \mathcal{L}(\mathbf{G})$ that has been translated by the additive Gaussian noise vector $\mathbf{n}$. Here,

$$\mathcal{L}(\mathbf{G}) \triangleq \Big\{\mathbf{G}\mathbf{x} \,|\, \mathbf{x} \in \mathcal{X}^{M_{\mathrm{T}}}\Big\} \tag{2.17}$$

denotes the finite lattice generated by $\mathbf{G}$.

Relaxation and Lattice Reduction After carrying out the transformation to the lattice $\mathcal{L}(\mathbf{G})$, ML-detection (MLD) for (2.16) corresponds to

$$\hat{\mathbf{u}}^{\mathrm{ML}} = \underset{\mathbf{u} \in \mathcal{L}(\mathbf{G})}{\arg\min} \|\mathbf{r} - \mathbf{u}\|^2 \tag{2.18}$$

which amounts to solving a CVP in the finite lattice $\mathcal{L}(\mathbf{G})$. Since each lattice point in $\mathcal{L}(\mathbf{G})$ is associated with a transmit vector in $\mathcal{X}^{M_{\mathrm{T}}}$

[6] In the case of non-square QAM constellations the boundary checks take a slightly more complicated form.

according to the relation $\mathbf{u} = \mathbf{G}\mathbf{x}$, the ML-estimate $\hat{\mathbf{u}}^{\mathrm{ML}}$ obtained by solving (2.18) can be transformed into[7] $\hat{\mathbf{x}}^{\mathrm{ML}} = \mathbf{G}^{\dagger}\hat{\mathbf{u}}^{\mathrm{ML}}$, which upon inversion of (2.15) yields the ML-estimate $\hat{\mathbf{s}}^{\mathrm{ML}}$ in (2.6).

The key requirement to employ LR for detection is to relax the finite lattice in (2.17) to the *infinite* lattice[8]

$$\underline{\mathcal{L}}(\mathbf{G}) \triangleq \left\{ \mathbf{G}\underline{\mathbf{x}} \,|\, \underline{\mathbf{x}} \in (\mathbb{C}\mathbb{Z})^{M_{\mathrm{T}}} \right\} \tag{2.19}$$

and to compute an equivalent and "more orthogonal" basis for the lattice $\underline{\mathcal{L}}(\mathbf{G})$ with the generator matrix $\mathbf{B} = \mathbf{G}\mathbf{T}$, where $\mathbf{T}$ is an $M_{\mathrm{T}} \times M_{\mathrm{T}}$ unimodular matrix, i.e., $|\det(\mathbf{T})| = 1$ with $T_{i,j} \in \mathbb{C}\mathbb{Z}$ $(\forall i, j)$. Thanks to the unimodularity of $\mathbf{T}$, we have $\underline{\mathcal{L}}(\mathbf{B}) = \underline{\mathcal{L}}(\mathbf{G})$. It is important to note that this equivalence only holds for infinite lattices, in contrast to finite lattices where, in general, $\mathcal{L}(\mathbf{B}) \neq \mathcal{L}(\mathbf{G})$.

LR-Aided MIMO Detection LR-aided MIMO detection is then performed on the relaxed and reduced lattice $\underline{\mathcal{L}}(\mathbf{B})$ by computing

$$\hat{\underline{\mathbf{u}}}^{\mathrm{ML}} = \underset{\underline{\mathbf{u}} \in \underline{\mathcal{L}}(\mathbf{B})}{\arg\min} \|\mathbf{r} - \underline{\mathbf{u}}\|^2 \tag{2.20}$$

followed by compensating the transformation caused by the unimodular matrix according to $\mathbf{T}\hat{\underline{\mathbf{u}}}^{\mathrm{ML}} = \hat{\underline{\mathbf{x}}}^{\mathrm{ML}}$. Since the resulting estimate $\hat{\underline{\mathbf{x}}}^{\mathrm{ML}}$ is not necessarily in $\mathcal{X}^{M_{\mathrm{T}}}$, remapping onto $\mathcal{X}^{M_{\mathrm{T}}}$ is required whenever $\hat{\underline{\mathbf{x}}}^{\mathrm{ML}} \notin \mathcal{X}^{M_{\mathrm{T}}}$. Quantization of $\hat{\underline{x}}_i^{\mathrm{ML}}$ to the nearest constellation point in $\mathcal{X}$ $(\forall i)$ is the "common" approach for remapping used in the literature [57, 58].

Approximating (2.20) using low-complexity detection algorithms, such as LD or SIC, results in often significant performance improvements compared to LD or SIC without relaxation and LR [57, 59]. Unfortunately, remapping of $\hat{\underline{\mathbf{x}}}^{\mathrm{ML}}$ onto $\mathcal{X}^{M_{\mathrm{T}}}$ if $\hat{\underline{\mathbf{x}}}^{\mathrm{ML}} \notin \mathcal{X}^{M_{\mathrm{T}}}$ entails, in general, a significant performance loss (in terms of an SNR gap) compared to ML performance. Regularization of the channel matrix can reduce this performance gap up to a certain extent [58]. However, the only way to mitigate the performance loss associated with relaxation

[7] Note that $\mathbf{G}^{\dagger}$ does not have to be computed explicitly as $\mathbf{u}$ in (2.18) can be replaced by $\mathbf{G}\mathbf{x}$ and the minimization can be performed over $\mathbf{x} \in \mathcal{X}^{M_{\mathrm{T}}}$.

[8] In the remainder of this section, underlined quantities refer to the infinite-lattice case.

is to solve a finite-lattice CVP [25]; this again results in high computational complexity. Generally speaking, LR-aided MIMO detection seems to be better suited for use in combination with low-complexity hard-output detectors, such as LD or SIC, with overall sub-optimal performance.

Performance of Hard-Output MIMO Detection

Diversity Gain A fundamental performance measure of MIMO detection algorithms is the *diversity gain* d, which refers to the asymptotic slope of the error probability as the SNR goes to infinity, i.e., [5]

$$d \triangleq -\lim_{\mathsf{SNR}\to\infty} \frac{\log\big(\mathrm{P_e}(\mathsf{SNR})\big)}{\log(\mathsf{SNR})} \tag{2.21}$$

where $\mathrm{P_e}(\mathsf{SNR})$ denotes the average (over noise and channel realizations) error probability of the considered MIMO detector. A large diversity gain is desirable since it ensures that the error rate decreases faster for increasing SNR values than it would for a detector with a low diversity gain.

For MIMO systems employing spatial multiplexing in i.i.d. (across space) Rayleigh fading channels, the optimum diversity gain is attained by MLD (2.6) and it corresponds to $d = M_{\mathrm{R}}$ [60, 61]. The diversity gain of ZF- and MMSE-based linear detection was shown to be $d = M_{\mathrm{R}} - M_{\mathrm{T}} + 1$, which is worse compared to that of MLD [61]. SIC yields, in general, better performance than linear detection schemes, but it achieves the *same* diversity gain as LD, irrespective of the employed column-ordering strategy and regularization [62]. Hence, the diversity gain for LD and SIC can only be improved by using more receive than transmit antennas.

The performance of LR-aided hard-output MIMO detection fundamentally differs from LD or SIC. In [63] it was shown that LR through the Lenstra, Lenstra, and Lovász (LLL) algorithm [64] followed by LD achieves the same diversity gain as MLD. Naive lattice decoding[9] was shown to achieve the same diversity gain as MLD, while leading to an unbounded SNR gap (growing logarithmically in SNR)

[9]Naive lattice decoding [65] refers to ML decoding on the infinite lattice and to declaring an error whenever the relaxed estimate $\underline{\hat{\mathbf{x}}}$ does not belong to $\mathcal{X}^{M_{\mathrm{T}}}$.

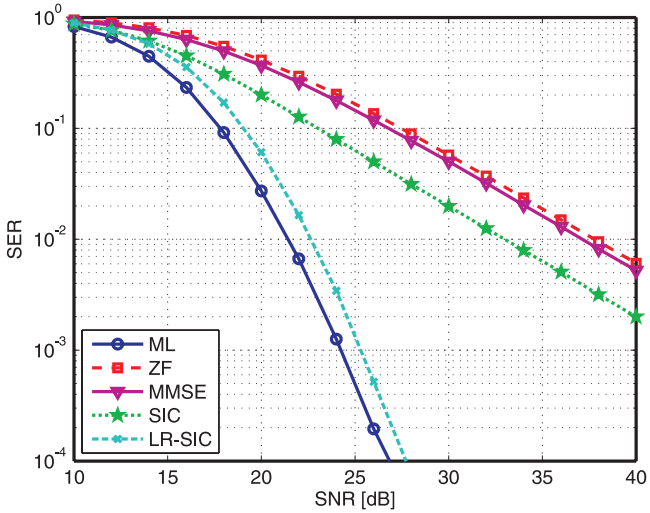


Figure 2.2: Symbol error rate (SER) performance of various MIMO detection schemes.

for $M_{\mathrm{R}} = M_{\mathrm{T}}$ [65]. We finally note that the true performance of LR-aided detection strongly depends on the remapping strategy [25].

Numerical Performance Since the diversity gain only characterizes the asymptotic performance behavior of MIMO detection schemes, numerical performance results using symbol (vector) error rate (SER) are shown in Figure 2.2. The simulation results are for an *uncoded* $M_{\mathrm{T}} = M_{\mathrm{R}} = 4$ MIMO system with 16-QAM symbol constellation. The entries of $\mathbf{H}$ are i.i.d. circularly symmetric complex Gaussian distributed with unit variance. For the LR-aided SIC detector (LR-SIC), lattice reduction has been carried out with a complex-valued version of the LLL algorithm [66] with $\delta = 3/4$ using regularized (MMSE) SQRD preprocessing [55] and remapping to the finite lattice is done by using quantization [25, 55].

In Figure 2.2, the diversity gain of all MIMO detectors can be identified in the high-SNR regime. ML detection and LR-aided SIC yield the same 4th order diversity gain, whereas LR-aided SIC loses

approximately 1.2 dB SNR (at SER=10^{-2}) to MLD. LD using ZF or MMSE show almost the same SER performance (and achieve only first order diversity), whereas regularized SIC using SQRD [55] yields approximately 4 dB SNR gain compared to MMSE-based LD. We conclude that it is essential to employ detection schemes that realize the maximum diversity gain, particularly in uncoded (or near rate-one coded) MIMO systems.

2.2.2 Soft-Input Soft-Output MIMO Detection

Iterative MIMO decoding requires soft-input soft-output (SISO) detection schemes to compute a posteriori reliability information for the coded bits based on the received vector, on channel state information, and on a priori reliability information. In order to perform this task, a variety of algorithms exists in the literature. The optimum SISO detection method and a prominent low-complexity algorithm are described below. In the remainder of this section, we only focus on the MIMO detector and omit the superscript ^{1} in the LLR notation, i.e., we write L^{A} and L^{E} instead of L^{A1} and L^{E1}, respectively.

Exact A Posteriori Probability MIMO Detection

Optimum performance in iterative MIMO systems can be achieved by computing exact a posteriori probabilities (APPs) in the form of *intrinsic* a posteriori LLRs [19, 67]

$$L_{i,b} \triangleq \log\left(\frac{\mathrm{P}[x_{i,b}=+1\,|\,\mathbf{y},\mathbf{H}]}{\mathrm{P}[x_{i,b}=-1\,|\,\mathbf{y},\mathbf{H}]}\right) \tag{2.22}$$

for all bits $i = 1,\ldots,M_{\mathrm{T}}$, $b = 1,\ldots,Q$ in the label $\mathbf{x}$. Note that the sign of an LLR-value L indicates whether the corresponding bit $x_{i,b}$ is more likely to be $+1$ or -1, while the magnitude $|L_{i,b}|$ denotes the reliability of the estimate $\hat{x}_{i,b} = \mathrm{sign}(L_{i,b})$. Large magnitudes indicate high confidence, whereas low magnitudes correspond to estimates with low reliability.

Bayes's theorem applied to (2.22) leads to the equivalent formula-

tion of intrinsic a posteriori LLRs

$$L_{i,b} = \log\left(\sum_{\mathbf{s}\in\mathcal{X}_{i,b}^{(+1)}} \mathrm{p}(\mathbf{y}\,|\,\mathbf{s}'=\mathbf{s},\mathbf{H})\,\mathrm{P}[\mathbf{s}'=\mathbf{s}]\right) - \log\left(\sum_{\mathbf{s}\in\mathcal{X}_{i,b}^{(-1)}} \mathrm{p}(\mathbf{y}\,|\,\mathbf{s}'=\mathbf{s},\mathbf{H})\,\mathrm{P}[\mathbf{s}'=\mathbf{s}]\right) \tag{2.23}$$

where $\mathcal{X}_{i,b}^{(+1)}$ and $\mathcal{X}_{i,b}^{(-1)}$ are the sets of symbol vectors that have the bit corresponding to the indices i and b equal to $+1$ and -1, respectively.

SISO detection for iterative MIMO decoding requires the computation of a posteriori LLRs while considering a priori information. The a priori information (also known as soft-input) in (2.23) is considered by means of the so-called prior $\mathrm{P}[\mathbf{s}'=\mathbf{s}]$. This prior can be computed on the basis of a priori LLRs $L_{i,b}^{\mathrm{A}}$, which represent the probability of the (transmitted) bit being more likely $+1$ or -1, i.e.,

$$L_{i,b}^{\mathrm{A}} = \log\left(\frac{\mathrm{P}[x_{i,b}=+1]}{\mathrm{P}[x_{i,b}=-1]}\right), \quad \forall i,b. \tag{2.24}$$

From (2.24) it follows that

$$\mathrm{P}[x_{i,b}=+1] = \frac{\exp\left(L_{i,b}^{\mathrm{A}}\right)}{1+\exp\left(L_{i,b}^{\mathrm{A}}\right)} \tag{2.25}$$

$$\mathrm{P}[x_{i,b}=-1] = \frac{1}{1+\exp\left(L_{i,b}^{\mathrm{A}}\right)} \tag{2.26}$$

and, since the bits $x_{i,b}$ are independent among spatial streams $i = 1,\ldots,M_{\mathrm{T}}$ *and* among bits $b=1,\ldots,Q$ in systems employing BICM, the prior term in (2.23) can be rewritten as

$$\mathrm{P}[s_i'=s_i] = \prod_{b:x_{i,b}=+1} \frac{\exp\left(L_{i,b}^{\mathrm{A}}\right)}{1+\exp\left(L_{i,b}^{\mathrm{A}}\right)} \prod_{b:x_{i,b}=-1} \frac{1}{1+\exp\left(L_{i,b}^{\mathrm{A}}\right)}$$

followed by computation of $\mathrm{P}[\mathbf{s}'=\mathbf{s}] = \prod_{i=1}^{M_{\mathrm{T}}} \mathrm{P}[s_i'=s_i]$. For the sake of simplicity of exposition, we write $\mathrm{P}[s]$ instead of $\mathrm{P}[s'=s]$ in the remainder of this thesis.

Iterative decoding is based on feeding back *extrinsic* a posteriori LLR-values [40], denoted by $L^{\mathrm{E}}_{i,b}$ $(\forall i, b)$, instead of the *intrinsic* ones computed in (2.23). Note that a particular intrinsic LLR-value $L_{i,b}$ has been computed on the basis of the received vector and *all* a priori LLRs $L^{\mathrm{A}}_{i,b}$ $(\forall i, b)$, i.e., information of all LLR-values $L^{\mathrm{A}}_{i,b}$ is contained in $L_{i,b}$. In order to prevent that "old" a priori information is contained in the new a posteriori output, only "new information" needs to be fed back. An extrinsic a posteriori LLR-value only contains a priori information from $L^{A}_{j,l}$ (where $j \neq i$ and $l \neq b$).[10] The common approach to compute extrinsic LLRs is to first compute intrinsic LLR according to (2.22) or (2.23), followed by subtracting the corresponding a priori LLR-value [19, 40]

$$L^{\mathrm{E}}_{i,b} = L_{i,b} - L^{\mathrm{A}}_{i,b}, \quad \forall i, b. \tag{2.27}$$

Exact APP detection in iterative MIMO systems corresponds to computing (2.23) followed by (2.27) for each LLR-value. However, straightforward evaluation of (2.23) requires computation and summation of $|\mathcal{O}|^{M_{\mathrm{T}}}$ terms, leading to (often) prohibitive computational complexity. In order to reduce this complexity, several algorithms to approximate (2.23) with low computational complexity have been proposed in the literature. One low-complexity approach is described below. More sophisticated SISO detection algorithms are treated in Chapter 3 and Chapter 4.

Linear Soft-Output MMSE Detection

One of the best-known low-complexity approach to approximate (2.23) is referred to as linear soft-output MMSE detection [68,69]. The main idea underlying this algorithm is to decompose the exact APP LLR computation in M_{T} single-input single-output detection problems and to assume that these detection problems can be solved independently. This approach is able to significantly lower the computational complexity compared to that of (2.23) and it leads to an acceptable (error-rate) performance.

The result of the effective channel after (unbiased) MMSE equal-

[10] We refer to [41] for more details on extrinsic and intrinsic LLRs.

ization (2.10) is a sufficient statistic and hence, (2.22) is equal to

$$L_{i,b} = \log\left(\frac{\mathrm{P}[x_{i,b}=+1\,|\,\hat{\mathbf{y}},\mathbf{H}]}{\mathrm{P}[x_{i,b}=-1\,|\,\hat{\mathbf{y}},\mathbf{H}]}\right) \tag{2.28}$$

where we set $\hat{\mathbf{y}} = \hat{\mathbf{y}}^{\mathrm{MMSE}} = \mathbf{s} + \tilde{\mathbf{n}}$ of (2.10) for the sake of simplicity of exposition. The assumption that $\hat{y}_i = s_i + \tilde{n}_i$ is statistically independent of $x_{j,b}$ (for $j \neq i$, $\forall b$), i.e., the layers i and $j \neq i$ are assumed to be statistically independent, leads to the approximation

$$L_{i,b} \approx \log\left(\frac{\mathrm{P}[x_{i,b}=+1\,|\,\hat{y}_i,\mathbf{H}]}{\mathrm{P}[x_{i,b}=-1\,|\,\hat{y}_i,\mathbf{H}]}\right). \tag{2.29}$$

Bayes's theorem applied to the probabilities in (2.29) yields

$$\mathrm{P}[x_{i,b}=x\,|\,\hat{y}_i,\mathbf{H}] = \frac{1}{\mathrm{p}(\hat{y}_i)}\sum_{a\in\mathcal{Z}_b^{(x)}}\mathrm{p}(\hat{y}_i\,|\,s_i=a,\mathbf{H})\,\mathrm{P}[s_i=a] \tag{2.30}$$

for $x \in \{+1,-1\}$. The sets $\mathcal{Z}_b^{(+1)}$, $\mathcal{Z}_b^{(-1)}$ in (2.30) denote the subsets of $\mathcal{O}$, where the bth bit corresponds to $+1$ and -1, respectively. Since the exact PDF of $\mathrm{p}(\hat{y}_i\,|\,s_i=a,\mathbf{H})$ is difficult to treat analytically, the (MMSE-equalized) symbols $\hat{y}_i$ are assumed to be complex Gaussian distributed according to [68, 69]

$$\mathrm{p_G}(\hat{y}_i\,|\,s_i=a) = \frac{1}{\pi\tilde{\nu}_i^2}\exp\left(-\frac{|\hat{y}_i-a|^2}{\tilde{\nu}_i^2}\right). \tag{2.31}$$

with $\mathbb{E}[\hat{y}_i] = a$ and $\mathrm{Var}[\hat{y}_i] = \mathbb{E}\left[|\tilde{n}_i|^2\right] = \tilde{\nu}_i^2$. The intrinsic LLRs in (2.29) can be approximated further using the Gaussian assumption in (2.31) leading to

$$\begin{aligned} L_{i,b} \approx &\log\left(\sum_{a\in\mathcal{Z}_b^{(+1)}}\exp\left(-\frac{|\hat{y}_i-a|^2}{\tilde{\nu}_i^2}-\log\mathrm{P}[s_i=a]\right)\right) \\ &-\log\left(\sum_{a\in\mathcal{Z}_b^{(-1)}}\exp\left(-\frac{|\hat{y}_i-a|^2}{\tilde{\nu}_i^2}-\log\mathrm{P}[s_i=a]\right)\right). \end{aligned} \tag{2.32}$$

It is important to note that (2.32) only requires to evaluate $|\mathcal{O}|$ terms per LLR-value, which significantly reduces the computational complexity compared to that of exact APP computation in (2.23). However, the resulting LLRs are no longer optimal and hence, linear soft-output MMSE detection entails a performance loss compared to exact APP detection. Corresponding performance results are given below.

Performance Comparison

In order to characterize the performance of the described SISO detection algorithms, numerical simulations are performed. Unless explicitly stated otherwise, all simulation results presented in the remainder of this thesis are for a convolutionally encoded (rate 1/2, generator polynomials $[133_o\ 171_o]$, and constraint length 7) iterative MIMO-OFDM system (as depicted in Figure 2.1) using $M_{\mathrm{T}} = M_{\mathrm{R}} = 4$, 16-QAM symbol constellation with Gray labeling as defined in IEEE 802.11n [2], 64 OFDM tones, and a TGn type C channel model [12]. SISO channel decoding is performed using the (sum-product) Bahl, Cocke, Jelinek, and Raviv (BCJR) algorithm [34]. One frame consists of 1024 randomly interleaved (across space and frequency) bits corresponding to one (spatial) OFDM symbol. The number of iterations I corresponds to using the soft-input soft-output MIMO detector (and the SISO channel decoder) once. All error-rate performance simulations are averaged over 640'000 channel realizations. For each channel realization a single noise realization has been generated.

Outage Lower-Bound The performance of iterative MIMO detection using large block-lengths and i.i.d. Rayleigh-fading channels has been compared to the ergodic channel capacity in [19]. Since most practical systems employ coding over relatively small block-lengths (e.g., over one OFDM symbol) and real-world channels usually show correlation across OFDM tones and spatial streams, the outage capacity characterizes the underlying performance limits more accurately. We define the ε-outage capacity $C_{\mathrm{out},\epsilon}$ as [9, 70]

$$\mathrm{P}[I\left(\mathsf{SNR},\mathcal{H}\right) < C_{\mathrm{out},\varepsilon}] = \varepsilon \tag{2.33}$$

where $\mathcal{H} = \{\mathbf{H}[1], \ldots, \mathbf{H}[N]\}$ contains the $M_{\mathrm{R}} \times M_{\mathrm{T}}$ channel matrices for the $N = 64$ OFDM tones and [71]

$$I(\mathsf{SNR}, \mathcal{H}) = \frac{1}{N} \sum_{\ell=1}^{N} \log_2 \det \left(\mathbf{I}_{M_{\mathrm{R}}} + \frac{\mathsf{SNR}}{M_{\mathrm{T}}} \mathbf{H}^H[\ell] \mathbf{H}[\ell] \right).$$

The FER of the system under consideration is lower-bounded by the outage probability in (2.33) according to [72]

$$\mathrm{P}[I(\mathsf{SNR}, \mathcal{H}) < RM_{\mathrm{T}}Q] \leq \mathrm{FER}(\mathsf{SNR}) \tag{2.34}$$

where $RM_{\mathrm{T}}Q$ is the information rate per OFDM tone. Note that the outage capacity in (2.33) depends on the joint distribution of $\mathcal{H}$, i.e., it is determined by the statistics of the channel model. We emphasize that analytical expressions for (2.34) are often difficult to obtain and hence, simulations are used to compute the outage lower-bound (OLB) in (2.34).

Numerical Performance Results Figure 2.3 compares the performance of hard-output ML detection, linear soft-output MMSE detection, and exact APP detection. Furthermore, the OLB in (2.34) is shown as a reference. This simulation shows that hard-output ML detection realizes similar performance compared to that of linear soft-output MMSE detection (for $I = 1$). We emphasize that exact soft-output APP detection (i.e., $I = 1$) attains more than 3 dB SNR performance improvement compared to that of hard-output ML detection (at FER=1 %). Linear soft-output MMSE detection does not seem to be suited for iterative MIMO detection, i.e., no significant performance improvements can be observed by increasing the number of iterations. In contrast, increasing the number of iterations (to $I = 4$) for the exact APP detector yields a performance gain of more than 7.5 dB SNR (at 1 % FER) compared to hard-output ML performance. Note that exact APP detection is able to approach the OLB by approximately 2.5 dB SNR, which demonstrates that near outage-capacity can be achieved with iterative MIMO detection.

From this simulation, we draw the following conclusions:

- MIMO detection in coded systems attains significantly better performance than that of uncoded systems (cf. Figure 2.2), since

the channel code is able to further mitigate the impact of fading and the additive Gaussian noise.

- Surprisingly, the performance of hard-output ML detection is similar to that of linear soft-output MMSE detection in this scenario. Hence, low-complexity soft-output MIMO detection can outperform hard-output schemes that require high computational complexity. In coded systems, it can therefore be more important to compute soft-outputs with a low-complexity MIMO detection scheme, instead of using a (computationally complex) hard-output detector that achieves full diversity. Note that optimum performance is still achieved by a MIMO detector that computes high-quality soft-outputs and achieves full diversity, such as exact APP.[11]

- Linear soft-output MMSE detection is one of the most promising solutions for low-complexity detection in coded MIMO systems [47]. However, this algorithm is *not* suited for iterative MIMO decoding.

We conclude that iterative MIMO decoding is able to substantially improve the performance of coded MIMO systems and is even able to approach the OLB up to a few dB. However, optimal soft-input soft-output MIMO detection according to (2.23) entails prohibitive computational complexity.

In the next two chapters, we focus on soft-input soft-output MIMO detection schemes that require low computational complexity, but still are able to achieve near-optimal SISO detection performance in iterative MIMO systems.

[11] These observations are in accordance with the results of [13]; therein it was shown that linear soft-output MMSE detection is able to outperform hard-output ML detection (in terms of information transfer characteristics) for low-rate codes.

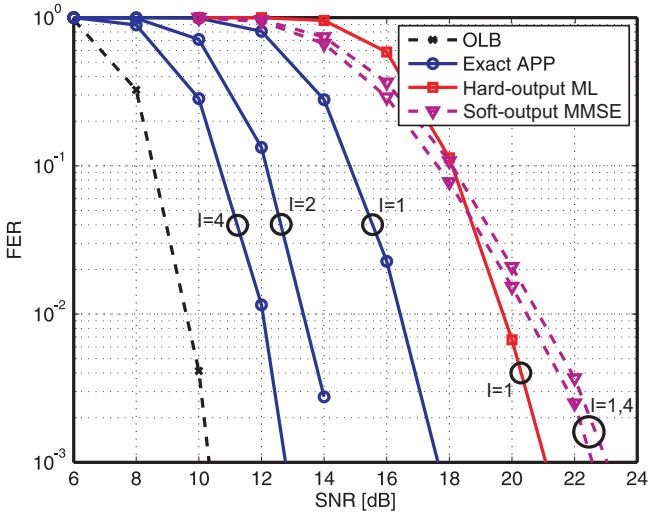


Figure 2.3: Numerical performance results of hard-output ML detection, linear soft-output MMSE detection, and exact APP detection compared to the outage lower-bound (OLB).

Chapter 3

SISO MMSE Parallel Interference Cancellation

The soft-input soft-output MIMO detection algorithm described in this chapter is based on the algorithm developed by Wang and Poor in 1999 [20]. In Section 3.2, we review the SISO MMSE parallel interference cancellation (PIC) algorithm and employ various optimizations on algorithmic level in order to obtain low computational complexity. In Section 3.3, we describe a corresponding VLSI architecture. Performance results of the resulting ASIC implementation in 90 nm complementary metal-oxide semiconductor (CMOS) technology prove that the optimized SISO MMSE PIC detection algorithm enables high-performance iterative MIMO decoding in practical systems.

3.1 Algorithm

The SISO MMSE PIC detection algorithm has initially been developed by Wang and Poor in 1999 [20] for iterative decoding in MU-CDMA systems using binary phase-shift keying (BPSK) constellations. In 2002, Tüchler, Singer, and Kötter [42] extended the SISO

MMSE PIC algorithm to more general phase-shift keying (PSK) symbol alphabets and proposed first approaches to reduce the computational complexity. In the same year, additional methods to reduce complexity or to improve the performance of the algorithm have been proposed by Dejonghe and Vandendorpe [73] and Witzke *et al.* [43]. In 2006, Tomasoni *et al.* [74] proposed further methods for complexity reduction. The first FPGA implementation of a SISO MMSE PIC detector was proposed in 2008 by Boher *et al.* [75], which, however, employs a *sub-optimal* MMSE filter in combination with STBCs to achieve satisfying performance.

In the remainder of this section we review the SISO MMSE PIC algorithm as proposed in [20, 42], i.e., without employing methods for complexity reduction.

3.1.1 The SISO MMSE PIC Algorithm

The main idea underlying the SISO MMSE PIC algorithm is to compute estimates of the transmitted symbols based on the a priori LLRs obtained from the SISO channel decoder. These estimates are used to cancel interference (caused by the MIMO channel) in the received vector. The remaining noise-plus-(self)-interference (NPI) term is then equalized using a MMSE filter, followed by computation of per-stream a posteriori LLRs. As detailed in the following, the SISO MMSE PIC algorithm performs soft-input soft-output MIMO detection in five steps.

Computation of Soft-Symbols

In the first step, estimates of the transmitted symbols—referred to as "soft-symbols" in the following—are computed with the aid of the a priori information obtained from the SISO channel decoder. The soft-symbols $\hat{s}_i$ $(i = 1, \ldots, M_{\mathrm{T}})$ are computed according to [20, 42]

$$\hat{s}_i = \mathbb{E}[s_i] = \sum_{a \in \mathcal{O}} \mathrm{P}[s_i = a]\, a \tag{3.1}$$

where expectation in (3.1) is on the basis of a priori information and $\mathrm{P}[s_i = a]$ corresponds to the a priori probability of the symbol $a \in \mathcal{O}$.

The error between the transmitted symbol s_i and the soft-symbol $\hat{s}_i$ is defined as

$$e_i = s_i - \hat{s}_i. \tag{3.2}$$

The reliability of each soft-symbol $\hat{s}_i$ is given by the variance of the error in (3.2), i.e.,

$$E_i = \mathrm{Var}[s_i] = \mathbb{E}\left[|e_i|^2\right]. \tag{3.3}$$

The a priori probabilities involved in the computation of the soft-symbols (3.1) and their variances (3.3) are calculated from the a priori LLRs (2.24) delivered by the channel decoder. From (2.25) and (2.26) follows that the probability of the transmitted bit $x_{i,b}$ corresponds to [41]

$$\mathrm{P}[x_{i,b} = x] = \frac{\exp\left(\frac{1}{2} x L^{\mathrm{A}}_{i,b}\right)}{\exp\left(+\frac{1}{2} L^{\mathrm{A}}_{i,b}\right) + \exp\left(-\frac{1}{2} L^{\mathrm{A}}_{i,b}\right)} \tag{3.4}$$

for $x = \{+1, -1\}$. Since we employ BICM, the bits resulting from the channel code are independent among spatial streams i and bits b. Hence, the a priori probabilities in (3.1) can be computed as

$$\mathrm{P}[s_i = a] = \prod_{b=1}^{Q} \mathrm{P}\left[x_{i,b} = [a]_b\right], \quad \forall i \tag{3.5}$$

using (3.4) and $[a]_b$ refers to the bth bit of the symbol $a \in \mathcal{O}$ (see (2.1)).

Parallel Interference Cancellation

The second step in the SISO MMSE PIC algorithm amounts to cancel interference in the received vector $\mathbf{y}$ (see Eq. 2.2) with the aid of the soft-symbols (3.1). To this end, the PIC process considers the ith stream and cancels interference of all other streams (i.e., $j \neq i$) using the results of (3.1); this leads to the ith interference-canceled received vector

$$\hat{\mathbf{y}}_i \triangleq \mathbf{y} - \sum_{j \neq i} \mathbf{h}_j \hat{s}_j = \mathbf{h}_i s_i + \tilde{\mathbf{n}}, \quad \forall i \tag{3.6}$$

where $\sum_{j\neq i}$ in (3.6) refers to summation over $j = 1, \ldots, M_{\mathrm{T}}$ for which $j \neq i$. The noise-plus-(remaining)-interference (NPI) vector $\tilde{\mathbf{n}}$ in (3.6) corresponds to

$$\tilde{\mathbf{n}} = \sum_{j\neq i} \mathbf{h}_j e_j + \mathbf{n}. \tag{3.7}$$

It is important to note that PIC in (3.6) has been used to transform the MIMO input-output relation (2.2) into a SIMO system with a NPI-vector that is no longer circularly symmetric complex Gaussian distributed.

MMSE Equalization

The third step amounts to suppressing the NPI (3.7) in the interference-canceled vector (3.6) using a MMSE filter. Similarly to (2.8), the MMSE filter vectors are computed according to

$$\tilde{\mathbf{w}}_i^H \triangleq \underset{\tilde{\mathbf{w}}^H \in \mathbb{C}^{1\times M_{\mathrm{T}}}}{\arg\min}\ \mathbb{E}\Big[\big|\tilde{\mathbf{w}}^H \hat{\mathbf{y}}_i - s_i\big|^2\Big], \quad i = 1, \ldots, M_{\mathrm{T}} \tag{3.8}$$

which leads to the MMSE filter vectors $\tilde{\mathbf{w}}_i^H$ minimizing the mean-square error (MSE) between the *filtered* (and interference-canceled) vector $\hat{\mathbf{y}}_i$ and the transmitted symbol on the ith stream. As derived in Appendix A.1, the MMSE filter vectors satisfying (3.8) correspond to [42]

$$\tilde{\mathbf{w}}_i^H = E_{\mathrm{s}} \mathbf{h}_i^H \Big(\mathbf{H}\widetilde{\mathbf{\Lambda}}_i \mathbf{H}^H + N_{\mathrm{o}} \mathbf{I}_{M_{\mathrm{R}}}\Big)^{-1} \tag{3.9}$$

with the real-valued $M_{\mathrm{T}} \times M_{\mathrm{T}}$ diagonal matrix $\widetilde{\mathbf{\Lambda}}_i$ having entries

$$\tilde{\Lambda}_{j,j} = \begin{cases} E_j, & j \neq i \\ E_s, & j = i \end{cases}$$

where the variances E_j are defined in (3.3). We emphasize that computation of (3.9) requires a $M_{\mathrm{R}} \times M_{\mathrm{R}}$-dimensional matrix inversion and needs to be carried out for each stream (i.e., M_{T} times), for each received vector, and each iteration. In order to substantially reduce this computational burden, we propose a low-complexity method to compute the MMSE filter vectors $\tilde{\mathbf{w}}_i^H$ in (3.9) in Section 3.2.

MMSE Filtering

The MMSE filter vectors in (3.9) are used to equalize the NPI vector from the corresponding interference-canceled received vectors in (3.6). The ith result of this filtering process is

$$\tilde{z}_i \triangleq \tilde{\mathbf{w}}_i^H \hat{\mathbf{y}}_i = \tilde{\mathbf{w}}_i^H \mathbf{h}_i s_i + \tilde{\mathbf{w}}_i^H \tilde{\mathbf{n}} \tag{3.10}$$

which consists of an (equalized) signal part and a filtered NPI part. In order to compute a posteriori LLRs from (3.10), the noise vector $\tilde{\mathbf{w}}_i^H \tilde{\mathbf{n}}$ is assumed to be Gaussian distributed with mean

$$\mathbb{E}[\tilde{z}_i] = \tilde{\mathbf{w}}_i^H \mathbf{h}_i s_i = \tilde{\mu}_i s_i \tag{3.11}$$

and variance (of the NPI term in (3.7)) according to

$$\tilde{\nu}_i^2 = \mathrm{Var}[z_i] = \tilde{\mathbf{w}}_i^H \left(\sum_{j \neq i} E_j \mathbf{h}_j \mathbf{h}_j^H + N_\mathrm{o} \mathbf{I}_{M_\mathrm{R}} \right) \tilde{\mathbf{w}}_i \tag{3.12}$$

i.e., we have $\tilde{z}_i \sim \mathcal{CN}\big(\tilde{\mu}_i s_i, \tilde{\nu}_i^2\big)$ for $i = 1, \ldots, M_\mathrm{T}$.

A Posteriori LLR Computation

With PIC, the MIMO input-output relation (2.2) has been transformed into M_T single-input single-output systems (cf. (3.10)). By assuming that these M_T single-stream systems are independent, we can approximate the intrinsic a posteriori LLRs in (2.22) as follows

$$L_{i,b} \approx \log \left(\frac{\mathrm{P}[x_{i,b} = +1 | \tilde{z}_i]}{\mathrm{P}[x_{i,b} = -1 | \tilde{z}_i]} \right), \quad \forall i, b \tag{3.13}$$

using $\tilde{z}_i$ in (3.10). Bayes's rule applied to the conditional probabilities $\mathrm{P}[x_{i,b} = \pm 1 \,|\, \tilde{z}_i]$ in (3.13) leads to

$$\mathrm{P}[x_{i,b} = x \,|\, \tilde{z}_i] = \frac{1}{\mathrm{p}(\tilde{z}_i)} \sum_{a \in \mathcal{Z}_b^{(x)}} \mathrm{p}(\tilde{z}_i \,|\, s_i = a)\, \mathrm{P}[s_i = a] \tag{3.14}$$

where $\mathcal{Z}_b^{(+1)}$ and $\mathcal{Z}_b^{(-1)}$ refer to the subsets of $\mathcal{O}$, where the bth bit corresponds to $+1$ and -1, respectively (see (2.30)). Inserting (3.14)

into the approximated intrinsic a posteriori LLRs in (3.13) yields

$$L_{i,b} \approx \log\left(\frac{\sum_{a\in\mathcal{Z}_b^{(+1)}} \mathrm{p}(\tilde{z}_i \,|\, s_i = a)\,\mathrm{P}[s_i = a]}{\sum_{a\in\mathcal{Z}_b^{(-1)}} \mathrm{p}(\tilde{z}_i \,|\, s_i = a)\,\mathrm{P}[s_i = a_i]}\right). \tag{3.15}$$

In order to simplify (3.15), the PDF $\mathrm{p}(\tilde{z}_i \,|\, s_i = a)$ is approximated (analogous to (2.31)) by the Gaussian PDF of $\tilde{z}_i$ given the transmit symbol $a \in \mathcal{O}$, i.e.

$$\mathrm{p_G}(\tilde{z}_i \,|\, s_i = a) = \frac{1}{\pi\tilde{\nu}_i^2} \exp\left(-\frac{|\tilde{z}_i - \tilde{\mu}_i a|^2}{\tilde{\nu}_i^2}\right) \tag{3.16}$$

where the mean $\tilde{\mu}_i$ and variance $\tilde{\nu}_i$ are defined in (3.11) and (3.12), respectively. Using the results of (3.4) yields

$$\mathrm{P}[s_i = a_i] = \prod_{b=1}^{Q} \mathrm{P}[x_{i,b} = [a_i]_b] = \frac{1}{c_i} \prod_{b=1}^{Q} \exp\left(\frac{1}{2}[a_i]_b L_{i,b}^{\mathrm{A}}\right)$$

with $c_i = \prod_{b=1}^{Q} \left(\exp(+\frac{1}{2}L_{i,b}^{\mathrm{A}}) + \exp(-\frac{1}{2}L_{i,b}^{\mathrm{A}})\right)$. The Gaussian approximation in (3.16) enables to write the (approximated) intrinsic a posteriori LLRs in (3.15) as [42]

$$\begin{aligned}\tilde{L}_{i,b}^{\mathrm{D}} \triangleq \log&\left(\sum_{a\in\mathcal{Z}_b^{(+1)}} \exp\left(-\frac{|\tilde{z}_i - \tilde{\mu}_i a|^2}{\tilde{\nu}_i^2} + \sum_{b=1}^{Q} \frac{[a]_b}{2} L_{i,b}^{\mathrm{A}}\right)\right) \\ &- \log\left(\sum_{a\in\mathcal{Z}_b^{(-1)}} \exp\left(-\frac{|\tilde{z}_i - \tilde{\mu}_i a|^2}{\tilde{\nu}_i^2} + \sum_{b=1}^{Q} \frac{[a]_b}{2} L_{i,b}^{\mathrm{A}}\right)\right).\end{aligned} \tag{3.17}$$

The *extrinsic* a posteriori LLRs of the SISO MMSE PIC are finally obtained by computing $\tilde{L}_{i,b}^{\mathrm{E}} = \tilde{L}_{i,b}^{\mathrm{D}} - L_{i,b}^{\mathrm{A}}$ as done in (2.27).

It is important to note that the estimates produced by the MMSE filter vectors in (3.9) are *biased*. However, since we account for the fact that $\tilde{\mathbf{w}}_i^H \mathbf{h}_i$ is not necessarily equal to one, i.e., we use $\tilde{\mu}_i s_i = \tilde{\mathbf{w}}_i^H \mathbf{h}_i s_i$ in the LLR computation (3.17) instead of s_i, the LLRs computed by the SISO MMSE PIC algorithm are obtained through an unbiased MMSE estimation process.

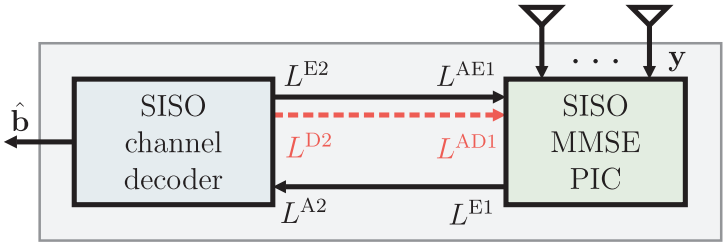


Figure 3.1: Iterative MIMO decoder using the SISO MMSE PIC detector and a channel decoder that provides extrinsic L^{E2} and intrinsic L^{D2} (corresponding to the dashed line) a posteriori LLRs [43].

3.1.2 Simulation Results

In order to characterize the performance of the SISO MMSE PIC algorithm described above, we perform simulations using a slightly extended (compared to that shown in Figure 2.1) iterative MIMO decoder as depicted in Figure 3.1. Note that this decoder structure contains extrinsic and (additionally) intrinsic LLRs on the feedback path from the SISO channel decoder to the SISO MMSE PIC detector. The reason for this additional feedback path will be clarified below.

Intrinsic vs. Extrinsic A Priori Input Witzke *et al.* [43] realized that using *intrinsic* a priori LLRs (denoted by L^{AD1}) for soft-symbol computation leads to a noticeable performance improvement compared to using extrinsic information instead. Extrinsic a priori LLRs (denoted by L^{AE1}) are only used in (3.17).

Figure 3.2 shows the performance of iterative MIMO decoding using the SISO MMSE PIC algorithm and confirms the observation made in [43]. It can be seen that for a larger number of iterations I, the SNR gap between intrinsic and extrinsic LLR feedback for soft-symbol computation increases. The method described in [43] enables a performance improvement (compared to the first iteration) of more than 7.5 dB SNR (measured at 1% FER), if performing four iterations. For a large number of iterations, the performance starts to saturate and hence, using more than four iterations does not yield significant improvements.

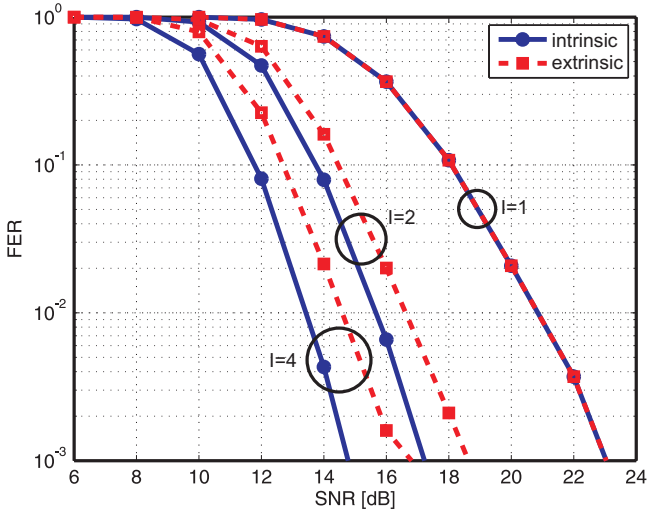


Figure 3.2: FER comparison of intrinsic and extrinsic a priori LLRs for soft-symbol computation in the SISO MMSE PIC algorithm.

We additionally see that iterative MIMO decoding based on the SISO MMSE PIC enables to obtain significant performance gains compared to that of soft-output-only MIMO detection. If using the SISO MMSE PIC algorithm, we will only consider intrinsic a priori input for slicing and extrinsic a priori input for the computation of (3.17) in the remainder of this thesis.

Comparison with Exact APP Figure 3.3 compares the performance of the SISO MMSE PIC algorithm to exact APP MIMO detection as described in Section 2.2.2. For a given number of iterations, exact APP performance *always* outperforms the SISO MMSE PIC algorithm. However, since i) exact APP detection is —due to the exceedingly high computational complexity— not suitable for practical applications and ii) both detector algorithms attain similar performance for a large number of iterations, SISO MMSE PIC is one of the most promising algorithms for low-complexity and high-performance soft-input soft-output MIMO detection.

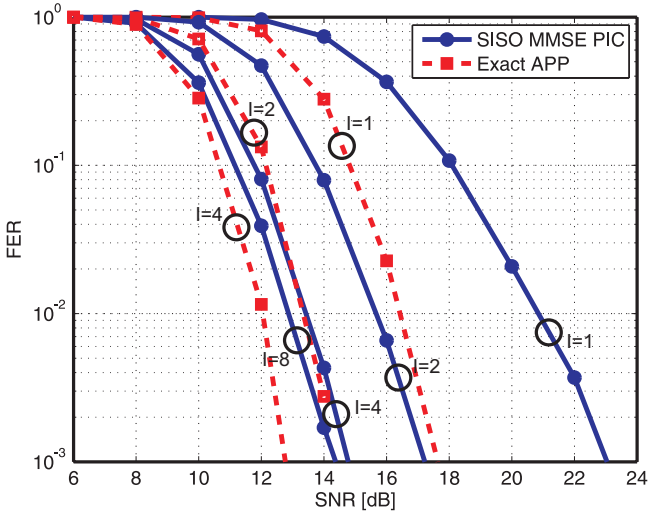


Figure 3.3: Numerical performance results of the SISO MMSE PIC algorithm compared to exact APP.

3.2 Algorithmic Optimizations

Straightforward implementation of the SISO MMSE PIC algorithm as described above entails high computational complexity and hence, is not well-suited for implementation in practical systems. The complexity of the algorithm is mainly dominated by the need for multiple matrix inversions per symbol vector and per iteration (required for computation of the MMSE filter vectors). This fact is further aggravated by the presence of exponential functions (required for computation of the soft-symbols and a posteriori LLRs), which leads to an exceedingly large dynamic range and hence, inhibits efficient fixed-point implementation.

In this section, we deploy a variety of techniques to reduce the computational complexity of the algorithm, which enables practical implementation without (noticeably) degrading the error-rate performance. The reference VLSI implementation developed in Section 3.3 proves that the resulting low-complexity variant of the SISO MMSE

PIC algorithm can indeed be implemented efficiently in practical systems.

3.2.1 Reduction of Algorithmic Complexity

In the following discussion, we describe several techniques to reduce complexity on algorithmic level. Some of these techniques rest on the assumption that Gray-mapping is used, while other methods can be applied for general mappings. In particular, we describe a novel approach to reduce the number of matrix inversions, i.e., only one matrix inversion is required for computation of all M_{T} MMSE filter vectors, while maintaining the performance of the SISO MMSE PIC algorithm as described in Section 3.1.1. For the sake of simplicity of exposition, the MIMO channel matrix is assumed to be normalized according to $(M_{\mathrm{T}}E_{\mathrm{s}})^{-\frac{1}{2}}\mathbf{H}$ in the discussion below.

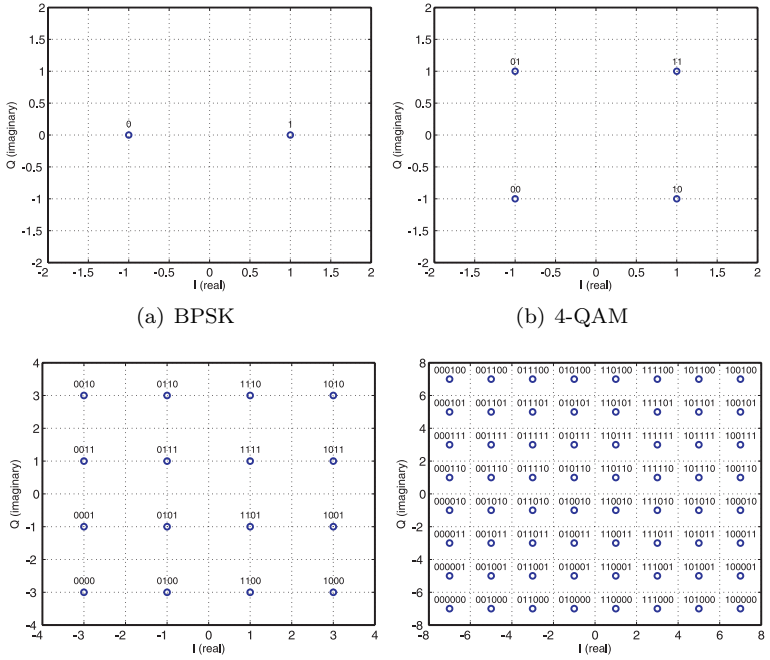
Efficient Soft-Symbol and Variance Computation

Consider the constellation sets and the Gray-mappings depicted in Figure 3.4. The real and imaginary part of each constellation symbol are mapped independently, i.e., some bits are used *only* for mapping of the real part and some only for the imaginary part. Note that this mapping is used in, e.g., the IEEE 802.11n standard [2], and that it can be exploited to substantially reduce the computational complexity associated with computation of the soft-symbols and their variances.

Efficient Soft-Symbol Computation We start by rewriting the bit-probability $\mathrm{P}[x_{i,b} = [s_i]_b]$ in (3.4) as

$$\begin{aligned}\mathrm{P}[x_{i,b} = [s_i]_b] &= \frac{1}{2}\left(1 + x\tanh\left(\frac{1}{2}L^{\mathrm{A}}_{i,b}\right)\right)\\ &= \frac{1}{2}\left(1 + [s_i]_b\,\mathrm{sign}\big(L^{\mathrm{A}}_{i,b}\big)\tanh\left(\frac{1}{2}\left|L^{\mathrm{A}}_{i,b}\right|\right)\right) = p_{i,b}\end{aligned} \tag{3.18}$$

where the hyperbolic tangent function in (3.18) can efficiently be approximated in hardware by using a look-up table (LUT) for the positive part only. Computation of $p_{i,b}$, hence, requires low complexity as



(a) BPSK (b) 4-QAM

(c) 16-QAM (d) 64-QAM

Figure 3.4: Constellations with Gray-mapping as defined in [2].

it only requires a table look-up (in a small LUT, where only half the tanh-values need to be stored) and additions/subtractions.

We now rewrite the soft-symbols in (3.1) as

$$\hat{s}_i = a(s_i) + \sqrt{-1}b(s_i), \quad \forall i \tag{3.19}$$

where $\Re\{\hat{s}_i\} = a(s_i)$ and $\Im\{\hat{s}_i\} = b(s_i)$ are both real-valued functions of the transmitted symbol s_i. Tomasoni *et al.* [74] realized that the functions $a(s_i)$ and $b(s_i)$ can be computed efficiently in presence of Gray mappings that separate the real and imaginary part. Efficient computation of the soft-symbols in (3.19) is obtained by computing the functions[1] in Table 3.1 and using the probabilities in (3.18).

[1]Note that the functions shown in Table 3.1 are obtained by explicit computa-

Table 3.1: Low-complexity computation of soft-symbols according to [74] for the IEEE 802.11n [2] Gray mapping.

	$\Re\{\hat{s}_i\} = a(s_i)$
BPSK	$1 - 2p_{i,1}$
4-QAM	$1 - 2p_{i,1}$
16-QAM	$(1 - 2p_{i,1})(1 + 2p_{i,2})$
64-QAM	$(1 - 2p_{i,1})(4p_{i,2}p_{i,3} + 2p_{i,2} - 2p_{i,3} + 3)$

	$\Im\{\hat{s}_i\} = b(s_i)$
BPSK	0
4-QAM	$1 - 2p_{i,2}$
16-QAM	$(1 - 2p_{i,3})(1 + 2p_{i,4})$
64-QAM	$(1 - 2p_{i,4})(4p_{i,5}p_{i,6} + 2p_{i,5} - 2p_{i,6} + 3)$

Efficient Variance Computation The variances of the soft-symbol can be computed in a similar fashion. We start by rewriting (3.3) as

$$E_i = \mathbb{E}\Big[|s_i|^2\Big] - \big|\mathbb{E}[s_i]\big|^2 = \mathbb{E}\Big[|s_i|^2\Big] - |\hat{s}_i|^2, \quad \forall i \tag{3.20}$$

where $|\hat{s}_i|^2 = a^2(s_i) + b^2(s_i)$ can be computed from the soft-symbol $\hat{s}_i$. Similarly to (3.19), the remaining term in (3.20) can be decomposed into

$$\mathbb{E}\Big[|s_i|^2\Big] = c(s_i) + d(s_i) \tag{3.21}$$

where $c(s_i)$ and $d(s_i)$ only depend on the real and imaginary part of s_i, respectively [74]. Both functions in (3.21) can efficiently be computed (in a similar manner as it has been done above) using the results provided in Table 3.2. We emphasize that efficient soft-symbol computation according to (3.19) and variance computation as shown in (3.20) provide *exact* results (i.e., no approximations are involved).

---

tion of the expectation operation in (3.1) followed by algebraic simplifications.

Table 3.2: Low-complexity variance computation according to [74] for the IEEE 802.11n [2] Gray mapping.

	$c(s_i)$
BPSK	1
4-QAM	2
16-QAM	$1 + 8p_{i,2}$
64-QAM	$(32p_{i,2}p_{i,3} + 16p_{i,2} - 8p_{i,3} + 9)$

	$d(s_i)$
BPSK	0
4-QAM	0
16-QAM	$1 + 8p_{i,4}$
64-QAM	$(32p_{i,5}p_{i,6} + 16p_{i,5} - 8p_{i,6} + 9)$

Low-Complexity MMSE Filter Computation

Computation of the M_{T} MMSE filter vectors in (3.9) requires M_{T} matrix inversions. The matrices to be inverted are of dimension $M_{\mathrm{R}} \times M_{\mathrm{R}}$, which is sub-optimal (in terms of computational complexity) for systems employing more receive than transmit antennas. We emphasize that matrix inversion is a computational complex task, posing significant challenges for VLSI implementation (see, e.g., [15]). Several approaches have been proposed in the literature in order to reduce the complexity associated with the MMSE filter vector computation in (3.9). Tüchler *et al.*, for example, proposed an approach based on rank-1 updates in order to avoid the recalculation of (full) matrix inversions [42]. Tomasoni *et al.* [74] described an approach that requires to compute M_{T} matrices, but only one row of the M_{T} inverses needs to be computed for each MMSE filter vector. However, both methods still require prohibitively high computational complexity in practice.

In order to significantly reduce the computational complexity associated with matrix inversion, we propose a novel approach that only requires to compute *one* matrix inversion (of dimension $M_{\mathrm{T}} \times M_{\mathrm{T}}$) to compute *all* M_{T} MMSE filter vectors at once. We emphasize that the proposed approach yields the *same results* as if using the MMSE

filter vectors described in (3.9). The derivation of our low-complexity approach is detailed in Appendix A.2 and amounts to computing

$$\mathbf{W}^H = \left(\mathbf{H}^H\mathbf{H}\mathbf{\Lambda} + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}}\right)^{-1}\mathbf{H}^H \tag{3.22}$$

where $\mathbf{\Lambda}$ is a $M_\mathrm{T} \times M_\mathrm{T}$ diagonal matrix with $\Lambda_{i,i} = E_i$ from (3.3) for $i = 1, \ldots, M_\mathrm{T}$. The rows of $\mathbf{W}^H$ correspond to the M_T MMSE filter vectors, i.e.,

$$\mathbf{W}^H = \left[\, \mathbf{w}_1 \ \cdots \ \mathbf{w}_{M_\mathrm{T}} \,\right]^H.$$

It is important to note that the MMSE filter vectors $\mathbf{w}_i^H$ $(\forall i)$ obtained through computation of (3.22) correspond to scaled (by a real-valued constant) versions of $\tilde{\mathbf{w}}_i^H$ given in (3.9). We emphasize that constant factors do *not* affect the resulting a posteriori LLRs. A detailed proof for this property is given in Appendix A.2.1. Computation of $\mathbf{W}^H$ according to (3.22) can be performed in a numerical stable way (corresponding results are shown in Section 3.2.2) and hence, is suitable for hardware implementation.

The low-complexity MMSE filter in (3.22) allows additional insight to the performance of the SISO MMSE PIC algorithm:

- Consider the case where $L_{i,b}^\mathrm{A} = 0$ $(\forall i, b)$. In this case, all soft-estimates are equal to zero and their variances are equal to $E_i = \mathbb{E}\left[|s_i|^2\right] = E_\mathrm{s}$. Hence, the resulting MMSE filter vectors correspond to

  $$\mathbf{W}^H = \left(\mathbf{H}^H\mathbf{H}E_\mathrm{s} + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}}\right)^{-1}\mathbf{H}^H$$

  which is a scaled version of the MMSE estimator matrix given in (2.9), i.e., $\mathbf{W}^H = E_\mathrm{s}\mathbf{M}$. Hence, if no a priori information is available (e.g., for $I = 1$), the SISO MMSE PIC algorithm performs the same computations as the linear soft-output MMSE detector and hence, both detectors achieve the same performance.

- If perfect a priori information is available, i.e., if $L_{i,b}^\mathrm{A} = x_{i,b} \cdot \infty$ $(\forall i, b)$, then $E_i = 0$ $(\forall i)$; additionally, the soft-symbols correspond to the transmitted symbols (i.e., $\hat{s}_i = s_i$, $\forall i$). Hence, the

result of the PIC process is

$$\hat{\mathbf{y}}_i = \mathbf{h}_i s_i + \mathbf{n}, \quad \forall i$$

which corresponds to a SIMO system with M_{R} receive antennas and $\mathbf{n} \sim \mathcal{CN}\left(0, N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{R}}}\right)$. Since the filter vectors correspond to scaled versions of the matched filter, i.e., $N_{\mathrm{o}}\mathbf{W}^H = \mathbf{H}^H$, the SISO MMSE PIC algorithm performs optimal detection in this case.

We conclude that the SISO MMSE PIC algorithm attains a performance ranging from soft-output MMSE detection to optimum soft-output MIMO detection [42]; this behavior can also be seen in the information transfer characteristic simulation shown in Section 4.6.6.

Efficient A Posteriori LLR Computation

In the following paragraphs, we combine several methods for complexity reduction in the a posteriori LLR computation using the results of [73, 74, 76].

Efficient Distance-Term Calculation To reduce the complexity of a posteriori LLR computation, we start by defining an unbiased version of the MMSE filter in (3.10) such that

$$z_i = \frac{\tilde{z}_i}{\tilde{\mu}_i} = \frac{\mathbf{w}_i^H \hat{\mathbf{y}}_i}{\mathbf{w}_i^H \mathbf{h}_i}, \quad \forall i \tag{3.23}$$

which enables to rewrite (3.17) as follows:

$$\begin{aligned}\tilde{L}_{i,b}^{\mathrm{D}} = &\log\left(\sum_{a \in \mathcal{Z}_b^{(+1)}} \exp\left(-\frac{\mu_i^2 \left|z_i - a\right|^2}{\nu_i^2} + \sum_{b=1}^{Q} \frac{[a]_b}{2} L_{i,b}^{\mathrm{A}}\right)\right) \\ &- \log\left(\sum_{a \in \mathcal{Z}_b^{(-1)}} \exp\left(-\frac{\mu_i^2 \left|z_i - a\right|^2}{\nu_i^2} + \sum_{b=1}^{Q} \frac{[a]_b}{2} L_{i,b}^{\mathrm{A}}\right)\right)\end{aligned} \tag{3.24}$$

where μ_i and ν_i^2 result from (3.11) and (3.12), respectively. Note that we are using the (low-complexity) MMSE filter vectors $\mathbf{w}_i^H$ instead of $\tilde{\mathbf{w}}_i^H$ in the sequel.

The result of the derivation shown in Appendix A.3 enables to compute the variances ν_i^2 $(\forall i)$ based on (3.12) in a more efficient manner, i.e.,

$$\nu_i^2 = \mathbf{w}_i^H \mathbf{h}_i - E_i\big(\mathbf{w}_i^H \mathbf{h}_i\big)^2$$

and, with $\mu_i = \mathbf{w}_i^H \mathbf{h}_i$, we can state the post-equalization signal-to-noise-and-interference ratio (SINR) on the ith stream as

$$\rho_i = \frac{\mu_i^2}{\nu_i^2} = \frac{\big(\mathbf{w}_i^H \mathbf{h}_i\big)^2}{\mathbf{w}_i^H \mathbf{h}_i - E_i\big(\mathbf{w}_i^H \mathbf{h}_i\big)^2} = \frac{\mu_i}{1 - E_i \mu_i}. \tag{3.25}$$

It is important to note that the post-equalization SINR obtained in (3.25) can be used to simplify the a posteriori LLR computation given in (3.24) to

$$\begin{aligned}\tilde{L}_{i,b}^{\mathrm{D}} = &\log\left(\sum_{a \in \mathcal{Z}_b^{(+1)}} \exp\left(-\rho_i \left|z_i - a\right|^2 + \sum_{b=1}^{Q} \frac{[a]_b}{2} L_{i,b}^{\mathrm{A}}\right)\right) \\ &- \log\left(\sum_{a \in \mathcal{Z}_b^{(-1)}} \exp\left(-\rho_i \left|z_i - a\right|^2 + \sum_{b=1}^{Q} \frac{[a]_b}{2} L_{i,b}^{\mathrm{A}}\right)\right)\end{aligned} \tag{3.26}$$

which, with $z_i = \mu_i^{-1} \mathbf{w}_i^H \hat{\mathbf{y}}_i$ from (3.23) and (3.25), requires significantly less computational complexity compared to the initial formulation given in (3.17), while leading to the same performance.

Max-Log Approximation So far, all techniques to reduce the computational complexity of the SISO MMSE PIC algorithm did not have any influence on the error-rate performance. Now, we apply the max-log approximation [77]

$$\log\left(\sum_i \exp(d_i)\right) \approx \max_i \{d_i\}. \tag{3.27}$$

which additionally reduces computational complexity with a generally negligible performance loss. In particular, application of the max-log

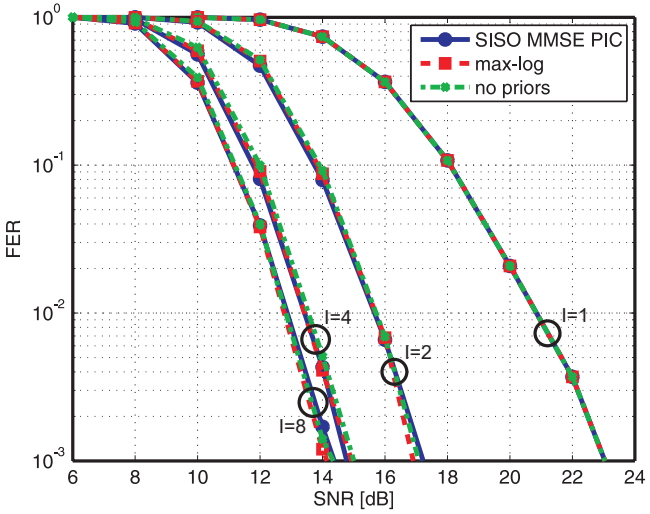


Figure 3.5: Impact on performance of the max-log approximation and of neglecting the priors in the a posteriori LLR computation.

approximation (3.27) to the a posteriori LLRs in (3.26) helps to avoid computation of exponential functions, i.e.,

$$\tilde{L}^{\mathrm{D}}_{i,b} \approx \min_{a \in \mathcal{Z}_b^{(-1)}} \left\{ \rho_i \left| z_i - a \right|^2 - \sum_{b=1}^{Q} \frac{[a]_b}{2} L^{\mathrm{A}}_{i,b} \right\} - \min_{a \in \mathcal{Z}_b^{(+1)}} \left\{ \rho_i \left| z_i - a \right|^2 - \sum_{b=1}^{Q} \frac{[a]_b}{2} L^{\mathrm{A}}_{i,b} \right\} \tag{3.28}$$

which renders it suitable for hardware implementation. We emphasize that using the max-log approximation in (3.28) entails a performance loss (except if using BPSK constellations). However, the associated performance loss is negligible if Gray-mapping is used (corresponding simulation results are shown in Figure 3.5).

Avoiding Priors during LLR Computation It is important to realize that (3.28) yields strong similarities to SISO detection in single-

antenna systems. As it has been demonstrated in [78] using extrinsic information transfer (EXIT) chart analysis, the improvement in terms of mutual information at the output of a single-antenna SISO detector is low if Gray mapping is used. Moreover, it was shown in [79] that neglecting the priors in (3.28) does *not* entail a performance loss for BPSK and 4-QAM constellations for Gray-mapping. Hence, we conclude that, in general, computation of (3.28) without using the priors does not lead to a severe degradation in terms of error-rate performance; this claim is supported by the simulation results shown in Figure 3.5. The resulting max-log a posteriori LLRs (3.28) correspond to

$$\tilde{L}_{i,b}^{\mathrm{E}} \triangleq \rho_i \left(\min_{a \in \mathcal{Z}_b^{(-1)}} |z_i - a|^2 - \min_{a \in \mathcal{Z}_b^{(+1)}} |z_i - a|^2 \right). \tag{3.29}$$

Note that we defined the result of (3.29) as an *extrinsic* LLR. The reason for this definition is to explicitly state that computation according to (2.27) is no longer required (i.e., the $\tilde{L}_{i,b}^{\mathrm{E}}$ $(\forall i, b)$ are directly fed to the channel decoder). In addition, priors do *not directly* affect the a posteriori LLR computation. It is, however, important to note that the priors still influence computation of soft-symbols and their variances. We emphasize that (3.29) does no longer require to feed back extrinsic a priori LLRs to the SISO MMSE PIC; this leads to an additional simplification of the iterative MIMO decoder structure depicted in Figure 3.1.

Low-Complexity LLR Computation Neglecting the priors in the LLR computation enables to use an elegant low-complexity method developed by Collings *et al.* [76] for computation of (3.29). This method has been applied to a soft-output MMSE detector implementation in [47] and it was shown to significantly lower the computational complexity associated with LLR computation (if a Gray-mapping is used). The key idea developed in [76] is to rewrite the max-log LLRs in (3.29) as $\tilde{L}_{i,b}^{\mathrm{E}} = \rho_i \lambda_b(z_i)$ with

$$\lambda_b(z_i) = \min_{a \in \mathcal{Z}_b^{(-1)}} |z_i - a|^2 - \min_{a \in \mathcal{Z}_b^{(+1)}} |z_i - a|^2$$

which can be computed efficiently in hardware by using the results from Table 3.3 for the IEEE 802.11n-mapping [2]. For a detailed derivation of the results in Table 3.3, we refer to [47, 76, 79].

Preprocessing for SISO MMSE PIC

Tomasoni *et al.* [74] described a method that reduces the amount of recurrent operations in the SISO MMSE PIC algorithm. This technique is referred to as preprocessing[2] and it amounts to computing—prior to detection—the Gram matrix of the MIMO channel matrix

$$\mathbf{G} = [\,\mathbf{g}_1 \;\cdots\; \mathbf{g}_{M_\mathrm{T}}\,] = \mathbf{H}^H\mathbf{H}$$

and the matched filter (MF) output

$$\mathbf{y}^\mathrm{MF} = \mathbf{H}^H\mathbf{y}.$$

In the remainder of this paragraph, we show how our improved version of the SISO MMSE PIC algorithm can be performed only based on $\mathbf{G}$, $\mathbf{y}^\mathrm{MF}$, the noise variance N_o, and the priors $L_{i,b}^\mathrm{A}$ $(\forall i, b)$.

Instead of performing PIC using the received symbol (3.6), parallel interference cancellation is now performed on the basis of the MF output according to

$$\hat{\mathbf{y}}_i^\mathrm{MF} \triangleq \mathbf{y}^\mathrm{MF} - \sum_{j\neq i} \mathbf{g}_j \hat{s}_j. \tag{3.30}$$

Preprocessing does no longer require computation of the MMSE filter matrix in (3.22); instead, a partial MMSE filter matrix needs to be computed

$$\mathbf{A}^{-1} = (\mathbf{G}\mathbf{\Lambda} + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}})^{-1} \tag{3.31}$$

where $\left(\mathbf{A}^{-1}\right)^H = [\,\mathbf{a}_1 \;\cdots\; \mathbf{a}_{M_\mathrm{T}}\,]^H$, i.e., the vector $\mathbf{a}_i$ corresponds to the ith column of $\mathbf{A}^{-1}$. Using the result of the MF PIC process in (3.30) and the partial MMSE filter matrix (3.31), the unbiased MMSE estimate in (3.23) can efficiently be computed according to

$$z_i = \frac{\mathbf{a}_i^H \hat{\mathbf{y}}_i^\mathrm{MF}}{\mu_i}, \quad i = 1, \ldots, M_\mathrm{T} \tag{3.32}$$

[2]This preprocessing technique should *not* be confused with tasks that only need to be performed once per frame.

Table 3.3: Low-complexity max-log LLR computation for IEEE 802.11n Gray-mapping according to [76].

	b	$\lambda_b(z_i)$	Range
BPSK	1	$4\Re\{z_i\}$	$\forall\Re\{z_i\}$
4-QAM	1	$4\Re\{z_i\}$	$\forall\Re\{z_i\}$
	2	$4\Im\{z_i\}$	$\forall\Im\{z_i\}$
16-QAM	1	$4\Re\{z_i\}$	$\lvert\Re\{z_i\}\rvert \leq 2$
		$8\Re\{z_i\} - 8\text{sign}(\Re\{z_i\})$	$\lvert\Re\{z_i\}\rvert \geq 2$
	2	$8 - 4\lvert\Re\{z_i\}\rvert$	$\forall\Re\{z_i\}$
	3	$4\Im\{z_i\}$	$\lvert\Im\{z_i\}\rvert \leq 2$
		$8\Im\{z_i\} - 8\text{sign}(\Im\{z_i\})$	$\lvert\Im\{z_i\}\rvert \geq 2$
	4	$8 - 4\lvert\Im\{z_i\}\rvert$	$\forall\Im\{z_i\}$
64-QAM	1	$4\Re\{z_i\}$	$\lvert\Re\{z_i\}\rvert \leq 2$
		$8\Re\{z_i\} - 8\text{sign}(\Re\{z_i\})$	$2 \geq \lvert\Re\{z_i\}\rvert \leq 4$
		$12\Re\{z_i\} - 24\text{sign}(\Re\{z_i\})$	$4 \geq \lvert\Re\{z_i\}\rvert \leq 6$
		$16\Re\{z_i\} - 48\text{sign}(\Re\{z_i\})$	$\lvert\Re\{z_i\}\rvert \geq 6$
	2	$24 - 8\lvert\Re\{z_i\}\rvert$	$\lvert\Re\{z_i\}\rvert \leq 2$
		$16 - 4\lvert\Re\{z_i\}\rvert$	$2 \geq \lvert\Re\{z_i\}\rvert \leq 6$
		$40 - 8\lvert\Re\{z_i\}\rvert$	$\lvert\Re\{z_i\}\rvert \geq 6$
	3	$4\lvert\Re\{z_i\}\rvert - 8$	$\lvert\Re\{z_i\}\rvert \leq 4$
		$24 - 4\lvert\Re\{z_i\}\rvert$	$\lvert\Re\{z_i\}\rvert \geq 4$
	4	$4\Im\{z_i\}$	$\lvert\Im\{z_i\}\rvert \leq 2$
		$8\Im\{z_i\} - 8\text{sign}(\Im\{z_i\})$	$2 \geq \lvert\Im\{z_i\}\rvert \leq 4$
		$12\Im\{z_i\} - 24\text{sign}(\Im\{z_i\})$	$4 \geq \lvert\Im\{z_i\}\rvert \leq 6$
		$16\Im\{z_i\} - 48\text{sign}(\Im\{z_i\})$	$\lvert\Im\{z_i\}\rvert \geq 6$
	5	$24 - 8\lvert\Im\{z_i\}\rvert$	$\lvert\Im\{z_i\}\rvert \leq 2$
		$16 - 4\lvert\Im\{z_i\}\rvert$	$2 \geq \lvert\Im\{z_i\}\rvert \leq 6$
		$40 - 8\lvert\Im\{z_i\}\rvert$	$\lvert\Im\{z_i\}\rvert \geq 6$
	6	$4\lvert\Im\{z_i\}\rvert - 8$	$\lvert\Im\{z_i\}\rvert \leq 4$
		$24 - 4\lvert\Im\{z_i\}\rvert$	$\lvert\Im\{z_i\}\rvert \geq 4$

where $\mu_i = \mathbf{a}_i^H \mathbf{g}_i$ $(\forall i)$. The post-equalization SINR terms ρ_i $(\forall i)$ can be computed from μ_i through (3.25). Finally, the a posteriori LLRs of the SISO MMSE PIC can be computed according to (3.29). It is important to note that preprocessing essentially avoids costly re-computation of scalar products (which involve additions and multiplications) and hence, the complexity of a corresponding VLSI implementation can be reduced (see Section 3.3).

3.2.2 Efficient Matrix Inversion

The main computational burden of the low-complexity SISO MMSE PIC algorithm developed in the last two sections, still corresponds to the complexity of the required matrix-inversion task. In order to design a hardware-efficient VLSI implementation of the detector, a suitable matrix-inversion algorithm needs to be found. In this section, we briefly evaluate potential candidates for efficient matrix inversion and we identify the LU-decomposition (LUD)-based approach to be the most promising one.

Comparison of Existing Matrix-Inversion Algorithms

In the last few years, a variety of matrix inversion algorithms and corresponding VLSI implementations for linear MIMO detection have been proposed, e.g., [80–83]. We emphasize that the matrix to be inverted for linear (soft-output) MMSE detection (2.9) is Hermitian (or self-adjoint), which enables to reduce the computational complexity by exploiting the structure of the matrix. However, the matrix to be inverted (see (3.31)) is, in general, not Hermitian. Hence, most of the available efficient matrix-inversion algorithms cannot be applied to our problem.[3]

Another approach is QRD-based matrix inversion, e.g., [81]. The main advantage of this method is that the QR-decomposition itself can be performed with high numerical stability, which renders this approach suitable for fixed-point implementation [84]. Unfortunately, the QRD requires, in general, a larger number of arithmetic operations compared to other inversion schemes (see [15, 85]). In addition, SISO

[3]Inversion algorithms that are able exploit Hermitian structure are, e.g., the Ricatti recursion [15, 80] or divide-and-conquer inversion [82, 83].

Algorithm 1 In-place LU-decomposition [87]

for $k = 1, \ldots, M_{\mathrm{T}} - 1$ **do**
  $r_k = 1/A_{k,k}$
  for $i = k + 1, \ldots, M_{\mathrm{T}}$ **do**
    $A_{i,k} = r_k A_{i,k}$
  end for
  for $i = k + 1, \ldots, M_{\mathrm{T}}$ **do**
    for $j = k + 1, \ldots, M_{\mathrm{T}}$ **do**
      $A_{i,j} = A_{i,j} - A_{i,k} A_{k,j}$
    end for
  end for
end for
$r_{M_{\mathrm{T}}} = 1/A_{M_{\mathrm{T}}, M_{\mathrm{T}}}$

MMSE PIC requires explicit computation of the inverse $\mathbf{A}^{-1}$ in (3.31). Hence, an additional back-substitution procedure is needed. In summary, QRD-based matrix inversion does not seem to be suitable in terms of computational complexity for our application.

Matrix inversion based on the singular value decomposition (SVD) has the potential to reduce the amount of recurrent operations if only the priors change (and hence, $\mathbf{\Lambda}$). However, computation of a SVD itself requires very high complexity in hardware (see [86]). The total complexity (i.e., SVD combined with matrix inversion) is therefore significantly higher than that of conventional matrix inversion algorithms. Hence, SVD-based matrix inversion is not considered in the sequel.

As it has been noted in [15] and [85], LUD-based matrix inversion exhibits —among all popular matrix inversion algorithms— the lowest number of arithmetic operations. Furthermore, this method does *not* require that the matrix to be inverted is Hermitian. These two advantages (combined with the fact that no VLSI implementation for MIMO detection of a LUD-based matrix inversion has been reported in the open literature) led to the decision of using this matrix-inversion method in the SISO MMSE PIC detector architecture described in the next section.

LUD-based Matrix Inversion

The matrix inversion algorithm for computation of the partial MMSE filter matrix in (3.31) used in the following is based on the LUD algorithm described in [87]. We first compute the LUD

$$\mathbf{A} = \mathbf{L}\mathbf{U} \tag{3.33}$$

where $\mathbf{A}$ is given in (3.31), $\mathbf{L}$ is a $M_\mathrm{T} \times M_\mathrm{T}$-dimensional complex-valued lower-triangular matrix with $L_{i,i} = 1$ $(i = 1, \ldots, M_\mathrm{T})$ and the complex-valued matrix $\mathbf{U}$ is of dimension $M_\mathrm{T} \times M_\mathrm{T}$ and upper-triangular. To perform the decomposition in (3.33), we use the in-place LUD algorithm described in [87], which is summarized in Algorithm 1. The resulting matrix $\mathbf{A}'$ contains $\mathbf{L}$ (without its main diagonal) and $\mathbf{U}$ such that

$$\mathbf{A}' = \begin{bmatrix} U_{1,1} & U_{1,1} & \cdots & U_{1,M_\mathrm{T}} \\ L_{2,1} & U_{2,2} & \cdots & U_{2,M_\mathrm{T}} \\ \vdots & \vdots & \ddots & \vdots \\ L_{M_\mathrm{T},1} & L_{M_\mathrm{T},2} & \cdots & U_{M_\mathrm{T},M_\mathrm{T}} \end{bmatrix}.$$

Note that M_T reciprocal operations are required and the resulting values $1/A_{i,i}$ $(\forall i)$ in Algorithm 1 (line 2) are stored in a separate M_T-dimensional vector $\mathbf{r}$ (for reasons that will be explained below). Note that the LUD can, in general, be efficiently computed in hardware, except for the reciprocals $r_i = 1/A_{i,i}$ $(i = 1, \ldots, M_\mathrm{T})$. We refer to Section 3.3.1 for corresponding implementation details.

Straightforward matrix inversion based on the LUD amounts to a separate inversion of $\mathbf{L}$ and $\mathbf{U}$, followed by a matrix multiplication, i.e., computation of $\mathbf{A}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$. It is important to note that this approach is not very efficient in terms of the involved number of arithmetic operations [79]. In order to attain a more efficient matrix-inversion procedure, we employ forward and backward substitution for computation on $\mathbf{A}^{-1}$. To this end, we write

$$\mathbf{A}\mathbf{X} = \mathbf{L}\mathbf{U}\mathbf{X} = \mathbf{I}_{M_\mathrm{T}} \tag{3.34}$$

where $\mathbf{X} = [\,\mathbf{x}_1 \ \cdots \ \mathbf{x}_{M_\mathrm{T}}\,] = \mathbf{A}^{-1}$ and $\mathbf{I}_{M_\mathrm{T}} = [\,\mathbf{e}_1 \ \cdots \ \mathbf{e}_{M_\mathrm{T}}\,]$, where $\mathbf{e}_i$ denotes the ith unit vector. The main idea underlying our LUD-based

matrix inversion approach corresponds to determine the vectors $\mathbf{x}_i$ ($\forall i$) in two steps. The first step amounts to computing the vectors $\mathbf{v}_i$ according to

$$\mathbf{L}\mathbf{v}_i = \mathbf{e}_i, \quad i = 1, \ldots, M_{\mathrm{T}} \tag{3.35}$$

which can be done efficiently using forward substitution [87]. Since $L_{i,i} = 1$ ($\forall i$) and $\mathbf{L}^{-1} = [\,\mathbf{v}_1 \;\cdots\; \mathbf{v}_{M_{\mathrm{T}}}\,]$, forward-substitution (3.35) enables to compute the inverse of $\mathbf{L}$ solely using multiplications and additions (i.e., no division operation is required). The second step of inversion corresponds to solve

$$\mathbf{U}\mathbf{x}_i = \mathbf{v}_i, \quad i = 1, \ldots, M_{\mathrm{T}} \tag{3.36}$$

through back-substitution. The resulting vectors $\mathbf{x}_i$ ($\forall i$) correspond to the columns of the inverse $\mathbf{A}^{-1}$ in (3.34). Since the diagonal entries of $\mathbf{U}$ are, in general, not equal to one, the back-substitution in (3.36) requires division operations [87]. However, as the reciprocal values of $U_{i,i}^{-1} = r_i$ ($\forall i$) in Algorithm 1 have been computed and stored previously (i.e., during the LUD), back-substitution can be performed with multiplications and additions only. This approach ultimately leads to a hardware-friendly and low-complexity matrix-inversion solution for the low-complexity SISO MMSE PIC algorithm.

3.3 VLSI Implementation

This section describes the first VLSI architecture for the SISO MMSE PIC algorithm for iterative MIMO decoding together with corresponding reference ASIC implementation results.

3.3.1 VLSI Architecture

The SISO MMSE PIC algorithm described in the previous sections requires matrix inversion at symbol-rate and per iteration, which is challenging for practical implementation. To this end, various techniques on architectural level have been applied to obtain a hardware-efficient VLSI implementation. The following paragraphs describe the VLSI architecture and provide details for the most critical (with respect to throughput and numerical precision) computation units.

Architectural Overview

Figure 3.6 shows the high-level VLSI architecture of the SISO MMSE PIC detector. The architectural principle is referred to as *systolic network of dedicated processing units (PUs)*. The main idea underlying the proposed VLSI architecture is to decompose the SISO MMSE PIC algorithm into eight PUs. Each PU performs one (or part of a) task of the reduced-complexity SISO MMSE PIC algorithm:

- The preprocessing unit computes the output of the matched-filter and the Gram matrix as described in Section 3.2.1.
- The unit "soft-symbol, variance, $\mathbf{A}$" computes the soft-symbols and their variances as described in Section 3.2.1. In addition, the matrix $\mathbf{A} = \mathbf{G}\mathbf{\Lambda} + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{T}}}$ in (3.31) is computed.
- The two PUs, namely PIC 1 and PIC 2, perform PIC based on the MF-vector $\mathbf{y}^{\mathrm{MF}}$ as described in Section 3.2.1.
- The unit labeled by "LUD and forward" performs the LUD of the matrix $\mathbf{A}$ followed by forward substitution (i.e., computation of $\mathbf{L}^{-1}$) as described in Section 3.2.2.
- The backward-substitution PU performs the remaining steps of matrix inversion as described in Section 3.2.2.
- The MMSE filtering unit computes the unbiased MMSE estimates z_i according to (3.32) and the post-equalization SINRs ρ_i for $\forall i$ (see Section 3.2.1).
- The LLR computation unit (LCU) computes the a posteriori LLRs $\tilde{L}_{i,b}^{\mathrm{E}}$ $(\forall i, b)$ as described in Section 3.2.1.

We emphasize that using this architectural principle enables to design, optimize, and verify each of the PUs separately.

Architecture of Processing Units

As mentioned above, computation of the SISO MMSE PIC algorithm is performed in the eight PUs. All PUs are designed on the basis of the same processor-like architecture, which is depicted in Figure 3.7.

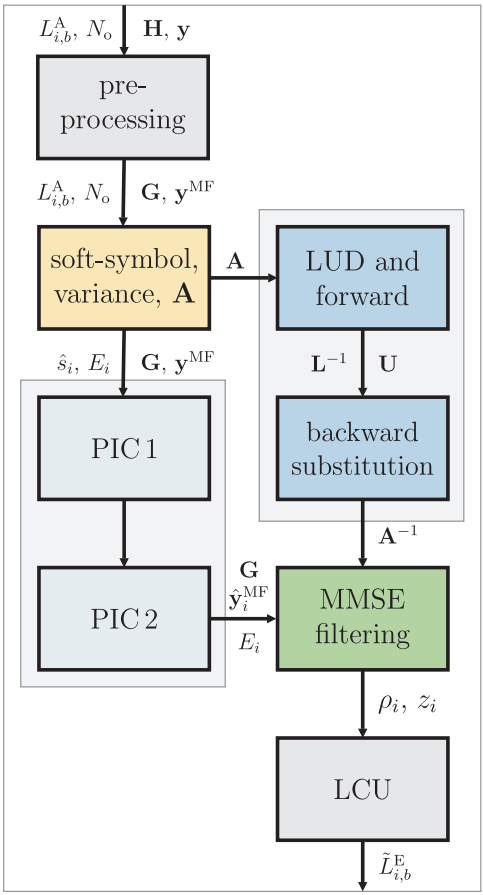


Figure 3.6: VLSI architecture of the SISO MMSE PIC detector.

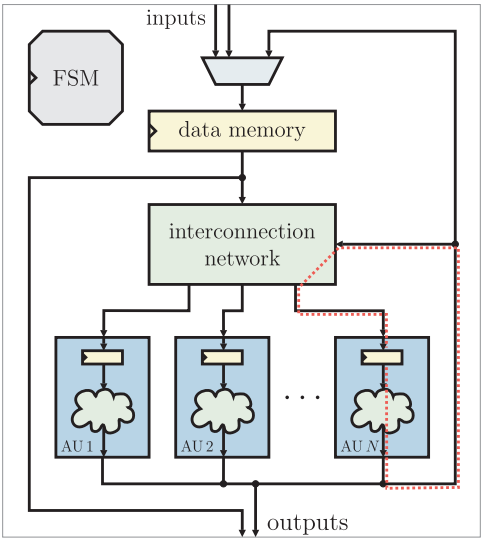


Figure 3.7: Architectural principle underlying each PU.

Each PU consists of several arithmetic units (AUs), which perform all required computations in a time-shared fashion.[4]

Each PE requires 18 clock cycles to complete its computation (reasons for this number are given below). In the 18th clock cycle, data is passed to the next PU and new data is obtained from the previous PU, resembling to the operational principle of a systolic array [14]. During the 18th clock cycle, some data elements are computed and directly passed to the next PU. Since 18 clock cycles are required to load a new channel matrix, a new received vector, a new set of a priori LLRs, and a noise variance into the SISO MMSE PIC architecture, the throughput (in terms of LLRs per second) of the architecture corresponds to

$$\Theta = \frac{M_{\mathrm{T}} Q}{18} f_{\mathrm{clk}} \tag{3.37}$$

[4]Note that the PEs are not reprogrammable or configurable and that they contain a hard-coded finite-state machine (FSM).

where f_{clk} is the clock frequency of the implementation. The latency of this architecture corresponds to $6 \cdot 18 = 108$ clock cycles.

Arithmetic Units Each PU contains one or more AU. Similar to a processor-based architecture, the AUs perform basic arithmetic operations, e.g., complex-valued multiplication, addition/subtraction, arithmetic right/left shifts, table lookups, and reciprocal computation. The AU architectures depend on the task to be computed. The critical path of the SISO MMSE PIC architecture is through an AU that contains a complex-valued multiplier and the interconnection network (see Figure 3.7). The maximum clock frequency of the SISO MMSE PIC is, therefore, mainly determined by a complex-valued multiplier.

Data Memories Data, such as intermediate values, vectors, and matrices, are stored in the data memory of each PUs, i.e., no central storage unit is employed. For high throughput, a large memory bandwidth is required. Since the access to the data elements in the memories has random characteristics, the data memory is built from flip-flops instead of using on-chip static random access memory (SRAM) macro-cells. Flip-flop arrays require a larger area (compared to that of on-chip SRAMs), but enable highly parallel and random data access, which is difficult to realize efficiently by using off-the-shelf SRAM macro-cell memories.

Improvements for Numerical Stability

Matrix inversion for MIMO detection requires high arithmetic precision, e.g., [15]. In this architecture, two techniques have been applied in order to improve the numerical stability of fixed-point matrix inversion. These techniques are described in the following paragraphs.

Limitation of the Noise Variance As it was shown in [15] for linear soft-output MMSE detection, it is essential to perform inversion based on a regularized matrix if fixed-point arithmetic is employed. For high SNR, the noise variance in $\mathbf{A} = (\mathbf{G}\mathbf{\Lambda} + N_{\text{o}}\mathbf{I}_{M_{\text{T}}})$ of (3.31) can get arbitrarily small, and N_{o} can no longer been represented using a finite number of bits. In this case, N_{o} might become zero and the

matrix to be inverted is no longer regularized; this causes problems with numerical stability of the inversion (especially if the matrix $\mathbf{G\Lambda}$ is ill-conditioned). In order to avoid the noise variance N_o being zero, its minimum value has been limited to 2^{-12}, i.e.,

$$\tilde{N}_\mathrm{o} = \max\left\{N_\mathrm{o}, 2^{-12}\right\} \tag{3.38}$$

is used instead of N_o in the computation of $\mathbf{A}$. The advantages of (3.38) are i) matrix inversion can be performed in a more stable manner at high SNRs and ii) the number of fraction bits to represent $\mathbf{A}$ does only need to be slightly larger that 12 bit. Note that limitation of N_o to 2^{-12} causes an error floor at approximately 36 dB SNR [15], which is sufficiently large for a rate-5/6 coded 4×4 MIMO system employing 64-QAM.

Scaling of the A-Matrix In order to additionally improve numerical stability of fixed-point matrix inversion, a quasi-floating-point approach has been used. To this end, the most critical fixed-point values (the ones with the largest dynamic range) have been associated with an exponent. The key idea is to undo the effect of this exponent at the latest possible stage in the sequence of computations. This technique enables to improve the precision of fixed-point calculations, while leading to a low hardware overhead (see [15, 84, 85]).

Since the entries of $\mathbf{\Lambda}$ can get close to zero[5], the entries of the inverse $\mathbf{A}^{-1}$ might become very large (especially in the high-SNR regime). In order to reduce the dynamic range of the inverse and the number of required bits in hardware, the matrix $\mathbf{A}$ in (3.31) is scaled in the "soft-symbol, variance, and $\mathbf{A}$" PU. To this end, the matrix $\mathbf{A}$ is right-multiplied by $\mathbf{\Gamma}$ according to

$$\widetilde{\mathbf{A}} = (\mathbf{G\Lambda} + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}})\mathbf{\Gamma} \tag{3.39}$$

where $\mathbf{\Gamma}$ is a real-valued $M_\mathrm{T} \times M_\mathrm{T}$ diagonal matrix with $\Gamma_{i,i} = 2^{-\gamma_i}$ for $i = 1, \ldots, M_\mathrm{T}$ and

$$\gamma_i = \left\lfloor \log_2\left(A_{i,i}\right) \right\rfloor + 1, \quad \forall i.$$

[5]This happens, for example, after a few iterations, i.e., where the magnitudes of the a priori LLRs get large (see the discussion in Section 3.2.1).

We note that the $A_{i,i} \in \mathbb{R}$ are non-negative. The scaling by $\Gamma_{i,i}$ ensures that the diagonal entries of the scaled matrix $\widetilde{\mathbf{A}}$ satisfy $0.5 \leq \tilde{A}_{i,i} < 1$ $(\forall i)$. Since $\gamma_i \in \mathbb{Z}$ $(\forall i)$, multiplication by $2^{-\gamma_i}$ and the $\lfloor \log_2(x) \rfloor$-function can efficiently be computed in hardware using arithmetic shifts and identification of the leading-one in the binary-valued representation of x, respectively. After the inversion of $\widetilde{\mathbf{A}}$, the effect of scaling in (3.39) is compensated by right-multiplication of the results by $\mathbf{\Gamma}$, i.e.,

$$\mathbf{\Gamma}\widetilde{\mathbf{A}}^{-1} = \mathbf{\Gamma}\mathbf{\Gamma}^{-1}(\mathbf{G}\mathbf{\Lambda} + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{T}}})^{-1} = \mathbf{A}^{-1}.$$

As mentioned above, it is beneficial (in terms of numerical stability) to undo the effect of scaling in (3.39) at the last possible stage of the SISO MMSE PIC algorithm. Hence, re-scaling (possibly followed by clipping) is performed during computation of the unbiased MMSE estimates (3.32) and during computation of the post-equalization SINRs in (3.25).

Real-Valued Reciprocal Unit

LU-decomposition, post-equalization SINR (3.25) computation, and MMSE filtering in (3.32) require division operations. In general, division is not well-suited for fixed-point hardware implementation. In our case, the real-valued reciprocal $r_k = 1/A_{k,k}$ in the LUD (see Algorithm 1, line 2) is the throughput bottleneck of the SISO MMSE PIC architecture, caused by data-dependencies. In particular, the result r_k is required immediately in the LUD-process (see Algorithm 1 on lines 3-5). To achieve a high throughput, a low number of clock cycles per reciprocal computation and a short critical path is of paramount importance.

Newton-Raphson Iteration For high numeric precision and high throughput, we compute the reciprocals (i.e., $1/x$) using a custom hardware unit, instead of off-the-shelf division circuitry. Division can then be performed using a multiplication with the reciprocal. Analogous to the quasi-floating point approach described above, we first scale input value $x \in \mathbb{R}$ such that

$$\tilde{x} = 2^{-\alpha}x, \quad \text{with} \quad \alpha = \lfloor \log_2(x) \rfloor + 1 \tag{3.40}$$

which ensures that $0.5 \leq \tilde{x} < 1$ and leads to improved numerical stability of the reciprocal-based division since $1 < 1/\tilde{x} \leq 2$. The reciprocal $1/\tilde{x}$ is then computed iteratively using the Newton-Raphson method [88]

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 1, \ldots, K \tag{3.41}$$

where $f(x_k)$ is a suitably chosen function that satisfies $f(1/\tilde{x}) = 0$; the initial value is denoted by x_1, $f'(x) = \frac{\mathrm{d}f(x)}{\mathrm{d}x}$, and K stands for the maximum number of iterations. Choosing $f(x_k) = \tilde{x} - 1/x_k$ in (3.41) leads to the iteration [89–92]

$$x_{k+1} = 2x_k - \tilde{x}x_k^2, \quad k = 1, \ldots, K \tag{3.42}$$

which converges[6] to $1/\tilde{x}$ if the initial value satisfies $0 < x_1 < 2/\tilde{x}$ [88]. We emphasize that the scaling in (3.40) can be compensated according to $1/x = (1/\tilde{x})2^{\alpha}$. Numerically stable division a/x is obtained in hardware by computing $1/\tilde{x}$ using (3.41) and performing the multiplication $a(1/\tilde{x})$, followed by re-scaling using the factor 2^{α}, which can efficiently be performed using arithmetic right- or left-shifts.

Architectures for Reciprocal Computation Various VLSI implementations for computation of reciprocals—ranging from high precision [90,91] to high throughput [92]—have been described in the literature. Almost all corresponding architectures consist of two parts: A look-up table (LUT), which generates the initial value of the iteration, and arithmetic circuitry that performs the Newton-Raphson iteration (3.42), e.g., [89]. In our case, the LUT considers the first b bit of $\tilde{x}$ in order to produce a fairly accurate initial estimate of $1/\tilde{x}$, i.e., $x_1 \approx 1/\tilde{x}$ satisfying $1 < x_1 < 2/\tilde{x}$. Then, a small number of Newton-Raphson iterations (3.42) are performed, in order to improve the accuracy of the result. Note that better approximations of $1/\tilde{x}$ require larger LUTs, but require fewer iterations to achieve a certain precision. Hence, for a given precision, there exists a tradeoff between area (influenced by the LUT) and throughput (depending on the number of iterations K).

[6] This fixed point iteration yields quadratic convergence to $1/\tilde{x}$ [89].

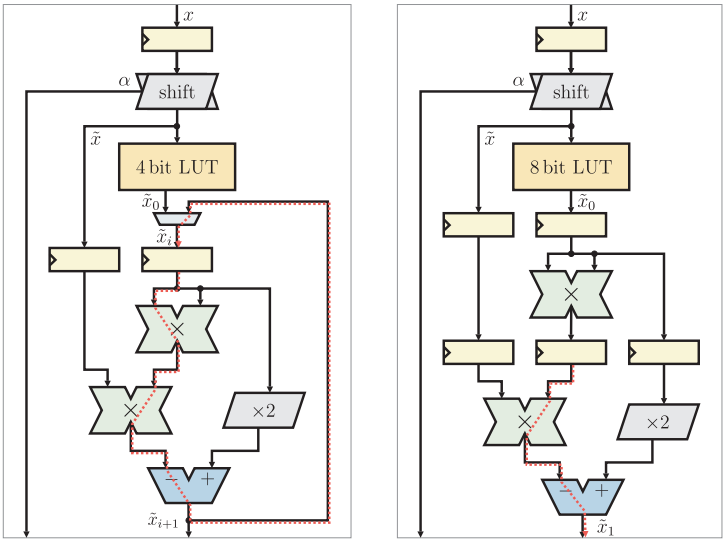


Figure 3.8: Two architectures for computation or reciprocals. Left: iterative architecture; right: pipelined architecture.

In order perform the LUD in 18 clock cycles, we decided to allow at most three clock cycles per reciprocal-value computation. Fixed-point simulation results have shown that 15 bit precision is sufficient for the results of the reciprocal unit. Hence, our reciprocal unit must be able to deliver reciprocals with 15 bit precision in three clock cycles. Figure 3.8 shows two different architectures that meet the given constraints. Note that both designs perform an initial shift according to (3.40) and use a LUT to compute the initial value of the iteration.

- The first architecture employs a 4 bit table look-up followed by two Newton-Raphson iterations and is referred to as the "iterative architecture." Within three clock cycles, two data items can be processed. The critical path passes through a squaring unit, one multiplier, one adder, and a multiplexer.

- The second architecture is pipelined and performs only one iteration, which requires a larger LUT to attain the required pre-

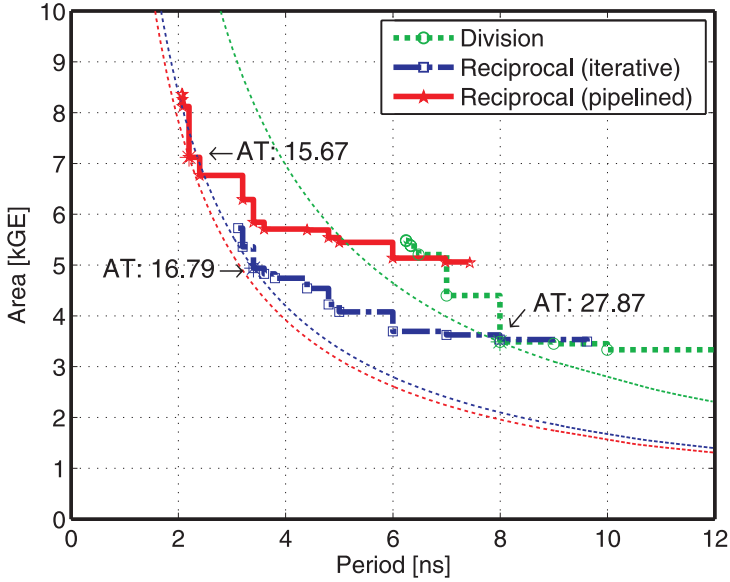


Figure 3.9: AT trade-off of division unit and two different reciprocal units in 130 nm (1P/8M) CMOS technology.

cision (i.e., we use the first 8 bit of $\tilde{x}$ to generate the initial estimate x_1). Note that the pipelined architecture is able to process one data item per clock cycle and has a shorter critical path than the iterative variant. Hence, the pipelined reciprocal unit achieves a significantly higher throughput (compared to the iterative architecture).

Comparison of Reciprocal Units Figure 3.9 compares synthesis results of the two reciprocal units described above with an off-the-shelf divider.[7] Note that the divider unit performs a full division, whereas the two reciprocal units require an additional multiplication to obtain the same functionality. Nevertheless, the two reciprocal units have a

[7] A sequential divider using three clock cycles to achieve 15 bit precision is used. In order to enable comparability of the results, the circuitry required for scaling (see Figure 3.8) has been added.

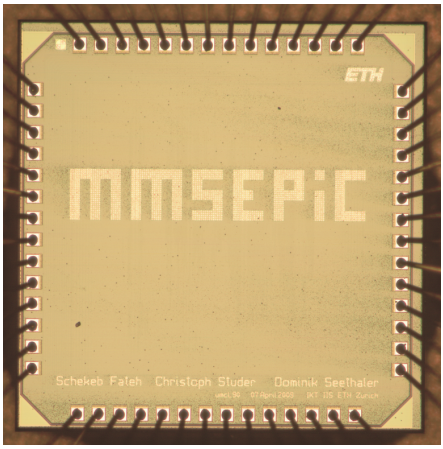


Figure 3.10: ASIC micrograph of the SISO MMSE PIC detector in 90 nm (1P/9M) CMOS technology.

much shorter critical path than the division unit. In addition, both reciprocal units are twice as efficient in terms of the area-delay (AT) product. The pipelined version attains a slightly higher maximum clock frequency than the iterative variant, but is slightly less efficient in terms of the AT-product. Since the key goal of our SISO MMSE PIC architecture was high throughput, we opted for the pipelined reciprocal unit, which offers a higher maximum clock frequency and processes one data item per clock cycle.

3.3.2 Implementation Results

Figure 3.10 shows the ASIC picture of the SISO MMSE PIC algorithm implementation in 90 nm (1P/9M) CMOS technology. The architecture has been designed to be compliant with the IEEE 802.11n [2] WLAN standard. The implementation supports MIMO systems with four spatial streams and BPSK, 4-QAM, 16-QAM, and 64-QAM constellation alphabets.

Table 3.4: Post-layout implementation results of the SISO MMSE PIC algorithm in 90 nm (1P/9M) CMOS technology.

Core area [mm^2]	1.29
Cell area [kGE[a]]	410
Maximum clock frequency [MHz]	620
Maximum throughput[b] [MLps]	826
Latency [ns]	174
Hardware-efficiency[c] [kGE/MLps/it]	0.50

[a]One GE corresponds to a two-input drive-one NAND gate of size 3.136 µm^2.
[b]The throughput is given by million LLRs per second [MLps].
[c]Hardware-efficiency is normalized to the number of iterations (it).

VLSI Implementation Results Reference (post-layout) implementation results of the SISO MMSE PIC implementation are given in Table 3.4. The decoder requires a total area of 410 kGE and achieves a maximum clock frequency of 620 MHz, which, according to (3.37), leads to a maximum throughput of 826 million LLRs per second. Note that IEEE 802.11n specifies a peak throughput of 600 Mbps for four-stream transmission, 64-QAM, and using a rate-5/6 code [2]. The proposed detector implementation attains $826 \cdot 5/6 = 688.3$ Mbps if using one iteration, which is sufficient to meet the requirements of the standard. In order to perform more than one iteration (i.e., to obtain gains through iterative MIMO decoding) either more instances of the SISO MMSE PIC are required or iterations can only be performed for modulation and coding schemes which require lower throughput (e.g., the ones in 20 MHz mode of IEEE 802.11n).

Area Breakdown A detailed area breakdown of the SISO MMSE PIC implementation is given in Table 3.5. Note that almost a third of the architecture is occupied by the MMSE filtering unit. The LUD (including forward and backward substitution) requires another third of the detector's area. All remaining processing units and the input/output interface occupy the remaining third of the total circuit area.

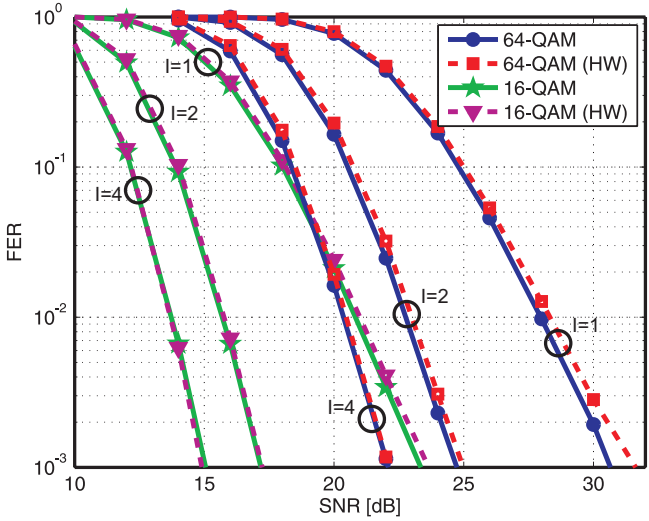


Figure 3.11: Fixed-point performance results for 64-QAM and 16-QAM of the SISO MMSE PIC implementation. Dashed lines correspond to the fixed-point hardware (HW) performance.

Fixed-Point Error-Rate Performance Thanks to using the techniques described in Section 3.3.1 for improving numerical stability, the internal precision of the SISO MMSE PIC architecture ranges between 12 bit to 29 bit[8], without a significant degradation in terms of error-rate performance. The fixed-point performance of the hardware implementation is shown in Figure 3.11 for 64-QAM and 16-QAM modulation. Note that a max-log BCJR algorithm has been used for SISO channel decoding. We emphasize that the implementation loss is remarkably small, i.e., lower than 0.35 dB SNR (at 1% FER) for 64-QAM in the first iteration ($I = 1$), and is negligible for 16-QAM. Hence, our implementation is able to achieve close-to floating-point performance.

[8]The maximum precision of 29 bit is required during LUD. The other units employ (often significantly) less bits.

Table 3.5: Area breakdown of the SISO MMSE PIC detector ASIC in 90 nm (1P/9M) CMOS technology.

Unit	mm^2	kGE	%
Preprocessing	0.16	50.4	12.3
Soft-symbol, variance, **A**	0.11	34.4	8.4
PIC 1 and PIC 2	0.12	38.4	9.4
LUD, forward substitution	0.21	68.1	16.6
Backward substitution	0.22	70.2	17.1
MMSE filtering	0.35	112.4	27.4
LLR computation unit	0.03	10.3	2.5
Miscellaneous[a]	0.09	26.0	6.3
Total	1.29	410.2	100

[a]Denotes logic used for the input/output-interface of the ASIC.

Table 3.6: VLSI Implementation comparison of hard-output k-best, linear soft-output MMSE, and SISO MMSE PIC detection.

	This work	Burg *et al.* [16]	Shabany and Gulak [93]
Detection algorithm	SISO MMSE PIC	soft-output MMSE	hard-output k-best
Technology [nm]	90	130	130
Clock freq. [MHz]	620	320	282
Preprocessing [kGE]	410[a]	251	n.a.
Detection [kGE]		67	114
Throughput[b] [Mbps]	826	1386	950
Eff. [kGE/Mbps]	0.50	0.23	n.a.

[a]Preprocessing is contained in the SISO MMSE PIC algorithm.

[b]Corresponds to the throughput (in terms of million LLR-values) per second. The throughput of the two 130 nm implementations has been up-scaled by a factor of 1.45 to account for technology scaling (see Section 6.1.2).

Comparison with the State-of-the-Art Since no VLSI implementation of a soft-input soft-output detector for MIMO systems exists in the open literature, we compare our results to the soft-output MMSE detector described in [16] (we include the preprocessing stage but exclude all memories). In addition, a comparison with the hard-output k-best MIMO detector reported in [93] is given. The k-best implementation does not report any preprocessing complexity (which requires significant amount of circuit area).

Table 3.6 compares the three architectures. The SISO MMSE PIC implementation achieves the highest clock frequency, mainly due to its technology advantage. The (technology-scaled) throughput of the soft-output MMSE detector [16] is 68% higher compared to that of the SISO MMSE PIC. The hard-output k-best detector reported in [93] attains a 15% higher throughput compared to that of our implementation. The soft-output MMSE detector is approximately two times more efficient (in terms of kGE/Mbps) than our SISO MMSE PIC implementation. Due to the lack of a preprocessing unit in [93], it is difficult to calculate the hardware-efficiency of the k-best implementation in a fair manner. If assuming that the QRD preprocessing unit of [16] is used for the k-best implementation, one obtains a total area of 365 kGE, leading to an efficiency of 0.38 kGE/Mbps, which is 28% better than that of the SISO MMSE PIC implementation.

Conclusion We conclude that the proposed SISO MMSE PIC implementation is approximately two times less efficient compared to state-of-the-art hard-output or soft-output MIMO detector implementations. However, the SISO MMSE PIC implementation is—to the best of our knowledge—the only SISO detector for iterative MIMO decoding and hence, is the only detector that is able to obtain the tremendous performance gains offered by iterative MIMO decoding.

Chapter 4

Soft-Input Soft-Output Sphere Decoding

In the previous chapter, we studied a sub-optimum SISO detection algorithm for iterative MIMO decoding. In this chapter, we describe a novel high-performance SISO detection algorithm based on sphere decoding (SD), which is able to achieve near-optimum SISO performance.

In Section 4.1, we introduce the basics of SD and describe its extension to soft-input soft-output MIMO detection. The novel SD-based SISO detection algorithm, referred to as SISO single tree-search (STS) SD algorithm, is introduced in Section 4.2. In the Sections 4.3, 4.4, and 4.5, we describe a variety of techniques to further reduce the computational complexity or improve the performance of the algorithm. Simulation results are shown in Section 4.6. In Section 4.7, we provide VLSI implementation results for the soft-output STS-SD algorithm and compare the performance to related implementations.

4.1 SISO Sphere Decoding

This section introduces the basics of (hard-output) SD and then, reviews well-established SD algorithms for detection in MIMO systems.

4.1.1 The Basics of Sphere Decoding

The detection algorithm known as sphere decoding (SD) was initially developed by Pohst in 1981 [94] for computation of minimal-length lattice vectors and has been refined by Fincke and Pohst in 1985 [95]. The first application of SD in communication theory was described in Mow's Master's Thesis in 1991 [96]; therein SD was used for ML sequence estimation in ISI channels. A corresponding conference publication appeared in 1992 [97]. Viterbo and Biglieri applied the Pohst algorithm to lattice decoding in 1993 [98], which sparked tremendous research activities on SD. In 1994, Schnorr and Euchner (SE) [99] presented an improvement to the algorithm, which leads to lower computational complexity than the Pohst variant. In 1999, Viterbo and Boutros used the Pohst algorithm for lattice decoding in fading channels [100] and in 2000, Damen *et al.* [101] applied Schnorr-Euchner sphere decoding (SESD) to detection of space-time codes. An extensive summary on SD and its applications has been published by Agrell *et al.* in 2002 [102]. The first soft-input soft-output MIMO detector based on SD was described in 2003 by Hochwald and ten Brink [19], where it was shown that the proposed SISO SD algorithm in combination with iterative MIMO decoding allows to achieve near-capacity in MIMO systems. The first VLSI implementation of the SD algorithm was described by Burg *et al.* in 2004 [103], where it has been demonstrated that SD is suitable for high-performance hard-output MIMO detection in practical systems.

In the following, we describe the basics of the complex-valued[1] SESD algorithm and briefly outline some of the most prominent (optimal and sub-optimal) MIMO detection algorithms that are related to SD.

[1] In some publications, a real-valued variant of SD is employed (e.g., [104]). This variant is obtained by decomposing the components of (2.2) into

$$\bar{\mathbf{y}} = \begin{bmatrix} \Re\{\mathbf{y}\} \\ \Im\{\mathbf{y}\} \end{bmatrix}, \quad \bar{\mathbf{H}} = \begin{bmatrix} \Re\{\mathbf{H}\} & -\Im\{\mathbf{H}\} \\ \Im\{\mathbf{H}\} & \Re\{\mathbf{H}\} \end{bmatrix}, \quad \text{and} \quad \bar{\mathbf{s}} = \begin{bmatrix} \Re\{\mathbf{s}\} \\ \Im\{\mathbf{s}\} \end{bmatrix}$$

leading to the equivalent real-valued input-output relation $\bar{\mathbf{y}} = \bar{\mathbf{H}}\bar{\mathbf{s}} + \bar{\mathbf{n}}$, where the noise is $\bar{\mathbf{n}} \sim \mathcal{N}(0, \frac{1}{2}N_\mathrm{o}\mathbf{I}_{2M_\mathrm{R}})$. The real-valued decomposition is, for example, essential for decoding of linear STBCs using the SD techniques. In this case, $\bar{\mathbf{G}}\bar{\mathbf{s}}$ is the transmitted vector, where $\bar{\mathbf{G}}$ corresponds to a (real-valued) matrix that represents the employed STBC (see [105] for more details).

Schnorr-Euchner SD with Radius Reduction

The main idea underlying SD is to transform ML detection (2.6) into a tree-search problem, which can then be solved more efficiently by the means of pruning. To this end, the channel matrix $\mathbf{H}$ of (2.2) is first QR-decomposed according to $\mathbf{H} = \mathbf{QR}$, where the $M_{\mathrm{R}} \times M_{\mathrm{T}}$ matrix $\mathbf{Q}$ is unitary, and the $M_{\mathrm{T}} \times M_{\mathrm{T}}$ upper-triangular matrix $\mathbf{R}$ has real-valued non-negative entries on its main diagonal. Left-multiplying (2.2) by $\mathbf{Q}^H$ leads to the modified input-output relation

$$\tilde{\mathbf{y}} = \mathbf{Rs} + \mathbf{Q}^H \mathbf{n} \tag{4.1}$$

with $\tilde{\mathbf{y}} = \mathbf{Q}^H \mathbf{y}$. Since $\mathbf{Q}^H \mathbf{n}$ has the same statistics as $\mathbf{n}$, an equivalent formulation of the ML detection problem in (2.6) corresponds to

$$\hat{\mathbf{s}}^{\mathrm{ML}} = \underset{\mathbf{s} \in \mathcal{O}^{M_{\mathrm{T}}}}{\arg\min} \|\tilde{\mathbf{y}} - \mathbf{Rs}\|^2. \tag{4.2}$$

We next define the partial symbol vectors (PSVs) $\mathbf{s}^{(i)} = [\, s_i \, \cdots \, s_{M_T} \,]^T$ and note that they can be arranged in a tree that has its root just above level $i = M_{\mathrm{T}}$ and leaves, on level $i = 1$, which correspond to symbol vectors $\mathbf{s}$. In the following, the bit-label associated with $\mathbf{s}^{(i)}$ is denoted by $\mathbf{x}^{(i)}$. The Euclidean distance

$$d(\mathbf{s}) = \|\tilde{\mathbf{y}} - \mathbf{Rs}\|^2 \tag{4.3}$$

in (4.2) can be computed recursively by defining $d\big(\mathbf{s}^{(i)}\big) = d_i$ with the partial Euclidean distances (PEDs)

$$d_i = d_{i+1} + |e_i|^2 \quad , \; i = M_{\mathrm{T}}, \ldots, 1\,, \tag{4.4}$$

the initialization $d_{M_T+1} = 0$, and the distance increments (DIs)

$$|e_i|^2 = \left| \tilde{y}_i - \sum_{j=i}^{M_{\mathrm{T}}} R_{i,j} s_j \right|^2. \tag{4.5}$$

Since the dependence of the PED d_i on the symbol vector $\mathbf{s}$ is only through the PSV $\mathbf{s}^{(i)}$, the ML detection problem has been transformed into a weighted tree-search problem: PSVs and PEDs are associated with *nodes* and *branches* correspond to DIs. The resulting

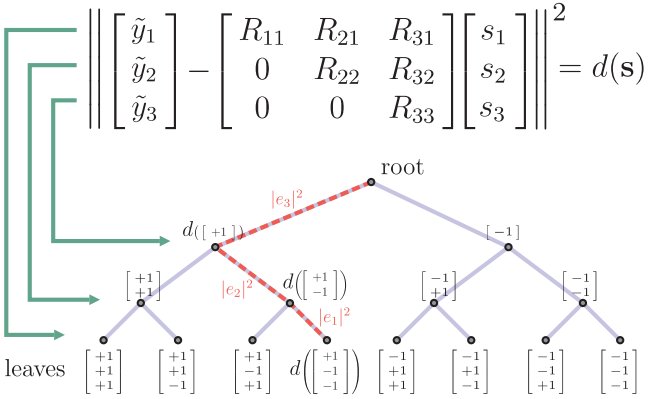


Figure 4.1: Example of the weighted tree for a MIMO system with $M_{\mathrm{T}} = 3$ and BPSK modulation.

tree-structure is illustrated in Figure 4.1. For brevity, we shall often say "the node $\mathbf{s}^{(i)}$" to refer to the node corresponding to the PSV $\mathbf{s}^{(i)}$. We shall furthermore use $d\big(\mathbf{s}^{(i)}\big)$ and $d\big(\mathbf{x}^{(i)}\big)$ interchangeably to denote d_i. Each path from the root down to a leaf corresponds to a symbol vector $\mathbf{s} \in \mathcal{O}^{M_T}$. The solution of (4.2) corresponds to the path from the root to the leaf associated with the smallest metric.

Schnorr-Euchner Sphere Decoding The basic building block underlying SESD with radius reduction [33, 99, 102] is briefly summarized as follows: The search in the tree is constrained to nodes which lie within a radius r around $\tilde{\mathbf{y}}$ and tree traversal is performed depth-first, visiting the children of a given node in ascending order of their PEDs. A node $\mathbf{s}^{(i)}$ with PED d_i can be pruned (along with the entire subtree originating from this node) whenever the sphere constraint (SC)

$$d\Big(\mathbf{s}^{(i)}\Big) < r^2$$

is violated. Figure 4.2 illustrates the SC.

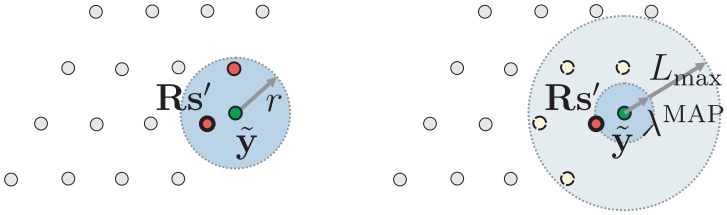


Figure 4.2: Left: the sphere constraint bounds the search to lattice points that are within a radius r around the received vector $\tilde{\mathbf{y}}$ (the transmitted symbol vector is denoted by $\mathbf{s}'$). Right: Extrinsic LLR clipping bounds the search for the MAP solution and all counter-hypotheses to $r_{\max} = \sqrt{\lambda^{\text{MAP}} + L_{\max}}$.

Radius Reduction The radius r has to be chosen sufficiently large such that the SD algorithm finds at least the ML solution. However, choosing r too large leads to high complexity, because a large number of nodes will satisfy the sphere constraint—a too low choice of the radius might inhibit the decoder to find any solution. In order to avoid the problem of choosing a suitable radius r altogether, we employ a technique known as radius reduction [102], which consists of initializing the algorithm with $r = \infty$, and performing the update $r^2 \leftarrow d(\mathbf{s})$ whenever a valid leaf node $\mathbf{s}$ has been found. This approach maintains ML-optimality and still leads to efficient pruning of the tree.

Complexity Measure The complexity measure for SD employed in the remainder of this chapter corresponds to the number of nodes visited by the decoder including the leaf nodes, but excluding the root. This measure was shown in [33] to be representative of the hardware complexity of a VLSI implementation for hard-output SESD.

Other Tree-Search-Based Detection Schemes

In order to attain low computational complexity using tree-search based MIMO detection, a plethora of optimal and sub-optimal algorithms have been proposed in the literature. The most prominent detection algorithms are briefly outlined below.

Successive interference cancellation (SIC) has already been described in Section 2.2.1 and is one of simplest MIMO detection algorithm that can be associated to a tree-search problem. A better performing (but still sub-optimal) MIMO detection method is the k-best algorithm [106], which attains a fixed-complexity while only searching for a small subset of nodes in the tree. Corresponding VLSI implementation results show that the k-best algorithm is well-suited for practical implementation of sub-optimal hard-output MIMO detection, e.g., [22,93,107,108]. Other tree-search-based detection methods that are related to the k-best algorithm are the M-algorithm [109] and smart candidate adding [110], which also exhibit a fixed-complexity.

The fixed sphere decoder (FSD) [111] is one of the best-performing sub-optimal MIMO detection schemes and was shown to achieve full diversity and a vanishing SNR performance loss for $\mathsf{SNR} \to 0$ [112]; this is achieved by employing layer-sorting followed by multiple SIC stages. Implementations on field-programmable gate array (FPGA) [113] and ASIC [114] prove that the FSD is suitable for practical implementation. Note, however, that it is difficult to obtain high-quality soft-outputs from this algorithm (possible approaches to soft-output detection are described in [115]).

Dijkstra's algorithm [116] applied to MIMO detection [117] was shown to visit the minimum number of nodes among all SD algorithms[2] while offering ML performance [119]. Unfortunately, the number of required comparisons is (often significantly) larger than that of the SESD and the memory requirements grow exponentially with the number of transmit antennas [120], which inhibits efficient implementation in practical systems.

Another decoding method—initially designed for low-complexity channel decoding—is the stack algorithm [121], which, in contrast to Dijkstra's algorithm, employs a limited amount of memory and hence, attains sub-optimal performance. The list-sequential (LISS) detector [30, 122] builds upon the stack algorithm and was shown to be suitable for low-complexity (but sub-optimal) soft-output MIMO detection. Sequential decoding [123] was shown to yield low-complexity for MIMO detection. To the best of our knowledge, no correspond-

[2]This statement is only valid if no lower-bounds are used to speed-up the tree-search stage (see, e.g., [118]).

ing soft-output MIMO detection algorithm has been described in the open literature.

In the remainder of this chapter, we only focus on SESD-related MIMO detection algorithms that are able to attain *optimum* SISO performance.

4.1.2 List Sphere Decoding

We first review the max-log approximation to exact LLRs, which is key to employ SD for computation of LLRs. Then, we briefly describe the SISO detection algorithm for iterative MIMO systems developed by Hochwald and ten Brink in 2003 [19], which is known as list sphere decoding (LSD) in the literature.

Computation of the Max-Log LLRs

In order to employ SD for computation of the intrinsic LLRs (2.23), the QRD of the channel matrix (as shown in Section 4.1.1 for the SESD) is used; this leads to an equivalent formulation of intrinsic LLRs

$$L_{i,b} = \log\left(\sum_{\mathbf{s}\in\mathcal{X}_{i,b}^{(+1)}} \mathrm{p}(\tilde{\mathbf{y}}\,|\,\mathbf{s},\mathbf{R})\,\mathrm{P}[\mathbf{s}]\right) - \log\left(\sum_{\mathbf{s}\in\mathcal{X}_{i,b}^{(-1)}} \mathrm{p}(\tilde{\mathbf{y}}\,|\,\mathbf{s},\mathbf{R})\,\mathrm{P}[\mathbf{s}]\right) \tag{4.6}$$

where $\mathrm{P}[\mathbf{s}]$ corresponds to the prior term and

$$\mathrm{p}(\mathbf{y}\,|\,\mathbf{s},\mathbf{R}) = \frac{1}{(\pi N_\mathrm{o})^{M_\mathrm{T}}} \exp\left(-\frac{\|\mathbf{y}-\mathbf{R}\mathbf{s}\|^2}{N_\mathrm{o}}\right).$$

Straightforward computation of (4.6) requires the evaluation of $|\mathcal{O}|^{M_\mathrm{T}}$ terms per LLR value, which leads to prohibitively high computational complexity, in general (see Section 2.2.2).

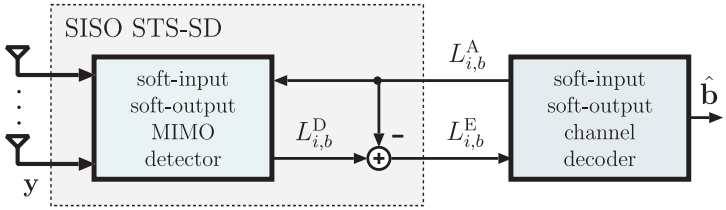


Figure 4.3: Iterative MIMO decoder. The SISO STS-SD algorithm (corresponding to the dashed box) directly computes extrinsic LLRs.

Key to enable SD (and hence, to obtain low complexity) for LLR computation is to apply the standard max-log approximation[3] to (4.6), which allows us to reformulate LLR computation as a weighted tree-search problem that can be solved efficiently using SD techniques [48,94–96,98–100,124–126]. In the remainder of this chapter, we consider an iterative MIMO decoder as depicted in Figure 4.3. The soft-input soft-output MIMO detector computes *intrinsic* max-log LLRs according to [19]

$$L^{\mathrm{D}}_{i,b} \triangleq \min_{\mathbf{s}\in\mathcal{X}^{(-1)}_{i,b}} \left\{ \frac{1}{N_{\mathrm{o}}} \left\| \tilde{\mathbf{y}} - \mathbf{R}\mathbf{s} \right\|^2 - \log \mathrm{P}[\mathbf{s}] \right\} - \min_{\mathbf{s}\in\mathcal{X}^{(+1)}_{i,b}} \left\{ \frac{1}{N_{\mathrm{o}}} \left\| \tilde{\mathbf{y}} - \mathbf{R}\mathbf{s} \right\|^2 - \log \mathrm{P}[\mathbf{s}] \right\}. \tag{4.7}$$

where the prior $\mathrm{P}[\mathbf{s}]$ is, for example, delivered by an outer channel decoder in the form of *a priori* LLRs

$$L^{\mathrm{A}}_{i,b} \triangleq \log\left(\frac{\mathrm{P}[x_{i,b}=+1]}{\mathrm{P}[x_{i,b}=-1]} \right), \quad \forall i,b.$$

Based on the intrinsic LLRs in (4.7), the detector computes the *extrinsic* LLRs using

$$L^{\mathrm{E}}_{i,b} \triangleq L^{\mathrm{D}}_{i,b} - L^{\mathrm{A}}_{i,b}, \quad \forall i,b, \tag{4.8}$$

[3]The max-log approximation (cf. Section 3.2.1) results from the Jacobian logarithm [77] and corresponds to $\log\left(\sum_k \exp(a_k)\right) \approx \max_k\{a_k\}$. This approximation entails a small error-rate performance loss compared to exact LLRs in (2.23). Corresponding simulation results are shown in Section 4.6.1.

that are passed to a subsequent SISO channel decoder. We emphasize that evaluation of the two terms in (4.7) can be performed using SD techniques, which requires (often significantly) less computational complexity than computation of exact LLRs in (2.23). Note that we neglected the additive constant in each of the two minima in (4.7) that results from the part of the noise $\mathbf{n}$ that is orthogonal to the range-space of $\mathbf{H}$. This is possible as the constant in question is independent of $\mathbf{s}$ and, hence, cancels out upon taking the difference in (4.7).

The List Sphere Decoding Algorithm

The main idea underlying the LSD algorithm [19] is to constrain the max-log LLR computation in (4.7) to a small subset of all candidate vectors in $\mathcal{O}^{M_\mathrm{T}}$ such that

$$\begin{aligned} L_{i,b}^{\mathrm{D}} \approx & \min_{\mathbf{s}\in\left\{\mathcal{S}\cap\mathcal{X}_{i,b}^{(-1)}\right\}} \left\{ \frac{1}{N_\mathrm{o}} \left\|\tilde{\mathbf{y}}-\mathbf{R}\mathbf{s}\right\|^2 - \log \mathrm{P}[\mathbf{s}] \right\} \\ & - \min_{\mathbf{s}\in\left\{\mathcal{S}\cap\mathcal{X}_{i,b}^{(+1)}\right\}} \left\{ \frac{1}{N_\mathrm{o}} \|\tilde{\mathbf{y}}-\mathbf{R}\mathbf{s}\|^2 - \log \mathrm{P}[\mathbf{s}] \right\} \end{aligned} \tag{4.9}$$

where the subset $\mathcal{S} \subset \mathcal{O}^{M_\mathrm{T}}$ is referred to as the *list* and $S = |\mathcal{S}|$ denotes the list size. Since

$$\left|\mathcal{S}\cap\mathcal{X}_{i,b}^{(x)}\right| \leq \left|\mathcal{X}_{i,b}^{(x)}\right| \quad \text{for} \quad x\in\{+1,-1\},\ \forall i,b$$

evaluation of (4.9) can lead to lower computational complexity than that of (4.7) if S is significantly smaller than $|\mathcal{O}|^{M_\mathrm{T}}$. In order to obtain a good approximation of (4.7) using (4.9), it is important that the list $\mathcal{S}$ contains candidate vectors with small Euclidean distance. To this end, the LSD builds the list such that it contains those candidate vectors corresponding to the S-smallest Euclidean distances (4.3).

The list is constructed using SD, which requires to extend the algorithm with list administration and a modified version of radius reduction. List administration corresponds to replacing the candidate vector in $\mathcal{S}$ associated with the largest Euclidean distance, whenever the LSD reaches a leaf. After list administration has been carried out, a new search radius can be obtained as follows

$$r^2 \leftarrow \max_{\mathbf{s}\in\mathcal{S}} d(\mathbf{s}) \tag{4.10}$$

which ensures that no candidate vector is being found that has a larger Euclidean distance than the worst one (in terms of Euclidean distance) in the list. Note that the Euclidean distances can be stored in a separate list to avoid costly re-computations. List administration and radius reduction (4.10) ensure that the list contains exactly the S-best candidates after the tree-search procedure, i.e.,

$$\max_{\mathbf{s}\in\mathcal{S}} d(\mathbf{s}) \leq \min_{\mathbf{s}'\in\{\mathcal{O}^{M_{\mathrm{T}}}\setminus\mathcal{S}\}} d\big(\mathbf{s}'\big).$$

The VLSI implementation of a soft-output LSD in [127] demonstrates that LSD can be implemented in practical systems, but suffers from the following disadvantages:

- Evaluation of (4.9) requires high computational complexity if the list-size is large. Additionally, list administration (i.e., searching and replacing of the worst candidate vector in the list) can quickly lead to prohibitive VLSI implementation complexity. Hence, LSD implementations are, in general, only efficient for small list-sizes [127].

- In the case where no transmit vector with the $x_{i,b}$th bit equal to a is contained in $\mathcal{S}$ (i.e., $\mathcal{S}\cap\mathcal{X}_{i,b}^{(a)}=\emptyset$), no LLR-value can be computed for this particular bit. In [19], it was suggested to set the corresponding LLR value to the ML solution $x_{i,b}^{\mathrm{ML}}$ with magnitude 8, which, however, leads to a noticeable performance loss, especially for small lists.

In summary, LSD requires a large list to obtain near (max-log) optimal performance, which leads to high computational complexity in the list administration procedure.

Two algorithms related to the LSD exist in the literature. The SISO detector proposed in [31] first computes the ML-estimate using SD and then, generates a list of candidate vectors around the ML solution during a second SD run. The algorithm described in [32] considers the priors during list computation (cf. Section 4.1.3). Note, however, that both detectors have the main disadvantage that the list needs to be re-built for each iteration, which leads to a (often significant) complexity increase compared to the LSD algorithm in [19].

4.1.3 Max-Log LLR Computation as a Tree-Search

In order to avoid the drawbacks of LSD, more sophisticated tree-search methods need to be considered. To this end, we first show how intrinsic max-log LLRs can be computed *exactly* using SD techniques and then, we briefly review the repeated tree-search (RTS) algorithm developed by Wang and Giannakis in 2004 [125], which builds upon the described method.

Transformation to a Tree-Search Problem

To enable exact computation of max-log LLRs in (4.7) using a tree search, it is important to realize that one of the two minima in (4.7) corresponds to

$$\lambda^{\mathrm{MAP}} \triangleq \frac{1}{N_{\mathrm{o}}} \left\| \tilde{\mathbf{y}} - \mathbf{R}\mathbf{s}^{\mathrm{MAP}} \right\|^2 - \log \mathrm{P}\left[\mathbf{s}^{\mathrm{MAP}}\right] \tag{4.11}$$

which is associated with the MAP solution of the MIMO detection problem given in (2.5)

$$\mathbf{s}^{\mathrm{MAP}} = \underset{\mathbf{s}\in\mathcal{O}^{M_{\mathrm{T}}}}{\arg\min} \left\{ \frac{1}{N_o} \left\| \tilde{\mathbf{y}} - \mathbf{R}\mathbf{s} \right\|^2 - \log \mathrm{P}[\mathbf{s}] \right\}. \tag{4.12}$$

The other minimum in (4.7) can be computed as

$$\lambda_{i,b}^{\overline{\mathrm{MAP}}} \triangleq \min_{\mathbf{s}\in\mathcal{X}_{i,b}^{\left(\overline{x_{i,b}^{\mathrm{MAP}}}\right)}} \left\{ \frac{1}{N_o} \left\| \tilde{\mathbf{y}} - \mathbf{R}\mathbf{s} \right\|^2 - \log \mathrm{P}[\mathbf{s}] \right\} \tag{4.13}$$

where $\overline{x_{i,b}^{\mathrm{MAP}}}$ is the (bit-wise) *counter-hypothesis* to the MAP hypothesis. With the definitions (4.11) and (4.13), the intrinsic max-log LLRs in (4.7) can be written ($\forall i, b$) in compact form as

$$L_{i,b}^{\mathrm{D}} = \begin{cases} \lambda_{i,b}^{\overline{\mathrm{MAP}}} - \lambda^{\mathrm{MAP}}, & x_{i,b}^{\mathrm{MAP}} = +1 \\ \lambda^{\mathrm{MAP}} - \lambda_{i,b}^{\overline{\mathrm{MAP}}}, & x_{i,b}^{\mathrm{MAP}} = -1. \end{cases} \tag{4.14}$$

We can therefore conclude that efficient max-log optimal soft-input soft-output MIMO detection reduces to efficiently identifying $\mathbf{s}^{\mathrm{MAP}}$,

λ^{MAP}, and all metrics associated with the counter-hypotheses $\lambda_{i,b}^{\overline{\mathrm{MAP}}}$ $(\forall i, b)$.

Using the same procedure as done for hard-output SESD described in Section 4.1.1, we use the notion of PSVs and note that they can be arranged in a tree that has its root just above level $i = M_{\mathrm{T}}$ and leaves, on level $i = 1$, which correspond to symbol vectors $\mathbf{s}$. The distances

$$d(\mathbf{s}) = \frac{1}{N_{\mathrm{o}}} \left\| \tilde{\mathbf{y}} - \mathbf{R}\mathbf{s} \right\|^2 - \log \mathrm{P}[\mathbf{s}] \tag{4.15}$$

in (4.11) and (4.13) can be computed recursively if the following factorization holds:

$$\mathrm{P}[\mathbf{s}] = \prod_{i=1}^{M_{\mathrm{T}}} \mathrm{P}\left[\mathbf{s}^{(i)}\right], \tag{4.16}$$

which is assumed from now on. Note that in practice, the symbols s_i $(i = 1, \ldots, M_{\mathrm{T}})$ are often statistically independent across spatial streams; this satisfies (4.16) trivially with $\mathrm{P}[\mathbf{s}] = \prod_{i=1}^{M_{\mathrm{T}}} \mathrm{P}[s_i]$. We can now rewrite (4.15) as

$$d(\mathbf{s}) = \sum_{i=1}^{M_{\mathrm{T}}} \left(\frac{1}{N_{\mathrm{o}}} \left| \tilde{y}_i - \sum_{j=i}^{M_{\mathrm{T}}} R_{i,j} s_j \right|^2 - \log \mathrm{P}\left[\mathbf{s}^{(i)}\right] \right)$$

which can be evaluated recursively as $d(\mathbf{s}) = d_1$, with the partial distances (PDs)

$$d_i = d_{i+1} + |e_i|, \quad i = M_{\mathrm{T}}, \ldots, 1,$$

the initialization $d_{M_{\mathrm{T}}+1} = 0$, and the DIs

$$|e_i| = \frac{1}{N_{\mathrm{o}}} \left| \tilde{y}_i - \sum_{j=i}^{M_{\mathrm{T}}} R_{i,j} s_j \right|^2 - \log \mathrm{P}\left[\mathbf{s}^{(i)}\right]. \tag{4.17}$$

Note that the DIs are non-negative since the prior terms satisfy $-\log \mathrm{P}\left[\mathbf{s}^{(i)}\right] \geq 0$. The dependence of the PD d_i on the symbol vector $\mathbf{s}$ is, thanks to the upper triangularity of $\mathbf{R}$ and the assumption (4.16), only through the PSV $\mathbf{s}^{(i)}$ (similarly to SESD). Thus, the MAP detection problem and the computation of the intrinsic max-log LLRs

has been transformed into a tree-search problem: PSVs and PDs are associated with nodes, branches correspond to DIs. For brevity, we shall often say "the node $\mathbf{s}^{(i)}$" to refer to the node corresponding to the PSV $\mathbf{s}^{(i)}$. We shall furthermore use $d\big(\mathbf{s}^{(i)}\big)$ and $d\big(\mathbf{x}^{(i)}\big)$ interchangeably to denote d_i. Each path from the root node down to a leaf node corresponds to a symbol vector $\mathbf{s} \in \mathcal{O}^{M_{\mathrm{T}}}$. The result in (4.11) and (4.13) corresponds to the leaf associated with the smallest metric in $\mathcal{O}^{M_{\mathrm{T}}}$ and $\mathcal{X}_{i,b}^{\left(\overline{x_{i,b}^{\mathrm{MAP}}}\right)}$, respectively. The RTS and STS-SD algorithms use elements of SESD [99, 102], briefly summarized as follows: The search in the weighted tree is constrained to nodes which lie within a radius[4] r around $\tilde{\mathbf{y}}$ and tree traversal is performed depth-first, visiting the children of a given node in ascending order of their PDs. A node $\mathbf{s}^{(i)}$ with PD d_i can be pruned (along with the entire subtree originating from this node) whenever the tree-pruning criterion

$$d_i \geq r^2 \tag{4.18}$$

is satisfied. In the remainder of this chapter, (4.18) is referred to as the "standard pruning criterion."

Computing the max-log LLRs in (4.7) requires to determine the metrics $\lambda_{i,b}^{\overline{\mathrm{MAP}}}$, which, for given i, b, is accomplished by traversing only those parts of the tree that have leaves in $\mathcal{X}_{i,b}^{\left(\overline{x_{i,b}^{\mathrm{MAP}}}\right)}$. Since this computation has to be carried out for every bit, it is immediately obvious that LLR computation results in an order of magnitude increase in computational complexity compared to hard-output SD. The situation is further exacerbated by the fact that forcing the SE sphere decoder into subtrees, when computing the minima in (4.13) leads to significantly less efficient tree pruning behavior, which finally results in an overall complexity increase (compared to that of hard-output SESD) of two orders of magnitude.

The Repeated Tree Search Algorithm

In the following, we briefly describe the RTS algorithm [125]. This method starts by solving (2.5) using SD (as described above) and

[4]Note that r only corresponds to the radius of a hypersphere if the prior is constant $\mathrm{P}[\mathbf{s}] = c$, $\forall \mathbf{s}$.

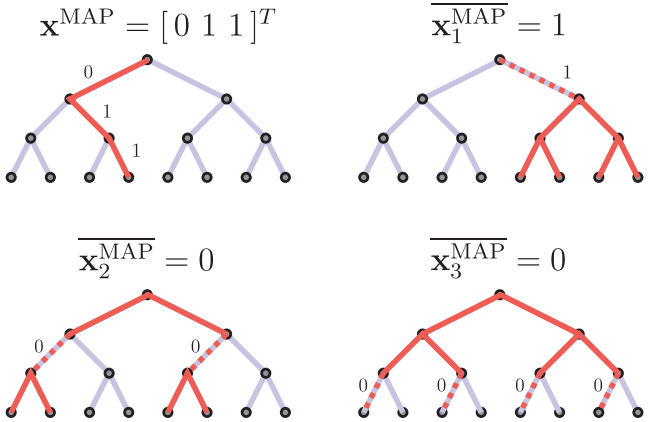


Figure 4.4: Example of the prepruning procedure (BPSK constellation, $M_{\mathrm{T}} = 3$) of the RTS algorithm [125]. Counter-hypotheses to the MAP solution are found by forcing SD through the dashed branches.

to rerun the decoder to solve (4.13) for each bit in the symbol vector (i.e., QM_{T} times). When rerunning the SESD algorithm to determine $\lambda_{i,b}^{\overline{\mathrm{MAP}}}$ in (4.13), the search tree is prepruned by forcing the decoder to exclude all nodes from the search for which $x_{i,b} = x_{i,b}^{\mathrm{MAP}}$. This prepruning procedure is illustrated in Figure 4.4. Following the proposal in [125] and initializing the sphere decoder with $r = \infty$ in each of the QM_{T} runs required to obtain $\lambda_{i,b}^{\overline{\mathrm{MAP}}}$, will lead to significant computational complexity. It is, therefore, important to realize that (without compromising max-log optimality) the search radius $r_{i,b}$ can be initialized by setting it to the minimum value of $\|\tilde{\mathbf{y}} - \mathbf{R}\mathbf{s}\|^2$ over all $\mathbf{s} \in \mathcal{X}_{i,b}^{\left(\overline{x_{i,b}^{\mathrm{MAP}}}\right)}$ found during preceding tree traversals.

The RTS algorithm suffers from the following disadvantages:

i) the repeated traversal of large parts of the tree, which entails a large number of redundant computations;

ii) significantly less efficient pruning behavior when computing the $\lambda_{i,b}^{\overline{\mathrm{MAP}}}$, which is caused by the need to minimize over the sub-

sets $\mathcal{X}_{i,b}^{\left(\overline{x_{i,b}^{\mathrm{MAP}}}\right)}$. The underlying reason is that pruning efficiency decreases significantly when forcing the sphere decoder through specific branches at levels further down the tree.

As noted in [110], the problem in ii) can partly be mitigated by changing the detection order in each run. The resulting need for multiple QR decompositions, however, leads to an overall increase in terms of hardware complexity.

4.1.4 Tightening of the Tree-Pruning Criterion

Tightening of the tree-pruning criterion (4.18), i.e., reduction of the right-hand side (RHS) of (4.18), without sacrificing (max-log) optimality is highly desirable as it reduces the (tree-search) complexity. Such a reduction can be accomplished, for example, through techniques based on semi-definite relaxation and H^{∞}-estimation theory as proposed in [128]. Unfortunately, these approaches often entail high computational complexity and are, hence, not well-suited for practical (VLSI) implementation.

The Basic Idea

In the following, we propose an alternative approach which relies on the observation that the DIs (4.17) contain a—generally non-zero—bias given by

$$|b_i| \triangleq \min_{\mathbf{s}^{(i)} \in \mathcal{O}^{M_{\mathrm{T}}+1-i}} |e_i|, \quad i = 1, \ldots, M_{\mathrm{T}}. \tag{4.19}$$

Consider the case where the detector stands at node $\mathbf{s}^{(i)}$ on level i with corresponding PD d_i. All leaf-level PDs d_1 that can be reached from the node $\mathbf{s}^{(i)}$ satisfy

$$d_1 \geq d_i + \sum_{j=1}^{i-1} |b_j|. \tag{4.20}$$

At level i, we can therefore prune every node that satisfies a tightened version of the tree-pruning criterion in (4.18), namely

$$d_i \geq r^2 - \sum_{j=1}^{i-1} |b_j|. \tag{4.21}$$

Computation of the bias term (4.19) requires enumeration of $|e_i|$ over all $\mathbf{s}^{(i)} \in \mathcal{O}^{M_\mathrm{T}+1-i}$, which leads to prohibitive computational complexity. The major portion of this complexity is caused by the computation of the Euclidean distance-term $\frac{1}{N_\mathrm{o}}\big|\tilde{y}_i - \sum_{j=i}^{M_\mathrm{T}} R_{i,j} s_j\big|^2$ in (4.17), whose contribution to the bias (4.19), as it turns out (corresponding simulation results are shown in Section 4.6.2), is negligible. Hence, we only consider the contribution to $|b_i|$ caused by the prior term $-\log \mathrm{P}\big[\mathbf{s}^{(i)}\big]$ and we define accordingly

$$|p_i| = \min_{\mathbf{s}^{(i)} \in \mathcal{O}^{M_\mathrm{T}+1-i}} \Big\{ -\log \mathrm{P}\Big[\mathbf{s}^{(i)}\Big] \Big\}. \tag{4.22}$$

The corresponding tightened tree-pruning criterion is then given by

$$d_i \geq r^2 - \sum_{j=1}^{i-1} |p_j|. \tag{4.23}$$

For the case of the individual symbols s_i ($i = 1, \ldots, M_\mathrm{T}$) being statistically independent, i.e., $\mathrm{P}[\mathbf{s}] = \prod_{i=1}^{M_\mathrm{T}} \mathrm{P}[s_i]$, we have

$$|p_i| = \min_{s_i \in \mathcal{O}} \big\{ -\log \mathrm{P}[s_i] \big\}$$

so that computation of the RHS of (4.22) results in significantly smaller complexity than that required to compute the RHS of (4.19).

We emphasize that the tightened tree-pruning criterion (4.23) preserves max-log optimality and leads, in general, to significant complexity savings, when compared to the standard pruning criterion (4.18), which is widely adopted in the literature [28–32]. To see this, consider the case where all constellation points are equally likely[5], i.e.,

[5]This, for example, is the case when no a priori information is available and all transmitted bits are equally likely.

$\mathrm{P}[s_i] = |\mathcal{O}|^{-1}$ for all $s_i \in \mathcal{O}$ and $i = 1, \ldots, M_{\mathrm{T}}$. The corresponding total bias from level i down to the leaf level is given by

$$\sum_{j=1}^{i-1} |p_j| = (i-1) \log |\mathcal{O}|$$

which can be large, especially for nodes close to the root. Since pruning at and close to the root level, has significant impact on the number of nodes visited in the tree search, the tightened tree-pruning criterion (4.23) can lead to a major complexity reduction. Corresponding simulation results are provided in Section 4.6.2.

Tree Search in the Case of Statistically Independent Bits

We have seen above that statistical independence among individual symbols enables us to tighten the tree-pruning criterion at low additional computational complexity. For bit-interleaved coded modulation [44], the bits $x_{i,b}$ ($i = 1, \ldots, M_{\mathrm{T}}$, $b = 1, \ldots, Q$) are assumed to be statistically independent. As shown next, this independence on the bit-level can be exploited to get further reductions in computational complexity. To see this, consider the case where the MIMO detector obtains a priori LLRs $L^{\mathrm{A}}_{i,b}$ ($\forall i, b$) from an external device, e.g., a SISO channel decoder as depicted in Figure 4.3. We then have [41]

$$\mathrm{P}[s_i] = \prod_{b:x_{i,b}=+1} \frac{\exp\left(L^{\mathrm{A}}_{i,b}\right)}{1+\exp\left(L^{\mathrm{A}}_{i,b}\right)} \prod_{b:x_{i,b}=-1} \frac{1}{1+\exp\left(L^{\mathrm{A}}_{i,b}\right)}$$

which can be reformulated in more compact form as

$$\mathrm{P}[s_i] = \prod_{b=1}^{Q} \frac{\exp\left(\frac{1}{2}\left(1+x_{i,b}\right)L^{\mathrm{A}}_{i,b}\right)}{1+\exp\left(L^{\mathrm{A}}_{i,b}\right)}. \tag{4.24}$$

The contribution of the a priori LLRs to the prior term in the DIs in (4.17) can then be obtained from (4.24) as

$$-\log \mathrm{P}[s_i] = \tilde{K}_i - \sum_{b=1}^{Q} \frac{1}{2} x_{i,b} L^{\mathrm{A}}_{i,b} \tag{4.25}$$

where the constants

$$\tilde{K}_i = \sum_{b=1}^{Q} \left(\frac{1}{2} \left| L_{i,b}^{\mathrm{A}} \right| + \log \left(1 + \exp \left(- \left| L_{i,b}^{\mathrm{A}} \right| \right) \right) \right) \tag{4.26}$$

are independent of the binary-valued variables $x_{i,b}$ and $\tilde{K}_i > 0$ for $i = 1, \ldots, M_{\mathrm{T}}$. Because of $-\log \mathrm{P}[s_i] \geq 0$, we can trivially infer from (4.25) that $\tilde{K}_i - \sum_{b=1}^{Q} \frac{1}{2} x_{i,b} L_{i,b}^{A} \geq 0$. From (4.14) it follows that constant terms (i.e., terms that are independent of the variables $x_{i,b}$ and hence of $\mathbf{s}$) in (4.11) and (4.13) cancel out in the computation of the intrinsic LLRs $L_{i,b}^{\mathrm{D}}$ $(\forall i, b)$ and can therefore be neglected. A straightforward method to avoid the hardware-inefficient task of computing transcendental functions in (4.26) is to set $\tilde{K}_i = 0$ in the computation of (4.25). This can, however, lead to branch metrics that are not necessarily non-negative, which would inhibit pruning of the search tree. On the other hand, modifying the DIs in (4.17) by setting

$$|e_i| \triangleq \frac{1}{N_{\mathrm{o}}} \left| \tilde{y}_i - \sum_{j=i}^{M_{\mathrm{T}}} R_{i,j} s_j \right|^2 + K_i - \sum_{b=1}^{Q} \frac{1}{2} x_{i,b} L_{i,b}^{\mathrm{A}} \tag{4.27}$$

with $K_i = \sum_{b=1}^{Q} \frac{1}{2} \left| L_{i,b}^{\mathrm{A}} \right|$ also avoids computing transcendental functions while guaranteeing that, thanks to $\left| L_{i,b}^{\mathrm{A}} \right| - x_{i,b} L_{i,b}^{\mathrm{A}} \geq 0$ $(\forall i, b)$, the so obtained branch metrics are non-negative. Furthermore, as $\tilde{K}_i \geq K_i$, using the modified DIs (often significantly) reduces the (tree-search) complexity compared to that implied by (4.17) using (4.25) and, thanks to (4.14), still yields max-log-optimal LLRs. The reason for complexity reduction when using the modified DIs (4.27) lies in the modified prior term being bias-free, i.e.,

$$\min_{s_i \in \mathcal{O}} \left\{ K_i - \sum_{b=1}^{Q} \frac{1}{2} x_{i,b} L_{i,b}^{\mathrm{A}} \right\} = 0, \quad \forall i, \tag{4.28}$$

which directly leads to tight tree pruning using the standard pruning criterion in (4.18) and hence, avoids explicit evaluation of (4.22).

Note that in [30, Eq. 9], the prior term (4.25) was approximated

for $\left|L_{i,b}^{\mathrm{A}}\right| > 2$ $(b = 1, \ldots, Q)$ as

$$-\log \mathrm{P}[s_i] \approx \sum_{b=1}^{Q} \frac{1}{2}\left(\left|L_{i,b}^{\mathrm{A}}\right| - x_{i,b} L_{i,b}^{\mathrm{A}}\right)$$

which corresponds exactly to what was done here in order to arrive at (4.27). It is important, though, to realize that using the modified DIs (4.27) does *not* lead to an approximation of (4.14), as only differences are considered in the intrinsic max-log LLR computation and the neglected $\log(\cdot)$-term does not depend on $x_{i,b}$.

4.2 Single Tree-Search Sphere Decoding

Computing the intrinsic max-log LLRs in (4.14) requires to determine λ^{MAP} and the metrics $\lambda_{i,b}^{\overline{\mathrm{MAP}}}$ associated with the counter-hypotheses. For given i and b, $\lambda_{i,b}^{\overline{\mathrm{MAP}}}$, is obtained by traversing only those parts of the search tree that have leaves in $\mathcal{X}_{i,b}^{\left(x_{i,b}^{\overline{\mathrm{MAP}}}\right)}$. The quantities λ^{MAP} and $\lambda_{i,b}^{\overline{\mathrm{MAP}}}$ can, in principle, be computed using the sphere decoder based on the repeated tree-search (RTS) approach described in [125]. The RTS strategy results, however, in redundant computations as (often significant) parts of the search tree are revisited during the RTS steps required to determine $\lambda_{i,b}^{\overline{\mathrm{MAP}}}$ for all i, b. Following the STS paradigm, we note that *efficient* computation of $L_{i,b}^{\mathrm{D}}$ $(\forall i, b)$ requires that every node in the tree be visited *at most* once. This can be achieved by searching for the MAP solution and computing the metrics $\lambda_{i,b}^{\overline{\mathrm{MAP}}}$ $(\forall i, b)$ *concurrently* while ensuring that the subtree originating from a given node in the tree is pruned if searching that subtree can not lead to an update of either λ^{MAP} or at least one of the $\lambda_{i,b}^{\overline{\mathrm{MAP}}}$. The main idea underlying SISO STS-SD presented in this chapter is to *directly* compute the extrinsic LLRs $L_{i,b}^{\mathrm{E}}$ through a tree search, rather than computing $L_{i,b}^{\mathrm{D}}$ first and then evaluating $L_{i,b}^{\mathrm{E}} = L_{i,b}^{\mathrm{D}} - L_{i,b}^{\mathrm{A}}$ $(\forall i, b)$.

Due to the large dynamic range of LLRs, fixed-point detector implementations need to constrain the magnitude of the LLR values. Evidently, clipping of the LLR magnitude leads to a performance

degradation in terms of error rate. It was noted in [129] that incorporating LLR clipping into the RTS algorithm is effective in terms of reducing the complexity. Hence, we incorporate LLR clipping into the tree-search procedure of the SISO STS-SD, which will allow us to tune the MIMO detection algorithm in terms of complexity versus performance by adjusting the clipping parameter. In the SISO case, we are ultimately interested in the *extrinsic* LLRs $L_{i,b}^{\mathrm{E}}$ and clipping should therefore ensure that $\left|L_{i,b}^{\mathrm{E}}\right| \leq L_{\max}$ $(\forall i, b)$, where $L_{\max}$ is the LLR clipping parameter. It is therefore sensible to ask whether clipping of the *extrinsic* LLRs can be built directly into the tree search. The answer is in the affirmative and the corresponding solution is described below. We start by writing the extrinsic LLRs as

$$L_{i,b}^{\mathrm{E}} = \begin{cases} \Lambda_{i,b}^{\overline{\mathrm{MAP}}} - \lambda^{\mathrm{MAP}}, & x_{i,b}^{\mathrm{MAP}} = +1 \\ \lambda^{\mathrm{MAP}} - \Lambda_{i,b}^{\overline{\mathrm{MAP}}}, & x_{i,b}^{\mathrm{MAP}} = -1 \end{cases} \tag{4.29}$$

where the quantities

$$\Lambda_{i,b}^{\overline{\mathrm{MAP}}} = \begin{cases} \lambda_{i,b}^{\overline{\mathrm{MAP}}} - L_{i,b}^{\mathrm{A}}, & x_{i,b}^{\mathrm{MAP}} = +1 \\ \lambda_{i,b}^{\overline{\mathrm{MAP}}} + L_{i,b}^{\mathrm{A}}, & x_{i,b}^{\mathrm{MAP}} = -1 \end{cases} \tag{4.30}$$

will be referred to as the *extrinsic metrics*. For the following developments it will be convenient to define the function $f(\cdot)$ that transforms an intrinsic metric λ with associated a priori LLR L^{A} and binary label x to an extrinsic metric Λ according to

$$\Lambda = f\big(\lambda, L^{\mathrm{A}}, x\big) = \begin{cases} \lambda - L^{\mathrm{A}}, & x = +1 \\ \lambda + L^{\mathrm{A}}, & x = -1. \end{cases} \tag{4.31}$$

With this notation, we can rewrite (4.30) more compactly as

$$\Lambda_{i,b}^{\overline{\mathrm{MAP}}} = f\Big(\lambda_{i,b}^{\overline{\mathrm{MAP}}}, L_{i,b}^{\mathrm{A}}, x_{i,b}^{\mathrm{MAP}}\Big).$$

The inverse function of (4.31) transforms an extrinsic metric Λ to an intrinsic metric λ and is given by

$$\lambda = f^{-1}\big(\Lambda, L^{\mathrm{A}}, x\big) = \begin{cases} \Lambda + L^{\mathrm{A}}, & x = +1 \\ \Lambda - L^{\mathrm{A}}, & x = -1. \end{cases} \tag{4.32}$$

We emphasize that the tree-search algorithm described in the following produces the extrinsic LLRs $L_{i,b}^{\mathrm{E}}$ $(\forall i, b)$ in (4.29) rather than the intrinsic ones in (4.14).

4.2.1 List Administration

The main idea of the SISO-STS paradigm is to search the subtree originating from a given node only if the result can lead to an update of either λ^{MAP} or of at least one of the $\Lambda_{i,b}^{\overline{\mathrm{MAP}}}$. To this end, the SD algorithm needs to maintain a list containing the current MAP hypothesis $\mathbf{x}^{\mathrm{MAP}}$, the corresponding metric λ^{MAP}, and all QM_{T} extrinsic metrics $\Lambda_{i,b}^{\overline{\mathrm{MAP}}}$. The algorithm is initialized with $\lambda^{\mathrm{MAP}} = \Lambda_{i,b}^{\overline{\mathrm{MAP}}} = \infty$ and $x_{i,b}^{\mathrm{MAP}} = 1$ $(\forall i, b)$. Whenever a leaf node with corresponding label $\mathbf{x}$ has been reached, the detector distinguishes between two cases:

i) MAP hypothesis update If $d(\mathbf{x}) < \lambda^{\mathrm{MAP}}$, a new MAP hypothesis has been found. First, all extrinsic metrics $\Lambda_{i,b}^{\overline{\mathrm{MAP}}}$ for which $x_{i,b} = \overline{x_{i,b}^{\mathrm{MAP}}}$ are updated according to

$$\Lambda_{i,b}^{\overline{\mathrm{MAP}}} \leftarrow f\Big(\lambda^{\mathrm{MAP}}, L_{i,b}^{\mathrm{A}}, x_{i,b}^{\overline{\mathrm{MAP}}}\Big)$$

followed by the updates $\lambda^{\mathrm{MAP}} \leftarrow d(\mathbf{x})$ and $\mathbf{x}^{\mathrm{MAP}} \leftarrow \mathbf{x}$. In other words, for each bit in the MAP hypothesis that is changed in the update process, the metric associated with the *former* MAP hypothesis becomes the extrinsic metric of the *new* counter-hypothesis.

ii) Extrinsic metric update In the case where $d(\mathbf{x}) > \lambda^{\mathrm{MAP}}$, only extrinsic metrics corresponding to counter-hypotheses might be updated. For each $i = 1, \ldots, M_{\mathrm{T}}$, $b = 1, \ldots, Q$ with $x_{i,b} = \overline{x_{i,b}^{\mathrm{MAP}}}$ and $f\Big(d(\mathbf{x}), L_{i,b}^{\mathrm{A}}, x_{i,b}^{\mathrm{MAP}}\Big) < \Lambda_{i,b}^{\overline{\mathrm{MAP}}}$, the SISO STS-SD algorithm performs the update

$$\Lambda_{i,b}^{\overline{\mathrm{MAP}}} \leftarrow f\big(d(\mathbf{x}), L_{i,b}^{\mathrm{A}}, x_{i,b}^{\mathrm{MAP}}\big)\,. \tag{4.33}$$

4.2.2 Extrinsic LLR Clipping

In order to ensure that the extrinsic LLRs delivered by the algorithm indeed satisfy $\big|L_{i,b}^{E}\big| \leq L_{\max}$ $(\forall i, b)$, the following update rule

$$\Lambda_{i,b}^{\overline{\mathrm{MAP}}} \leftarrow \min\Big\{\Lambda_{i,b}^{\overline{\mathrm{MAP}}}, \lambda^{\mathrm{MAP}} + L_{\max}\Big\}, \quad \forall i, b \tag{4.34}$$

has to be applied after carrying out the steps in Case i) of the list administration procedure described in Section 4.2.1. Figure 4.2 illustrates the principle of extrinsic LLR clipping. The search for counter-hypotheses associated with extrinsic metrics is constrained to a hypersphere of radius $r = \sqrt{\lambda^{\mathrm{MAP}} + L_{\max}}$ around the (transformed) received signal vector $\tilde{\mathbf{y}}$. In Section 4.6.3, it will be demonstrated numerically that incorporating the constraint $|L_{i,b}^{\mathrm{E}}| \leq L_{\max}$ directly into the tree search significantly reduces complexity. We emphasize that for $L_{\max} = \infty$, the detector attains max-log optimal SISO performance, whereas for $L_{\max} = 0$, the LLRs satisfy $L_{i,b}^{\mathrm{E}} = 0$ and the hard-output MAP solution (4.12) is obtained.

4.2.3 The Tree-Pruning Criterion

Consider the node $\mathbf{s}^{(i)}$ on level i corresponding to the label bits $x_{j,b}$ $(j = i, \ldots, M_{\mathrm{T}},\ \ b = 1, \ldots, Q)$. Assume that the subtree originating from this node and corresponding to the label bits $x_{j,b}$ $(j = 1, \ldots, i-1,\, b = 1, \ldots, Q)$ has not been expanded yet. The tree-pruning criterion for the node $\mathbf{s}^{(i)}$ along with its subtree is compiled from two sets, defined as follows:

1) The bits in the partial label $\mathbf{x}^{(i)}$ (corresponding to the node $\mathbf{s}^{(i)}$) are compared with the corresponding bits in the label of the current MAP hypothesis. All extrinsic metrics $\Lambda_{i,b}^{\overline{\mathrm{MAP}}}$ with $x_{i,b} = \overline{x_{i,b}^{\mathrm{MAP}}}$ found in this comparison, may be affected when searching the subtree originating from $\mathbf{s}^{(i)}$. As $d\big(\mathbf{x}^{(i)}\big)$ is an intrinsic metric, the extrinsic metrics $\Lambda_{i,b}^{\overline{\mathrm{MAP}}}$ need to be mapped to intrinsic metrics according to (4.32). The resulting set of *intrinsic* metrics, which may be affected by an update, is given by

$$\mathcal{A}_1\Big(\mathbf{x}^{(i)}\Big) = \Big\{ f^{-1}\Big(\Lambda_{j,b}^{\overline{\mathrm{MAP}}}, L_{j,b}^{\mathrm{A}}, x_{j,b}^{\mathrm{MAP}}\Big) \;\Big|\; \big(j \geq i, \forall b\big) \wedge \Big(x_{j,b} = \overline{x_{j,b}^{\mathrm{MAP}}}\Big) \Big\}.$$

2) The extrinsic metrics $\Lambda_{j,b}^{\overline{\mathrm{MAP}}}$ for $j = 1, \ldots, i-1,\ b = 1, \ldots, Q$ corresponding to the counter-hypotheses in the subtree of $\mathbf{s}^{(i)}$ may be affected as well. Correspondingly, we define

$$\mathcal{A}_2\left(\mathbf{x}^{(i)}\right) = \left\{ f^{-1}\left(\Lambda_{j,b}^{\overline{\mathrm{MAP}}}, L_{j,b}^{\mathrm{A}}, x_{j,b}^{\mathrm{MAP}}\right) \,\middle|\, j < i, \forall b \right\}.$$

The set of intrinsic metrics which may be affected during the search in the subtree originating from node $\mathbf{s}^{(i)}$ is given by

$$\mathcal{A}\left(\mathbf{x}^{(i)}\right) = \{a_l\} = \mathcal{A}_1\left(\mathbf{x}^{(i)}\right) \cup \mathcal{A}_2\left(\mathbf{x}^{(i)}\right).$$

The node $\mathbf{s}^{(i)}$ along with its subtree is pruned if the corresponding PD $d\left(\mathbf{x}^{(i)}\right)$ satisfies the tree-pruning criterion

$$d\left(\mathbf{x}^{(i)}\right) > \max_{a_l \in \mathcal{A}\left(\mathbf{x}^{(i)}\right)} a_l.$$

This tree-pruning criterion ensures that a given node and the entire subtree originating from that node are explored only if this could lead to an update of either λ^{MAP} or of at least one of the extrinsic metrics $\Lambda_{i,b}^{\overline{\mathrm{MAP}}}$. Note that λ^{MAP} does not appear in the set $\mathcal{A}\left(\mathbf{x}^{(i)}\right)$, as the update criteria given in Section 4.2.1 ensure that λ^{MAP} is always smaller than or equal to all intrinsic metrics associated with the counter-hypotheses.

4.3 Channel Matrix Preprocessing

In this section, we describe how performing the QR-decomposition (QRD) on a column-sorted and regularized version of the channel matrix $\mathbf{H}$ in combination with compensation of self-interference in the LLRs —caused by channel-matrix regularization— carried out directly in the SISO STS-SD, can result in a significant complexity reduction at negligible performance loss.

4.3.1 Column-Sorting and Regularization of the Channel Matrix

Methods for column-sorting and regularization of the channel matrix $\mathbf{H}$ performed on the basis of the received symbol vector $\mathbf{y}$ have been discussed in, e.g., [130, 131]. Unfortunately, such techniques require QRD on symbol-vector rate, which leads to a significant computational burden. In contrast, column-sorting and regularization based

solely on the channel matrix $\mathbf{H}$ (and possibly on the noise variance) require QRDs only when the channel state changes, which entails a significantly smaller computational burden.

Column-Sorting

The complexity of SD can be reduced (often significantly) by performing the QRD on a column-sorted version of $\mathbf{H}$ rather than on $\mathbf{H}$ directly, i.e., by computing $\mathbf{HP} = \mathbf{QR}$, where $\mathbf{P}$ is an $M_{\mathrm{T}} \times M_{\mathrm{T}}$ permutation matrix [22]. Reduction in terms of complexity is obtained if levels closer to the root correspond to main-diagonal entries of $\mathbf{R}$ with larger magnitude, or equivalently, to spatial streams with higher effective SNR. A corresponding computationally efficient heuristic was proposed in [54] and is referred to as sorted QRD (SQRD) in the following.

Regularization

A further reduction in terms of complexity—at the cost of slightly reduced performance—can be obtained by performing the tree search on a Tikhonov-regularized (and column-sorted) version of $\mathbf{H}$ according to [132]

$$\underbrace{\begin{bmatrix} \mathbf{H} \\ \alpha \mathbf{I}_{M_{\mathrm{T}}} \end{bmatrix}}_{\triangleq \overline{\mathbf{H}}} \mathbf{P} = \underbrace{\begin{bmatrix} \mathbf{Q}_a & \mathbf{Q}_c \\ \mathbf{Q}_b & \mathbf{Q}_d \end{bmatrix}}_{\triangleq \overline{\mathbf{Q}}} \underbrace{\begin{bmatrix} \tilde{\mathbf{R}} \\ \mathbf{0}_{M_{\mathrm{R}} \times M_{\mathrm{T}}} \end{bmatrix}}_{\triangleq \overline{\mathbf{R}}} \tag{4.35}$$

where $\alpha \in \mathbb{R}$ is a suitably chosen regularization parameter. Here, $\overline{\mathbf{R}}$ and $\overline{\mathbf{Q}}$ are partitioned such that $\tilde{\mathbf{R}}$, $\mathbf{Q}_a$, $\mathbf{Q}_b$, $\mathbf{Q}_c$, and $\mathbf{Q}_d$ are of dimension $M_{\mathrm{T}} \times M_{\mathrm{T}}$, $M_{\mathrm{R}} \times M_{\mathrm{T}}$, $M_{\mathrm{T}} \times M_{\mathrm{T}}$, $M_{\mathrm{R}} \times M_{\mathrm{R}}$, and $M_{\mathrm{T}} \times M_{\mathrm{R}}$, respectively. The computational complexity for regularized SQRD as compared to non-regularized SQRD is approximately 50% higher [133]. However, the QRD needs to be performed only if the channel matrix $\mathbf{H}$ changes, as opposed to the tree-search itself, which needs to be carried out at symbol-vector rate.

LLR computation (and MAP detection) based on regularized SQRD corresponds to replacing the modified input-output relation

in (4.1) by

$$\hat{\mathbf{y}} = \tilde{\mathbf{R}}\tilde{\mathbf{s}} + \tilde{\mathbf{n}} \tag{4.36}$$

where $\hat{\mathbf{y}} = \mathbf{Q}_a^H \mathbf{y}$, $\tilde{\mathbf{s}} = \mathbf{P}^T \mathbf{s}$, and $\tilde{\mathbf{n}} = -\alpha \mathbf{Q}_b^H \mathbf{s} + \mathbf{Q}_a^H \mathbf{n}$. The corresponding intrinsic (max-log) LLRs in (4.7) are obtained by pretending that the resulting noise $\tilde{\mathbf{n}}$ has the same statistics as $\mathbf{n}$, which leads to

$$\begin{aligned} \tilde{L}_{i,b}^{\mathrm{D}} \triangleq & \min_{\tilde{\mathbf{s}} \in \mathcal{X}_{i,b}^{(-1)}} \left\{ \frac{1}{N_{\mathrm{o}}} \left\| \hat{\mathbf{y}} - \tilde{\mathbf{R}}\tilde{\mathbf{s}} \right\|^2 - \log \mathrm{P}[\tilde{\mathbf{s}}] \right\} \\ & - \min_{\tilde{\mathbf{s}} \in \mathcal{X}_{i,b}^{(+1)}} \left\{ \frac{1}{N_{\mathrm{o}}} \| \hat{\mathbf{y}} - \tilde{\mathbf{R}}\tilde{\mathbf{s}} \|^2 - \log \mathrm{P}[\tilde{\mathbf{s}}] \right\} \end{aligned} \tag{4.37}$$

where $\mathrm{P}[\tilde{\mathbf{s}}] = \mathrm{P}[\mathbf{s}]$. The intrinsic LLRs $\tilde{L}_{i,b}^{\mathrm{D}}$ in (4.37) will, in general, only be approximations to the intrinsic max-log LLRs $L_{i,b}^{\mathrm{D}}$ in (4.7). This is a consequence of $\tilde{\mathbf{n}}$ no longer being i.i.d. circularly symmetric complex Gaussian distributed with variance N_{o} per complex entry, as it contains self-interference (i.e., it depends on $\mathbf{s}$) and $\mathbf{Q}_a$ is, in general, not unitary. Nevertheless, computing the covariance matrix of $\tilde{\mathbf{n}}$, by averaging over $\mathbf{n}$ and $\mathbf{s}$, allows to identify good choices for the regularization parameter α. By noting that $\mathbb{E}[\mathbf{s}\mathbf{s}^H] = E_{\mathrm{s}} \mathbf{I}_{M_{\mathrm{T}}}$, straightforward manipulations reveal that

$$\mathbf{K} = \mathbb{E}\big[\tilde{\mathbf{n}}\tilde{\mathbf{n}}^H\big] = \Big(\mathbf{R}\mathbf{R}^H\Big)^{-1} \alpha^2 \Big(E_{\mathrm{s}}\alpha^2 - N_{\mathrm{o}}\Big) + N_{\mathrm{o}} \mathbf{I}_{M_{\mathrm{T}}} \, .$$

Setting $\alpha = \sqrt{N_{\mathrm{o}}/E_{\mathrm{s}}}$, corresponds to MMSE regularization [55], results in $\mathbf{K} = N_o \mathbf{I}_{M_{\mathrm{T}}}$, and yields a good performance/complexity tradeoff. We emphasize, however, that setting $\alpha = \sqrt{N_{\mathrm{o}}/E_{\mathrm{s}}}$ will *not* render the effective NPI vector $\tilde{\mathbf{n}}$ Gaussian. In the remainder of the chapter, we denote the QR-decomposition in (4.35) with $\alpha = \sqrt{N_{\mathrm{o}}/E_{\mathrm{s}}}$ MMSE-SQRD and regularization will always refer to using MMSE-SQRD. Finally, we note that the LLRs in (4.37) need to be reordered after the detection stage to account for the permutation induced by $\mathbf{P}$.

4.3.2 Compensation of Self-Interference

Using the approximate (max-log) LLRs in (4.37) with $\alpha \neq 0$ instead of the exact max-log LLRs in (4.7) results in a performance loss. In order

to recover part of this performance loss, a method for the compensation of self-interference was developed in [134] for list-based MIMO detectors. The approach described in [134] can not be applied directly to SISO STS-SD. It turns out, however, that compensation of self-interference can be incorporated directly into the tree-search procedure. This leads to a noticeable performance improvement compared to using (4.37), while the corresponding increase in complexity is negligible (corresponding simulation results are shown in Section 4.6.3).

Compensation of Self-Interference

As shown in [134], the squared Euclidean distance $\left\|\underline{\mathbf{y}} - \underline{\mathbf{H}}\mathbf{s}\right\|^2$ with $\underline{\mathbf{y}} = \begin{bmatrix}\mathbf{y}^T & \mathbf{0}_{1\times M_{\mathrm{T}}}\end{bmatrix}^T$ can be expanded in two different ways according to

$$\left\|\underline{\mathbf{y}} - \underline{\mathbf{H}}\mathbf{s}\right\|^2 = \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2 + \alpha^2\|\mathbf{s}\|^2 \tag{4.38}$$

$$\left\|\underline{\mathbf{y}} - \underline{\mathbf{H}}\mathbf{s}\right\|^2 = \left\|\underline{\mathbf{Q}}^H\underline{\mathbf{y}} - \underline{\mathbf{R}}\mathbf{P}^T\mathbf{s}\right\|^2 = \left\|\hat{\mathbf{y}} - \tilde{\mathbf{R}}\tilde{\mathbf{s}}\right\|^2 + \left\|\mathbf{Q}_c^H\mathbf{y}\right\|^2 \tag{4.39}$$

where (4.39) is obtained by using (4.35). Equating the RHS terms of (4.38) and (4.39) and using $\|\mathbf{s}\|^2 = \|\tilde{\mathbf{s}}\|^2$ yields

$$\|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2 = \left\|\hat{\mathbf{y}} - \tilde{\mathbf{R}}\tilde{\mathbf{s}}\right\|^2 + \left\|\mathbf{Q}_c^H\mathbf{y}\right\|^2 - \alpha^2\|\tilde{\mathbf{s}}\|^2 \tag{4.40}$$

which allows us to conclude that the metric $\|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2$ contains a contribution that is independent of the symbol vectors, namely $\|\mathbf{Q}_c^H\mathbf{y}\|^2$, and a term caused by self-interference given by $-\alpha^2\|\tilde{\mathbf{s}}\|^2$. Since we use $\left\|\hat{\mathbf{y}} - \tilde{\mathbf{R}}\tilde{\mathbf{s}}\right\|^2$ (instead of the left-hand side of (4.40)) in the LLR computation (4.37), the two remaining RHS-terms in (4.40) must be compensated. As already observed in Section 4.1.4, constant terms (i.e., terms that are independent of $\mathbf{s}$) cancel out in the LLR computation (4.14) and can therefore be neglected without affecting the resulting LLRs, whereas the term $-\alpha^2\|\tilde{\mathbf{s}}\|^2$ does depend on $\mathbf{s}$ and therefore needs to be compensated. This is accomplished by computing the self-interference free (SIF) intrinsic max-log LLRs according

to [134]

$$\bar{L}_{i,b}^{\mathrm{D}} \triangleq \min_{\tilde{\mathbf{s}} \in \mathcal{X}_{i,b}^{(-1)}} \left\{ \frac{1}{N_{\mathrm{o}}} \left\| \hat{\mathbf{y}} - \tilde{\mathbf{R}}\tilde{\mathbf{s}} \right\|^2 - \frac{\alpha^2}{N_{\mathrm{o}}} \|\tilde{\mathbf{s}}\|^2 - \log \mathrm{P}[\tilde{\mathbf{s}}] \right\} - \min_{\tilde{\mathbf{s}} \in \mathcal{X}_{i,b}^{(+1)}} \left\{ \frac{1}{N_{\mathrm{o}}} \|\hat{\mathbf{y}} - \tilde{\mathbf{R}}\tilde{\mathbf{s}}\|^2 - \frac{\alpha^2}{N_{\mathrm{o}}} \|\tilde{\mathbf{s}}\|^2 - \log \mathrm{P}[\tilde{\mathbf{s}}] \right\}. \tag{4.41}$$

We emphasize, however, that (4.41) remains an approximation to (4.7) as the noise term $\tilde{\mathbf{n}}$ resulting from (4.36) is *not* i.i.d. circularly symmetric Gaussian distributed with variance N_{o} per complex entry.

Compensation in the SISO STS-SD Algorithm

In [134] it was suggested to compensate self-interference *after* the tree-search. For the SISO STS-SD algorithm described in Section 4.2, extrinsic LLRs are computed *only* on the basis of the MAP hypothesis $\mathbf{x}^{\mathrm{MAP}}$, its metric λ^{MAP}, and extrinsic metrics $\Lambda_{i,b}^{\overline{\mathrm{MAP}}}$ (see (4.29)). Compensation of self-interference according to (4.41) *after* carrying out the SISO STS-SD algorithm, would additionally require explicit knowledge of the symbol vectors $\mathbf{s} \in \mathcal{X}_{i,b}^{\left(\overline{x_{i,b}^{\mathrm{MAP}}}\right)}$, which is not available. Inspection of (4.41) suggests, however, that self-interference compensation may be incorporated into the tree-search procedure. Straightforward modification of the DIs in (4.27) to accomplish this would lead to the modified DIs

$$|\tilde{e}_i| = |e_i| - \frac{\alpha^2}{N_{\mathrm{o}}} |\tilde{s}_i|^2$$

which are, however, no longer guaranteed to be strictly non-negative. As in the tightening of the tree-pruning criterion described previously in Section 4.1.4, we recognize that symbol-vector-independent terms can be added to the DIs without loss of (max-log) optimality. Therefore, setting the DIs to

$$|\bar{e}_i| \triangleq |e_i| + m(\tilde{s}_i) \tag{4.42}$$

with the non-negative term

$$m(\tilde{s}_i) = \frac{\alpha^2}{N_{\mathrm{o}}} \left(\max_{s \in \mathcal{O}} |s|^2 - |\tilde{s}_i|^2 \right) \tag{4.43}$$

leads to the smallest possible non-negative DIs that compensate self-interference directly in the tree search. Note that adding non-negative terms to the DIs as done in (4.42) increases, in general, the (tree-search) complexity. Recall, however, that channel-matrix regularization itself almost always significantly reduces complexity [23], so that this increase, which is shown numerically in Section 4.6.3 to be marginal, is tolerable. In addition, it turns out that self-interference compensation recovers the performance loss due to channel-matrix regularization to a point where near-max-log optimal performance is achieved (see Section 4.6.3). In the case of constant-modulus symbol alphabets (e.g., BPSK or 4-QAM) we have $m(\tilde{s}_i) = 0$ ($i = 1, \ldots, M_{\mathrm{T}}$) and compensation of self-interference in the tree-search is not required (and hence, does not increase the complexity). We conclude by noting that the quantities $\max_{s_i \in \mathcal{O}} |s_i|^2$ can be pre-computed and hence, the additional computational complexity required to incorporate compensation of self-interference into the tree-search procedure is very small.

4.4 Run-Time Constraints

The complexity of the SISO STS-SD algorithm depends critically on the noise realization $\mathbf{n}$, the channel-matrix realization $\mathbf{H}$, the transmit-vector $\mathbf{s}$, and the a priori LLRs $L_{i,b}^{\mathrm{A}}$. As it will be discussed below, the often prohibitively high worst-case complexity of SD constitutes a problem in many practical application scenarios, as it inhibits realizing the throughput requirements of many of the available communication standards.

4.4.1 Issues with SD Complexity

As it was shown in [126, 135], the average complexity of ℓ^2-norm SD and ℓ^∞-norm SD grows exponentially in the number of transmit antennas. Moreover, if $\mathsf{SNR} \to 0$, the (hard-output) SESD visits at least all nodes down to and including the level just above the leaves, which corresponds (for $\mathbf{R} = \mathbf{I}_{M_{\mathrm{T}}}$) to

$$C = \frac{|\mathcal{O}|^{M_{\mathrm{T}}} - 1}{|\mathcal{O}| - 1}$$

visited nodes [25]. High complexity is caused by the fact that the noise can shift the transmit vector arbitrarily far away from the finite lattice that causes the minimum radius of SESD (corresponding to the Euclidean distance between the received vector and the ML solution) to get arbitrarily large. Hence, the complexity of SD can get very high, especially in the low-SNR regime.

Recently, it was shown in [136] for the Pohst-SD algorithm operating on the infinite-lattice [94, 95], that the complexity tail-behavior in i.i.d. Gaussian lattices is of Pareto-type, i.e.,

$$\mathrm{P}[C \geq L] \approx L^{-(M_{\mathrm{R}}-M_{\mathrm{T}}+1)}, \quad L \to \infty$$

where C stands for the complexity in number of visited nodes. Hence, the SD complexity is large with high probability. Simulation results have shown that the tail-complexity of SESD is similar to that of Pohst-SD in the infinite-lattice (but not necessarily Pareto-type).

4.4.2 Early-Termination and Scheduling

Due to the prohibitive worst-case complexity of SD, is of paramount importance that the SD algorithm's maximum complexity must be constrained to meet the practically important requirement of high throughput. This, in turn, leads to a constraint on the maximum detection effort or, equivalently, a constraint on the maximum number of nodes SD is allowed to visit. Clearly, this will prevent the detector from achieving MAP or max-log optimal SISO APP performance, in general. It is therefore important to find a way of imposing run-time constraints while keeping the resulting performance degradation at a minimum. In practice, it is highly desirable to have a smooth performance degradation as the run-time constraint becomes more stringent. In the following, we present corresponding solutions for the SISO STS-SD algorithm.

Maximum-First Scheduling

A straightforward way of enforcing a run-time constraint is to terminate the search, on a symbol vector by symbol vector basis, after a maximum number of visited nodes. The SISO STS-SD algorithm then returns the best solution found so far, i.e., the current MAP

hypothesis and the current extrinsic metrics. A better solution is to impose an aggregate run-time constraint of ND_{avg} visited nodes for an entire block of N symbol vectors [21].[6] The maximum number of visited nodes allocated to the detection of the kth symbol vector can be chosen according to the maximum-first (MF) scheduling strategy as [21, 22]

$$D_{\max}(k) = ND_{\text{avg}} - \sum_{i=1}^{k-1} D(i) - (N-k)M \tag{4.44}$$

for $k = 1, \ldots, N$, where $D(i)$ denotes the actual number of visited nodes for the ith symbol vector and M is a safety margin.

The main idea realized by the policy (4.44) is that detection of the kth symbol vector is allowed to use up all of the remaining complexity budget within the block of N symbol vectors up to $(N-k)M$ nodes, i.e., the parameter M determines that in decoding the remaining $N-k$ symbol vectors, we can afford a budget of at least M nodes per symbol vector. Setting $M = M_{\text{T}}$ and choosing $D_{\text{avg}} \geq M_{\text{T}}$ (what is used in the remainder of this thesis), ensures that for each of the remaining $N-k$ symbol vectors at least the hard-output successive interference cancellation (SIC) solution is found. We emphasize that, under run-time constraints and no LLR clipping (i.e., $L_{\max} = \infty$), there may be LLRs at the end of the decoding process that have not been updated from their initial value of ∞ and hence need to be set to $L_{\max}$. The performance of MF scheduling in combination with SISO STS-SD is shown in Section 4.6.5.

FIFO Scheduling

In many practical receiver implementations, the receive-vectors $\mathbf{y}$ are stored in a memory, which is able to deliver these vectors to the MIMO detector at a certain maximum rate (i.e., limited by the memory bandwidth). The scheduling method described below will be referred to as FIFO scheduling and extends the method described in [21].

The principle of FIFO scheduling is illustrated in Figure 4.5 and consists of a first-in first-out (FIFO) unit (with F storage entries),

[6] In a MIMO-OFDM system, N would be the number of OFDM tones.

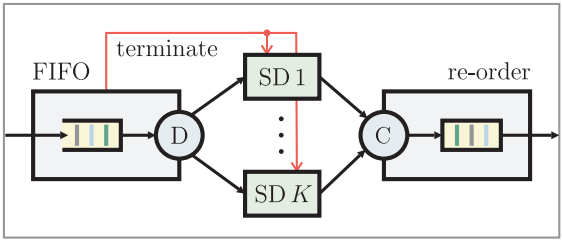


Figure 4.5: FIFO scheduling with K parallel SD-cores.

a distribution unit (D), K parallel operating SD units (e.g., SISO STS-SD units), a collector unit (C), and a re-ordering buffer. We assume that the K parallel SD units follow the one-node-per-cycle paradigm [33], i.e., the number of clock cycles required to detect a symbol vector is equal to the complexity. We furthermore consider the scenario where for every D_{avg} clock cycle, a new receive-vector arrives at the FIFO (the arrival rate is $1/D_{\text{avg}}$ receive-vectors per clock cycle).

The FIFO scheduling algorithm is as follows: Consider the case when the FIFO is empty and a receive-vector $\mathbf{y}$ arrives at the FIFO. In this situation, the distribution unit passes this vector immediately to an idle SD unit. If all of the K units are busy, the receive-vector is stored in the FIFO. In the case where the FIFO is full[7], the SD unit that used up the maximum complexity so far is terminated early. This strategy enables to immediately decode a new receive-vector in this SD unit. Moreover, it ensures that the arrived receive-vector can be stored in the FIFO and does not need to be discarded. If a SD unit completes detection, the collector stores the produced LLRs in a memory. Since it may happen that the order-of-arrival (at the FIFO) is not equal that the order-of-completion, re-ordering of the LLRs is performed in the collector unit.

The presented FIFO scheduling approach ensures that each receive-vector can be decoded with at least $D_{\text{avg}}K$ visited nodes, i.e., increasing the number of parallel SD instances also increases the minimum available complexity available for each detection task. The FIFO-size

[7]This many happen if the complexity of all K tasks is much larger than D_{avg}.

F determines the amount of "averaging" that can be exploited, i.e., if F is large it will be less likely that the FIFO gets full (for the case when the arrival-rate is less than the service-rate) and hence, the amount of early-terminated SD runs is reduced. However, the (worst-case) latency increases linearly in F. Therefore, the overall performance of SD using FIFO scheduling depends critically on the number of parallel SD units K as well as on F.

4.5 LLR Correction

The max-log approximation, channel-matrix regularization, and other complexity-reducing mechanisms, such as early termination of the tree-search, lead to LLRs that are approximations to the true LLRs in (2.22). However, channel decoders rely on exact LLRs in order to achieve optimum performance. In the following, we present a post-processing method for correcting approximate LLRs resulting from sub-optimal detectors. This method is based on ideas developed in [137, 138] and is able to (often significantly) improve the performance in (iterative) MIMO decoders while requiring low additional computational complexity.

4.5.1 The Basic Idea

We start by defining (or recalling the definitions of) the following objects (see Figure 4.6):

- the *effective channel* with the binary-valued inputs $x_{i,b}$ and the associated a priori LLRs $L^{\mathrm{A}}_{i,b}$ and outputs given by the (possibly approximated) extrinsic LLRs $L^{\mathrm{E}}_{i,b}$.
- the *physical MIMO channel* with input $\mathbf{s}$ and output $\mathbf{y}$.
- the *soft-input soft-output MIMO detector* with inputs $\mathbf{y}$ and $L^{\mathrm{A}}_{i,b}$ and outputs $L^{\mathrm{E}}_{i,b}$.
- the *LLR correction unit* (see Figure 4.6) computes corrected extrinsic LLRs $L^{\mathrm{C}}_{i,b}$ based on (approximated) extrinsic LLRs $L^{\mathrm{E}}_{i,b}$

and on side information $\mathcal{Z}$, by applying an LLR correction function

$$L^{\mathrm{C}}_{i,b} = g\Big(L^{\mathrm{E}}_{i,b}, \mathcal{Z}\Big). \tag{4.45}$$

- the *side information* $\mathcal{Z}$ is, for example, obtained from the (instantaneous) receive SNR, the singular values of the channel matrix $\mathbf{H}$, and from knowledge of whether the soft-input soft-output MIMO detector was terminated prematurely [23].

For the LLR correction function to yield valid LLRs, we define

$$g\Big(L^{\mathrm{E}}_{i,b}, \mathcal{Z}\Big) = \log\left(\frac{\mathrm{P}\Big[x_{i,b} = +1 \,|\, L^{\mathrm{E}}_{i,b}, \mathcal{Z}\Big]}{\mathrm{P}\Big[x_{i,b} = -1 \,|\, L^{\mathrm{E}}_{i,b}, \mathcal{Z}\Big]}\right). \tag{4.46}$$

Just like the LLRs in (2.22) are computed based on the received vector $\mathbf{y}$ and the channel state $\mathbf{H}$, the corrected LLRs are computed based on the (approximated) extrinsic LLRs $L^{\mathrm{E}}_{i,b}$ and the side information $\mathcal{Z}$. The formulation (4.45) and (4.46) entails that $L^{\mathrm{C}}_{i,b}$ depends only on $L^{\mathrm{E}}_{i,b}$ (and $\mathcal{Z}$) rather than on all extrinsic (approximated) LLR values $L^{\mathrm{E}}_{i,b}$ (for all i, b). Making the correction function depend on other LLR values (besides the one to be corrected) would certainly improve the correction performance, but at the same time also dramatically increase the computational effort for LLR correction, as it will become clear in the discussion of the numerical procedure for LLR correction proposed below.

The main idea is now—depending on the mechanisms used to approximate the extrinsic LLRs (e.g., the max-log approximation, channel-matrix regularization, early termination of the tree-search)—to extract suitable side information $\mathcal{Z}$. To see that this is non-trivial and the problem is multi-faceted, simply note that the set of all possible channel matrices $\mathbf{H}$ is a continuum of $M_{\mathrm{R}} \times M_{\mathrm{T}}$ complex-valued matrices. This continuum will be absorbed in $\mathcal{Z}$ through, e.g., the singular values or the rank of $\mathbf{H}$. We emphasize that in practice, LLR correction requires that the set $\mathcal{Z}$ be finite. In addition, the individual entries of $\mathcal{Z}$ must have finite cardinality as well. Hence, continuous-valued quantities, such as, e.g., the SNR or singular values, must be

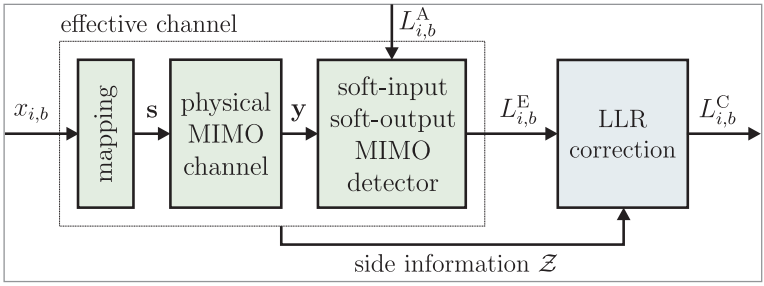


Figure 4.6: LLR correction post-processes the LLRs resulting from the effective channel using side information $\mathcal{Z}$.

suitably quantized. The total number of different instances of the side information $\mathcal{Z}$ is denoted by Z in the following.

4.5.2 Computation of LLR Correction Functions

Once we have chosen $\mathcal{Z}$, the LLR correction function in (4.46) is (in principle) obtained from the conditional probabilities $\mathrm{P}\left[x_{i,b} = \pm 1 \,|\, L^{\mathrm{E}}_{i,b}, \mathcal{Z}\right]$. Analytical expressions for correction functions seem very hard to obtain (even for simple examples such as for Hagenauer's approximation to the box function [137]). We next propose an approach for numerically computing (approximations to) the LLR correction function in (4.46).

First, the range of the LLRs to be corrected needs to be constrained (motivated, e.g., by the use of LLR clipping) to

$$L^{\mathrm{E}}_{i,b} \in [-L_{\max}, +L_{\max}].$$

This interval is then divided into K equally-sized bins such that the kth bin corresponds to

$$\mathcal{B}_k = \left[-L_{\max} + k\frac{2L_{\max}}{K}, -L_{\max} + (k+1)\frac{2L_{\max}}{K}\right)$$

for $k = 0, \ldots, K-1$. Then, the histogram

$$p_k(\mathcal{Z}) = \mathrm{P}\left[x_{i,b} = +1 \,|\, L^{\mathrm{E}}_{i,b} \in \mathcal{B}_k, \mathcal{Z}\right], \quad k = 0, \ldots, K-1 \qquad (4.47)$$

can be computed by performing Monte-Carlo simulations (averaged over noise and channel realizations) with randomly generated bits $x_{i,b}$. For each $L_{i,b}^{\mathrm{E}}$ and a given instance of $\mathcal{Z}$, the (approximated) LLR correction function is obtained by linear interpolation between the base points

$$\left(-L_{\max} + \left(k + \frac{1}{2}\right)\frac{2L_{\max}}{K} \, , \, \log\left(\frac{p_k(\mathcal{Z})}{1 - p_k(\mathcal{Z})}\right)\right). \tag{4.48}$$

We emphasize that for each instance of $\mathcal{Z}$, in general, a different LLR correction function is obtained. Note that the LLRs resulting from (4.48) may have a magnitude that is larger than $L_{\max}$ (see Section 4.6.5). The corrected LLRs can be clipped again to satisfy $|L_{i,b}^{\mathrm{C}}| \leq L_{\max,\mathrm{c}}$, where $L_{\max,\mathrm{c}} \geq L_{\max}$, thereby limiting the dynamic range of LLRs (rather than performing LLR clipping for complexity reduction and tuning of the detector as done so far).

The computational complexity needed to compute the histogram in (4.47) and the corresponding storage requirements depend critically on the number of bins K and on the total number of different instances of the side information $\mathcal{Z}$ given by Z. In particular, ZK histogram values need to be stored and hence, it is important to keep both Z and K small. Application of the LLR correction function itself amounts to simple table look-up operations followed by linear interpolation, which can be performed at very low computational complexity.

4.5.3 An Example

We next discuss an example that illustrates the impact (and importance) of LLR correction. Consider the case where early termination with MF scheduling is used (see Section 4.4.2). Now, if early termination happens before the extrinsic metric $\Lambda_{i,b}^{\overline{\mathrm{MAP}}}$ was updated from its initial value ∞, the corresponding LLR satisfies $\left|L_{i,b}^{\mathrm{E}}\right| = L_{\max}$ as only LLR clipping according to (4.34) was performed. Hence, early termination may result in LLRs with a higher reliability than they would actually have if no complexity constraints were imposed. This calls for LLR correction with the goal of reducing the magnitude of such LLRs. Consequently, the side information set $\mathcal{Z}$ should contain

a binary-valued state variable, which indicates whether early termination occurred or not (i.e., the magnitude should be reduced if early termination happened). Corresponding numerical results are provided in Section 4.6.5.

4.6 Simulation Results

Unless explicitly stated otherwise, all simulation results are for simulation environment similar to that described in Section 2.2.2. We consider a convolutionally encoded (rate $R = 1/2$, generator polynomials $[133_o\ 171_o]$, and constraint length 7) iterative MIMO-OFDM system with $M_{\mathrm{T}} = M_{\mathrm{R}} = 4$, 16-QAM constellation $\mathcal{O}$ with Gray labeling, 64 OFDM tones, and TGn type C channel model [12]. Channel decoding is performed using a max-log BCJR algorithm [34]. One frame consists of 1024 randomly interleaved (across space and frequency) bits corresponding to one (spatial) OFDM symbol and we assume that the bits $x_{i,b}$ $(\forall i, b)$ are statistically independent. The SNR is per receive antenna and the SNR values specified in the figures are in decibels (dBs). The number of iterations I is the number of times the soft-input soft-output MIMO detector (and the SISO channel decoder) are used, i.e., $I = 1$ corresponds to soft-output SD. The LLR clipping parameters shown in the simulation results correspond to *normalized* LLR clipping parameters according to $L_{\max}/N_{\mathrm{o}}$.

4.6.1 Impact of the Max-Log Approximation

Figure 4.7 compares the error-rate performance of exact APP detection in (2.22) to that of the max-log approximation in (4.7) using the (optimal) sum-product BCJR algorithm for channel decoding. We can see that the max-log approximation entails a small performance loss (between 0.2 dB and 0.6 dB SNR for 1% FER), which increases with growing I. However, the loss associated with the max-log approximation can be considered to be low, in particular in the light of the prohibitive computational complexity associated with exact APP detection.

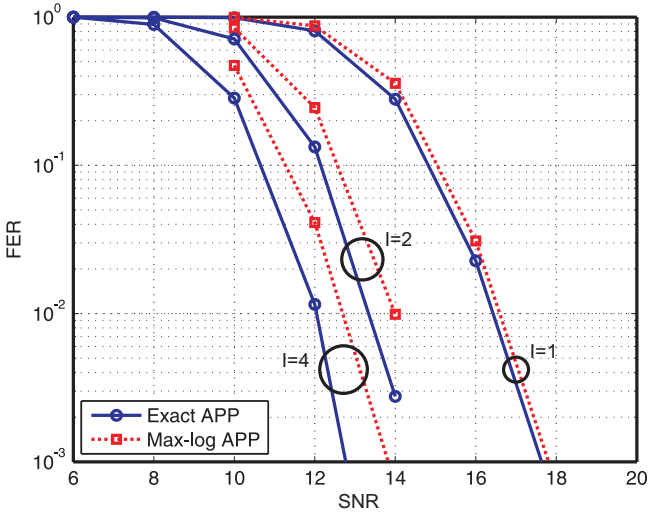


Figure 4.7: Performance impact of the max-log approximation to MIMO APP detection.

4.6.2 Tightening of the Tree-Pruning Criterion

Impact of the Euclidean-Distance Term

The goal of the simulation results shown in Table 4.1 is to quantify the impact of the Euclidean distance term $\frac{1}{N_0}\left|\tilde{y}_i - \sum_{j=i}^{M_T} R_{i,j} s_j\right|^2$ in the bias (4.19) on the complexity reduction obtained by tightening the tree-pruning criterion according to (4.21). To this end, we set the prior term to zero, i.e., $\log \mathrm{P}[\mathbf{s}] = 0$, and compare the complexity resulting from the tightened tree-pruning criterion to that of the standard tree-pruning criterion (denoted by "std." in Table 4.1) given in (4.18). We observe that the complexity reduction obtained by tightening of the tree-pruning criterion based on the Euclidean distance-term only, is marginal, in particular in the light of the prohibitive effort required to compute (4.19).

Table 4.1: Average complexity reduction obtained by tightening of the tree-pruning criterion based on the Euclidean-distance term only.

SNR	$L_{\max}$	std. [nodes]	tight [nodes]	reduction
10 dB	0.0125	34.9	34.4	1.4%
	∞	328.3	327.8	0.2%
20 dB	0.0125	11.0	10.8	1.8%
	∞	227.2	227.0	0.1%

Table 4.2: Average complexity reduction obtained by tightening of the tree-pruning criterion based on the prior term only.

SNR	I	$L_{\max}$	std. [nodes]	tight [nodes]	reduction
10 dB	1	0.0125	1890.4	34.9	98.2%
		∞	2440.2	328.3	86.5%
	2	0.0125	1630.6	43.4	97.3%
		∞	2148.4	406.6	81.1%
20 dB	1	0.0125	1914.7	11.0	99.4%
		∞	2397.0	227.2	90.5%
	2	0.0125	1228.7	6.2	99.5%
		∞	361.9	132.4	65.9%

Impact of the Prior Term

Next, we start with uniform priors, i.e., $L_{i,b}^{\mathrm{A}} = 0$ $(\forall i, b)$ for the first iteration, and perform tightening of the tree-pruning criterion according to (4.23). Table 4.2 shows that removing the bias $|p_i|$ in (4.22) leads to a dramatic reduction in terms of complexity, ranging from 65.9% to 99.5%. Furthermore, we can see that the impact on complexity reduction in the second iteration ($I = 2$) is less pronounced (but still significant) than in the first iteration. This behavior can be explained by noting that for $I = 1$ the priors satisfy $L_{i,b}^{A} = 0$, which leads to the largest possible values for $|p_i|$, $i = 1, \ldots, M_{\mathrm{T}}$. We note that, in general, the impact on complexity reduction is further reduced with increasing I.

We can now conclude that removing the Euclidean-distance com-

ponent of the bias term (4.19) is not worth the effort. In contrast, tightening of the tree-pruning criterion based on the prior only (4.23) leads to a significant complexity reduction and requires no additional computational complexity if the individual bits $x_{i,b}$ $(\forall i, b)$ are statistically independent (see (4.28) in Section 4.1.4). In the remainder of this thesis, we always employ tightening of the tree-pruning criterion according to (4.23).

4.6.3 Performance/Complexity Tradeoffs

The performance/complexity tradeoffs discussed next and quantified in Figs. 4.8 – 4.11, 4.13, and 4.14 refer to the *cumulative* (tree-search) complexity in terms of the total number of nodes visited (averaged over independent channel, noise, and data realizations) for SISO detection over I iterations, designated as "average complexity" from now on. The computational complexity incurred by channel decoding is ignored in the following. The minimum SNR required to achieve a given frame error rate (FER) is referred to as the "SNR operating point" for that FER in the remainder of this thesis.

Impact of LLR clipping

From Figure 4.8, we can conclude that LLR clipping allows for a smooth performance/complexity tradeoff, adjustable through a single parameter, namely the LLR clipping parameter $L_{\max}$. Note that for a fixed SNR operating point, the minimum complexity is not necessarily achieved by maximizing the number of iterations. The performance corresponding to the case where clipping of the extrinsic LLRs is performed *after* the tree search, i.e., LLR clipping is not incorporated into the tree search, is that obtained for $L_{\max} = \infty$. We can therefore conclude that incorporating LLR clipping into the tree search is of paramount importance as it reduces the complexity substantially and renders the detector easily adjustable in terms of performance versus complexity.

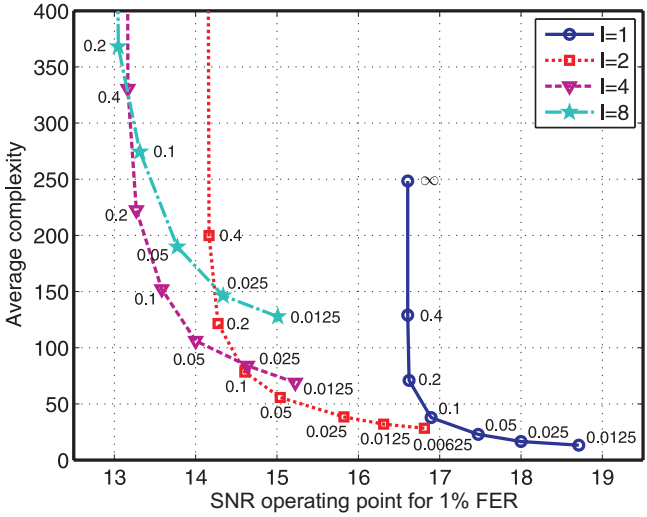


Figure 4.8: Performance/complexity tradeoff of SISO STS-SD with SQRD. The numbers next to the curves correspond to normalized LLR clipping parameters.

Column-Sorting and Regularization

We next examine the impact of column-sorting and regularization of the channel matrix on the performance/complexity tradeoff. It can be seen in Figure 4.9 that in the low-complexity regime, the Pareto-optimal tradeoff curve is achieved by MMSE-SQRD. In the high-complexity regime, the performance loss incurred by regularization renders MMSE-SQRD inferior to un-regularized SQRD. This observation has already been made for the soft-output-only case in [23], but is also valid for $I > 1$ using SISO STS-SD.

Self-Interference Free LLRs

Figure 4.9 additionally quantifies the impact of compensating self-interference—according to Section 4.3.2—on the performance and complexity of the SISO STS-SD. We observe that compensation of self-interference results in a performance improvement in terms of

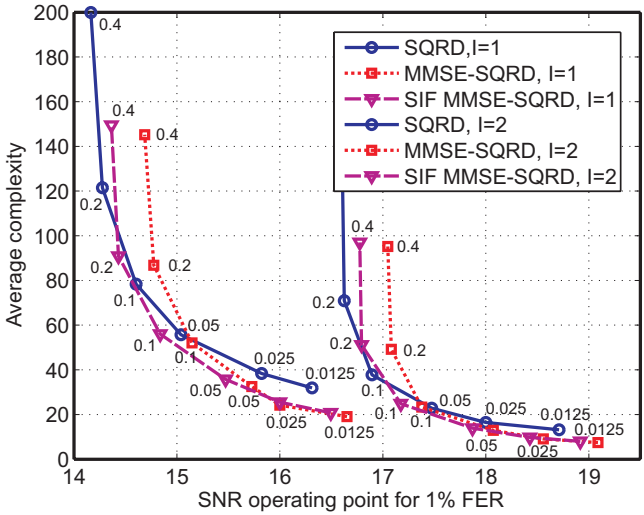


Figure 4.9: Performance/complexity tradeoff of SISO STS-SD with SQRD, MMSE-SQRD, and SIF MMSE-SQRD. The numbers next to the curves correspond to normalized LLR clipping parameters.

SNR operating point of 0.3 dB to 0.5 dB in almost all regions. In the high-complexity regime, un-regularized SQRD outperforms channel-matrix regularization and has an SNR operating point that is 0.15 dB below that obtained in the SIF case.

4.6.4 Comparison with RTS and LSD

Comparison with Repeated Tree Search

Figure 4.10 compares the performance/complexity achieved by the RTS-SD algorithm proposed in [125]. In addition, we employ LLR clipping for the RTS-SD. To this end, $r_{i,b}$ is initialized as described in Section 4.1.3 followed by an immediate update according to

$$r_{i,b} \leftarrow \min\left\{r_{i,b}, \lambda^{\mathrm{MAP}} + L_{\max}\right\} \tag{4.49}$$

which ensures that $|L^{\mathrm{E}}_{i,b}| \leq L_{\max}$ is satisfied. As a consequence of (4.49), metrics associated with counter-hypotheses for which no

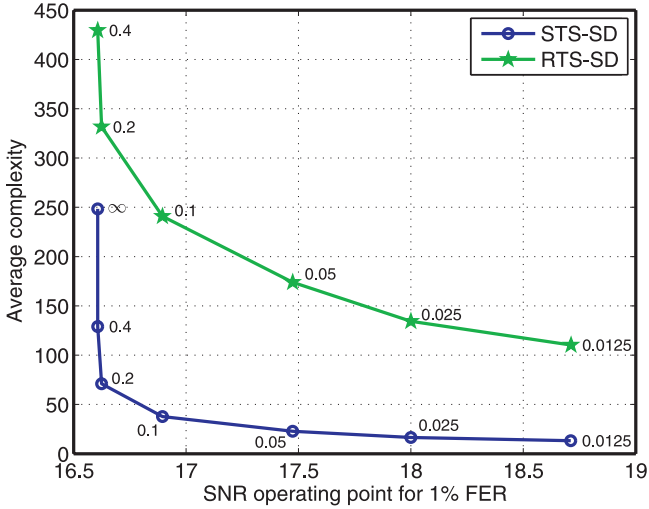


Figure 4.10: Performance/complexity tradeoff of RTS-SD and STS-SD, both using SQRD. The numbers next to the curves correspond to normalized LLR clipping parameters.

valid lattice point is found equal $\lambda^{\mathrm{MAP}} + L_{\max}$. We can see from Figure 4.10, that the RTS-SD algorithm has a significantly higher complexity—ranging between 3 to 8 times more than that of SISO STS-SD— for a given SNR operating point. We conclude that the RTS strategy has a worse performance/complexity tradeoff profile than that of the SISO STS-SD algorithm, which is caused by redundant computations of the RTS-SD during the tree-search.

Comparison with List Sphere Decoding

Figure 4.11 compares the performance/complexity tradeoff achieved by list sphere decoding (LSD) as proposed in [19] to that obtained through SISO STS-SD. For the LSD algorithm, we take the complexity to equal the number of nodes visited when building the initial candidate list. The (often significant) computational burden incurred by list administration in LSD is neglected, leading to a complexity

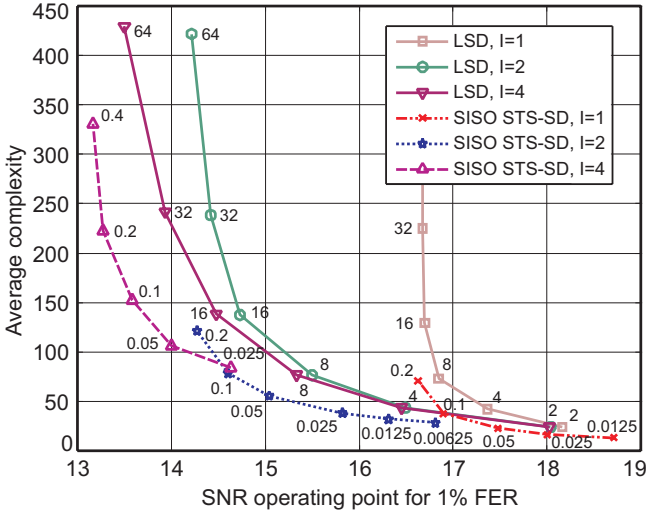


Figure 4.11: Performance/complexity tradeoff of LSD [19] and SISO STS-SD, both using SQRD. The numbers next to the curves correspond to the list size for LSD and to normalized LLR clipping parameters for SISO STS-SD.

measure that favors the LSD algorithm. We can draw the following conclusions from Figure 4.11:

i) SISO STS-SD outperforms LSD for all SNR operating points.

ii) LSD requires relatively large list sizes and hence a large amount of memory to approach (max-log) optimum SISO performance.[8] The underlying reason is that LSD obtains extrinsic LLRs from a candidate list that has been computed around the maximum-likelihood solution, i.e., in the absence of a priori information. In contrast, SISO STS-SD requires memory mainly for the extrinsic metrics, which are obtained through a search that is concentrated around the MAP solution. Consequently, SISO STS-SD

[8]In addition to the memory requirements, the search-and-replace operations required in the LSD algorithm's list administration, quickly lead to prohibitively high VLSI-implementation complexity when the list size grows (see Section 4.1.2).

tends to require (often significantly) less memory than LSD.

Besides LSD, various other SISO detection algorithms for MIMO systems have been developed, see e.g., [30–32, 42, 67, 122]. The algorithms described in [31] and [32] are related to LSD but require rebuilding the candidate list in each iteration; this can lead to a substantial complexity increase compared to LSD. For the SISO MMSE PIC described in Chapter 3 and the algorithm described in [67], issues indicating potentially high computational complexity include the requirement for matrix inversion for each symbol vector in each iteration. In contrast, the QRD required for SD has to be computed only when the channel state changes. A detailed complexity comparison (based on VLSI implementation results) between the SISO MMSE PIC and the SISO STS-SD is provided in Chapter 6. The computational complexity of the list-sequential (LISS) algorithm in [30, 122] seems difficult to relate to the complexity measure employed in this thesis. However, due to the need for sorting of candidate vectors and the structural similarity of the LISS algorithm to LSD, we expect the performance/complexity tradeoff realized by the LISS algorithm to be comparable to that of the LSD algorithm.

4.6.5 Impact of LLR Correction

Figure 4.12 shows examples for LLR correction functions of SISO STS-SD obtained by linear interpolation using $K = 31$ bins and side information given by

$$\mathcal{Z} = \left\{ L_{\max}, D_{\text{avg}}, \mathsf{SNR}, T \right\} \tag{4.50}$$

where $L_{\max} = 0.2$, $D_{\text{avg}} \in \{16, \infty\}$, $\mathsf{SNR} = 16\,\text{dB}$, and $T \in \{0, 1\}$ indicates whether early termination occurred ($T = 1$) or not ($T = 0$). Here, the number of instances of $\mathcal{Z}$ is given by $Z = 4$. Note that in practice, the parameters $L_{\max}$, D_{avg}, and SNR in $\mathcal{Z}$ remain constant as long as the channel state remains constant, whereas T may change at symbol-vector rate, i.e., depending on T, different LLR correction functions need to be applied to the extrinsic LLRs $L^{\text{E}}_{i,b}$. We compare the LLR correction functions corresponding to SISO STS-SD using column-sorting (SQRD), regularization and column-sorting (MMSE-SQRD), and compensation of self-interference in combination with

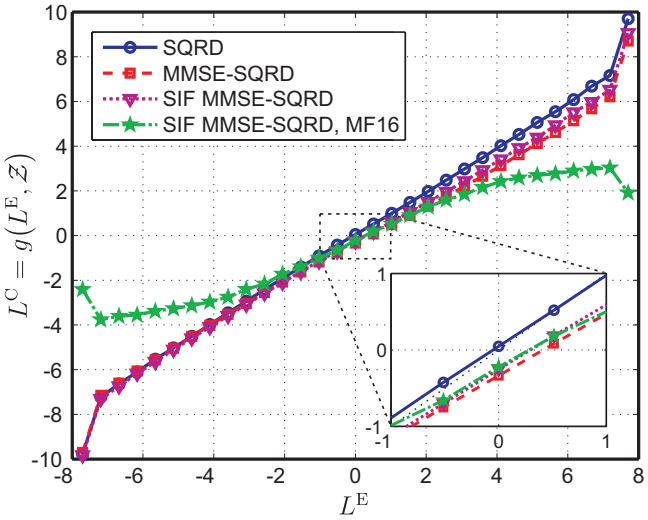


Figure 4.12: Different LLR correction functions for $I = 1$ at $\mathsf{SNR} = 16\,\mathrm{dB}$ using $\mathcal{Z} = \{L_{\max}, D_{\mathrm{avg}}, \mathsf{SNR}, T\}$.

MMSE-SQRD, all having unconstrained maximum complexity (i.e., $D_{\mathrm{avg}} = \infty$ and, hence, $T = 0$). We furthermore show the correction function of SIF (MMSE-SQRD) LLRs in combination with MF scheduling for $D_{\mathrm{avg}} = 16$ (denoted by "MF16" in Figure 4.12) and $T = 1$. The following observations can be made:

i) For unconstrained complexity, i.e., $D_{\mathrm{avg}} = \infty$, LLRs corresponding to $\pm L_{\max}$ are corrected to LLRs with larger magnitude; this is a result of clipping LLRs with magnitude larger than $L_{\max}$ to $\pm L_{\max}$. We note that since the LLR correction functions are obtained by binning and linear interpolation, LLR-values that have slightly smaller (mandated by the bin-width) magnitude than $L_{\max}$ are also corrected to values larger than $L_{\max}$.

ii) For early termination with MF-scheduling (i.e., $D_{\mathrm{avg}} = 16$ and $T = 1$), LLRs with magnitude close to $L_{\max}$ are corrected to LLRs with smaller magnitude (i.e., their reliability is reduced). LLRs corresponding to $L^{\mathrm{E}}_{i,b} = \pm L_{\max}$ are, as already mentioned

in Section 4.5.3, often caused by early termination and hence, are corrected to less reliable LLR-values.

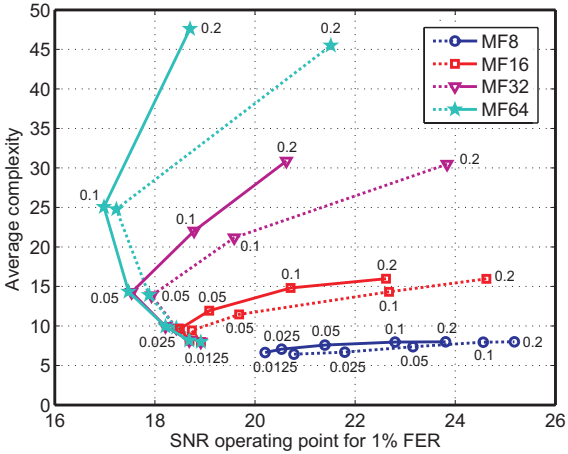
iii) The LLR correction function associated with column-sorting (SQRD) only is almost a linear function with slope one, i.e., $L_{i,b}^{\mathrm{C}} = L_{i,b}^{\mathrm{E}}$, which indicates that little correction is performed. The reason for this behavior is that column-sorting maintains (max-log) optimality and the impact of the max-log approximation on performance is small, in general (see Section 4.6.1). The correction functions associated with channel-matrix regularization show a stronger deviation from $L_{i,b}^{\mathrm{C}} = L_{i,b}^{\mathrm{E}}$ (cf. the zoom in Figure 4.12), indicating that more correction is required, since regularization leads to an approximation of the max-log LLRs (see Section 4.3.1).

Performance/Complexity Tradeoff with Early Termination

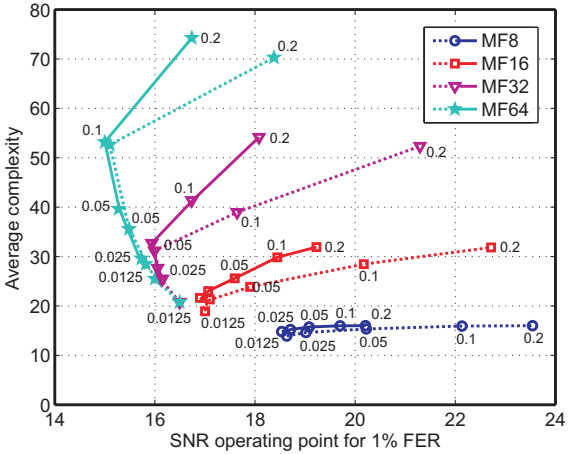
Figure 4.13 shows the performance/complexity tradeoff for early-termination based on MF scheduling with and without LLR correction. The side information was chosen according to (4.50) and the LLR correction function was computed based on $K = 31$ bins with linear interpolation.

Depending on the average run-time constraint, LLR correction can reduce (i.e., improve) the SNR operating point by up to 3 dB. As expected, the performance gains resulting from LLR correction are more pronounced for larger clipping parameters as in these cases performance is dominated by the run-time constraint and early termination happens more often. Note that LLR correction also yields slight performance gains for small LLR clipping levels, where the run-time constraints do not affect performance. This indicates that LLR correction can also correct—at least partly—the errors induced by LLR clipping and by channel-matrix regularization.

In summary, we can conclude that LLR correction is able to significantly improve the SNR operating point. Moreover, we note that for a given run-time constraint, there exists an optimum LLR clipping level, in the sense of minimizing the SNR operating point. It is therefore important to choose the LLR clipping level in accordance with the average run-time constraint.



(a) $I = 1$



(b) $I = 2$

Figure 4.13: Impact of LLR correction. The solid lines correspond to the performance obtained with LLR correction, whereas the dotted lines pertain to un-corrected LLRs. Both variants employ early termination with MF scheduling and compensation of self-interference in the LLRs in combination with MMSE-SQRD.

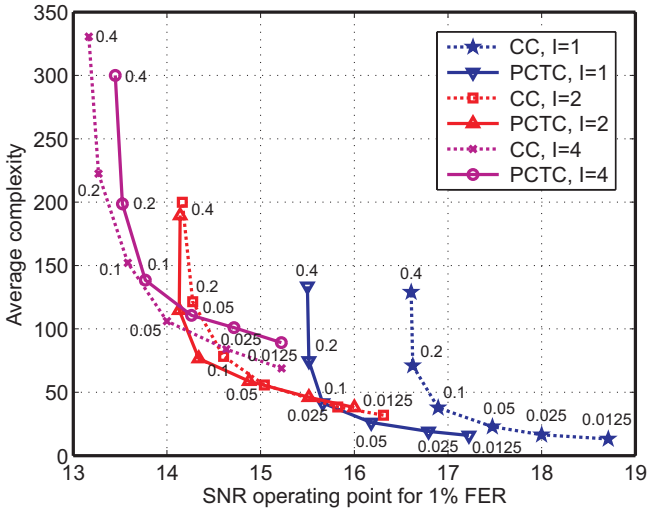


Figure 4.14: Performance/complexity tradeoff of SISO STS-SD with SQRD (without regularization). Comparison between parallel-concatenated turbo codes (PCTCs) and convolutional codes (CCs).

Performance/Complexity Tradeoff for Turbo Codes

The next simulation result is aimed at understanding which of the conclusions drawn so far change in the presence of more sophisticated channel codes. To this end, we evaluated the performance/complexity tradeoff for a parallel-concatenated turbo code (PCTC) of rate 1/2 (punctured, memory 2, and generator polynomial $[7_o\ 5_o]$, where 7_o pertains to the feedback path) with eight iterations in the turbo decoder. We use the interleaver specified in the 3GPP standard [139] with 508 information bits. One code-block corresponds to 1024 coded bits including two times four bits for termination of the trellises. For aggressive LLR clipping, simulation results have shown that using the sum-product algorithm within the turbo decoder requires precise (and hence, corrected) LLRs to yield satisfactory results, whereas max-log-based decoders seem to be more robust to effects incurred by LLR clipping. Since we employ the sum-product BCJR algorithm [34] for

decoding of the PCTCs, LLR correction is used.

The results in Figure 4.14 indicate that the performance and complexity achieved by the PCTC in the first iteration is significantly better than that obtained for the convolutional code (CC) used in the previous simulations. In the second iteration, the performance/complexity tradeoff is almost identical for both codes. For $I > 2$, the CC slightly outperforms the PCTC, which could be due to the fact that we use a turbo code with very short block length and a channel model that exhibits correlation across frequency and space (see, e.g., [140]).

4.6.6 Information Transfer Characteristics

In order to characterize the performance of soft-input soft-output MIMO detectors *independently* of the channel code and channel decoder, we compute information transfer characteristics (ICTs) using an i.i.d. (across space and OFDM tones) Rayleigh multi-path fading channel model and assuming a Gaussian model for the a priori LLRs according to [141]

$$L_{i,b}^{\mathrm{A}} = \frac{2}{\sigma^2}\big(x_{i,b} + n\big)$$

where n is a real-valued Gaussian RV with zero mean and variance σ^2. The a priori information content is determined by σ^2 and characterized by the mutual information between the transmitted bits $x_{i,b}$ and the a priori input of the SISO detector, i.e., $I_{\mathrm{A}} = I\big(x_{i,b}; L_{i,b}^{\mathrm{A}}\big)$ (in bits per binary symbol) where $0 \leq I_{\mathrm{A}} \leq 1$. Note that large and small values of σ^2, reduce and increase the mutual information I_{A}, respectively. The extrinsic information at the output of the detector (averaged over all transmit antennas and bits) is defined as

$$I_{\mathrm{E}} = \frac{1}{M_{\mathrm{T}}Q}\sum_{i=1}^{M_{\mathrm{T}}}\sum_{b=1}^{Q} I\Big(x_{i,b}; L_{i,b}^{\mathrm{E}}\Big)$$

in bits per binary symbol where $0 \leq I_{\mathrm{E}} \leq 1$. Note that $L_{i,b}^{\mathrm{A}} = 0$ implies $I_{\mathrm{A}} = 0$ and corresponds to soft-output-only MIMO detection. The information transfer characteristic (ITC) corresponds to

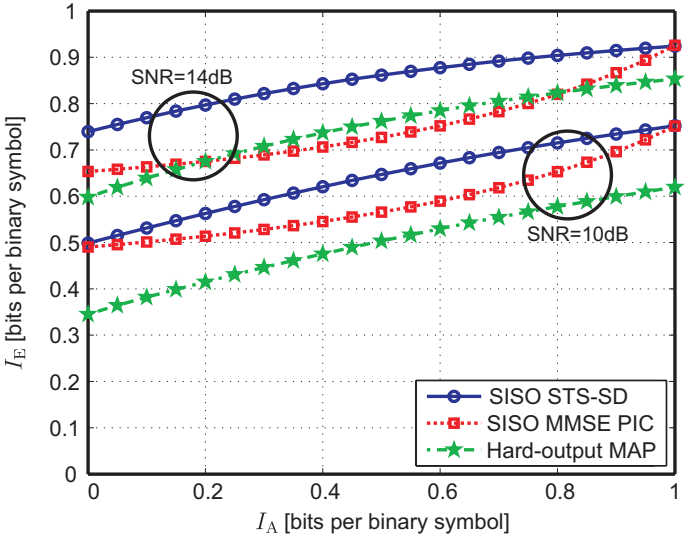


Figure 4.15: ITCs of SISO STS-SD, SISO MMSE PIC, and hard-output MAP detection for 10 dB and 14 dB SNR.

the function $I_E = h(I_A)$, for a given SNR, and enables us to assess the performance of soft-input soft-output MIMO detectors in a fundamental way. Note that the application of ITCs as described here was originally proposed in [4, Chapter 16].

SISO MMSE PIC vs. SISO STS-SD

Figure 4.15 compares the ITC of the low-complexity SISO MMSE PIC described in Section 3.2 with the SISO STS-SD (using $L_{\max} = \infty$) and the hard-output MAP detector (2.4) for 10 dB and 14 dB SNR. We can see that for $I_A = 0$, the SISO STS-SD outperforms the SISO MMSE PIC algorithm and hard-output MAP detection. Remarkably, the SISO MMSE PIC algorithm attains the same information transfer as the SISO STS-SD, if perfect a priori information is available (i.e., for $I_A = 1$). Note that for $I_A \approx 0.5$, the SISO MMSE PIC yields poor performance and is even outperformed by the hard-output MAP decoder at 14 dB SNR. We can therefore conclude that the SISO STS-

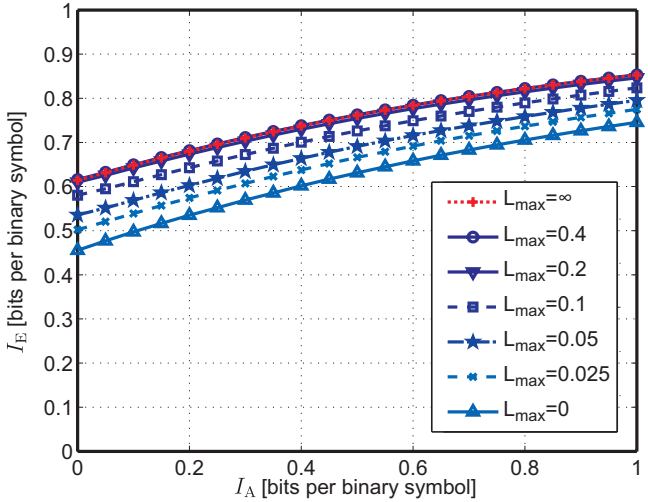


Figure 4.16: ITC of SISO STS-SD at $\mathsf{SNR} = 12\,\mathrm{dB}$ for different (normalized) LLR clipping parameters.

SD has the overall better ITC compared to that of SISO MMSE PIC.

Impact of LLR clipping

Figure 4.16 shows that a normalized LLR clipping parameter of $L_{\max} = 0.4$ achieves almost the same ITC as max-log optimal SISO STS-SD with $L_{\max} = \infty$. Hence, increasing the LLR clipping parameter to a value above 0.4 does not further improve performance of the detector and only leads to an increase in complexity. We note that the same observation was made in the performance/complexity tradeoff simulations in Figure 4.8.

Performance comparison with LSD

Figure 4.17 compares the ITC of SISO STS-SD to that of LSD [19]. For I_A close to 1, LSD requires large list-sizes to yield a performance close to that of the max-log-optimal SISO STS-SD algorithm. Note that even hard-output MAP detection (which corresponds to SISO

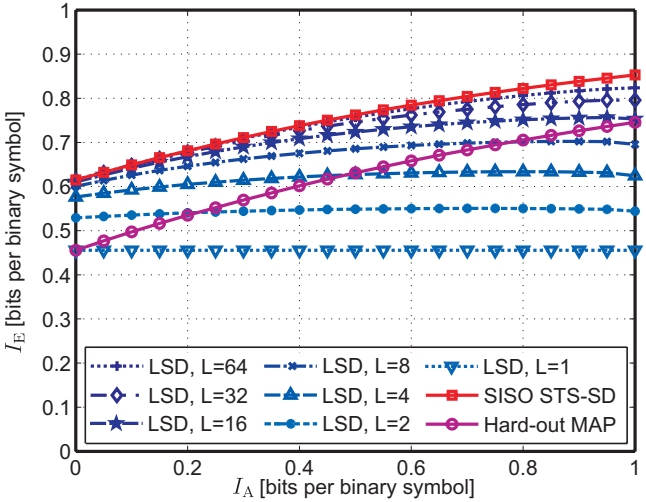


Figure 4.17: ITC of LSD compared to hard-output MAP and (max-log) optimal SISO STS-SD performance at $\mathsf{SNR} = 12\,\mathrm{dB}$.

STS-SD with $L_{\max} = 0$) can outperform LSD—in terms of ITCs—if I_A is close to 1 and the list-size is small. We therefore conclude that SISO STS-SD has a fundamental performance advantage over LSD, which is in agreement with the observations made in Section 4.6.4

4.6.7 Approaching Outage-Capacity with SISO STS Sphere Decoding

We finally compare the performance obtained with SISO STS-SD to the outage capacity given in (2.33). The performance comparison consists of setting the outage probability and the FER to 1% and identifying the corresponding SNR operating points. Figure 4.18 shows the corresponding results for SISO STS-SD with different modulation schemes for $I = 1$ and $I = 8$. Note that the LLR clipping parameters are chosen so as to minimize complexity while retaining near-max-log optimal performance at 1% FER (i.e., we used $L_{\max} = 0.1$, $L_{\max} = 0.4$, $L_{\max} = 2.0$, and $L_{\max} = 6.0$ for 64-QAM, 16-QAM,

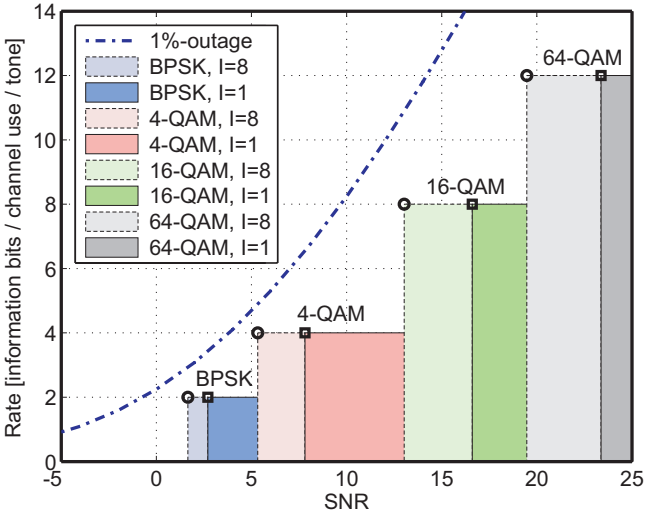


Figure 4.18: 1%-outage-capacity compared to the SNR operating points of SISO STS-SD for 1% FER. Squares and circles correspond to SNR operating points realized by $I = 1$ and $I = 8$, respectively.

QPSK, and BPSK, respectively). We can see that SISO STS-SD operates between 1.5 dB (for 4-QAM) and 5.3 dB SNR (for 64-QAM) away from outage capacity.

4.7 VLSI Implementation of Soft-Output Single Tree-Search Sphere Decoding

The first VLSI implementation for hard-output SESD has been described in 2004 by Burg *et al.* [103], which was the proof-of-concept that ML detection for MIMO systems is feasible in practical systems. Further complexity reduction on algorithmic and hardware level led to the highly-efficient one-node-per-cycle (ONPC) SESD architecture described in [33]. Up till now, a variety of SD architectures for hard-output SD have been proposed in the literature, e.g., [138, 142, 143]. The VLSI implementation described by Cerato *et al.* in 2008 [105]

demonstrates that SD suitable for efficient decoding of STBCs in practice as well. The first soft-output detector for MIMO systems based on list sphere-decoding (LSD) developed in [19] was described by Wenk *et al.* in 2006 [127]. The number of recent SD implementations available in the literature indicates that VLSI implementation for SD is still an active research area.

In this section, we describe a reference VLSI implementation of the soft-output (SO) STS-SD algorithm developed above. We start by briefly reviewing the hard-output SESD architecture developed in [33], which is the basis of our proposed VLSI implementation. Then, algorithmic modifications of the soft-output STS-SD algorithm are presented, in order to enable economic implementation in practical systems. Finally, we describe a VLSI architecture for soft-output STS-SD and show corresponding implementation results.

4.7.1 VLSI Architecture

Since the proposed soft-output STS-SD VLSI implementation is based on the one-node-per-cycle (ONPC) VLSI architecture developed in [33] for hard-output SD, we start our discussion by briefly reviewing relevant aspects of [33].

A Brief Review of the ONPC Architecture in [33]

The VLSI architecture proposed in [33] employs two functional units:

Metric Computation Unit (MCU) The MCU handles the forward iteration in the search tree by identifying the starting-point for the SE enumeration (i.e., the current node's child that has the smallest PED) using the direct-QAM enumeration algorithm initially proposed in [19] and slightly modified in [33]. The basic idea behind this enumeration method for QAM constellations is as follows: The QAM constellation is first decomposed into subsets of constellation points that have the same modulus, referred to as phase-shift keying (PSK) subsets. Within each of these PSK subsets, the child associated with the smallest PED can be identified based on the phase of $b_i = \tilde{y}_i - \sum_{j=i+1}^{M_T} R_{i,j} s_j$ only. The corresponding minimum PEDs

(one for each subset) are then computed and compared. The minimum PED across subsets identifies the starting point for the SE enumeration. If the resulting child neither corresponds to a leaf nor qualifies for pruning, the decoder proceeds in forward direction by declaring this child as the next parent node to be examined by the MCU (cf. ① in Figure 4.19).

Metric Enumeration Unit (MEU) The MEU maintains a list of preferred children, one for each node between the root and the parent of the node whose children are currently under examination by the MCU. To this end, the MEU follows the MCU on its path through the tree with one cycle delay. While the MCU visits a node, the MEU considers this node's siblings and identifies the one that should be visited next according to the SE criterion. This sibling is found by applying the direct-QAM enumeration principle described above, where within each PSK subset the next (according to the SE criterion) candidate follows immediately by zig-zag enumeration along the circle. The decision on the preferred child across subsets must again be made by explicit computation and comparison of the smallest PEDs of the individual PSK subsets.

When the forward iteration stalls, either because the child identified by the MCU corresponds to a leaf or must be pruned, the MEU provides a new parent node to the MCU in the next clock cycle (cf. ② in Figure 4.19). This parent node is chosen by the MEU, following the depth-first paradigm, from those members of the list of preferred children which do not qualify for pruning.

VLSI Architecture for Soft-Output STS-SD

The block diagram of the proposed soft-output STS-SD VLSI implementation is shown in Figure 4.19. Compared to the architecture for hard-output SD described in [33], changes are made in the MCU and two additional units are required, one for list administration as described in Section 4.2.1 and one for the implementation of the pruning criterion as described in Section 4.2.3. We shall next describe the specifics of these changes. Note that we assume soft-output MIMO detection in the following, i.e., we consider the case where $L_{i,b}^{\mathrm{A}} = 0$ $(\forall i, b)$.

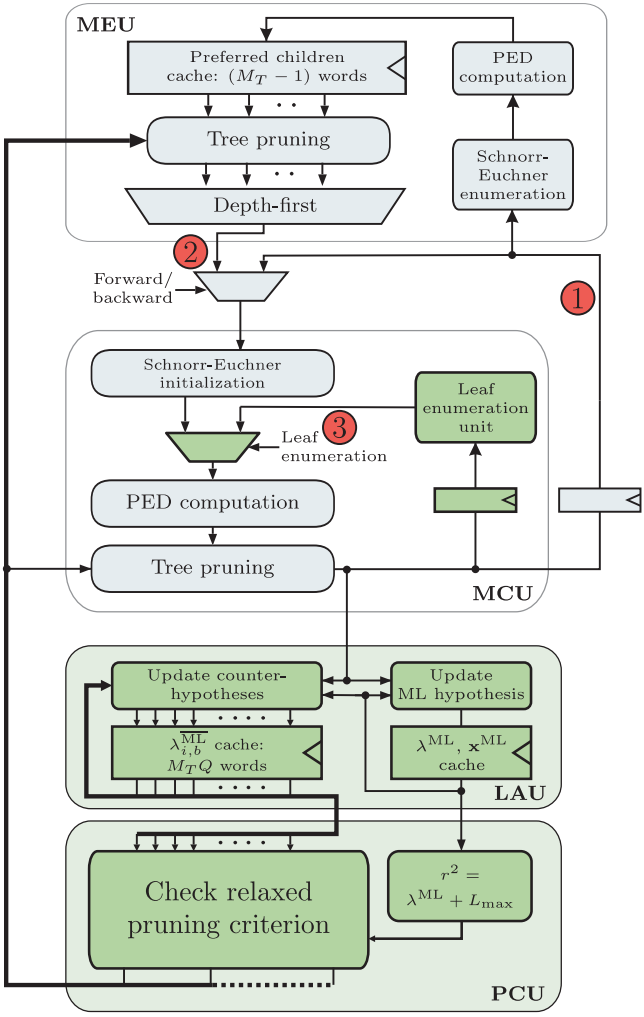


Figure 4.19: Block diagram of the VLSI architecture for soft-output STS-SD. Additional units, compared to the hard-output SD described in [33], are highlighted.

Architectural Changes in the MCU From a high-level architectural perspective, there is one fundamental difference between tree-traversal for hard-output SD and for soft-output STS-SD algorithm: When the node currently examined by the MCU is on the level just above the leaves (i.e., on level $i = 2$), hard-output SD only considers one child, namely the one associated with the smallest PED. In contrast, the metrics which may be affected during SO STS-SD procedure in the subtree emanating from the node $\mathbf{s}^{(i)}$ are given by the set

$$\begin{aligned}\mathcal{A}\Big(\mathbf{x}^{(i)}\Big) = \{a_l\} = &\Big\{\lambda_{j,b}^{\overline{\mathrm{ML}}} \,\Big|\, \big(j \geq i, \forall b\big) \wedge \big(x_{j,b} = \overline{x_{j,b}^{\mathrm{ML}}}\big)\Big\} \\ &\cup \Big\{\lambda_{j,b}^{\overline{\mathrm{ML}}} \,\Big|\, j < i, \forall b\Big\} \end{aligned} \tag{4.51}$$

where the superscript $^{\mathrm{ML}}$ refers to the ML solution. The node $\mathbf{s}^{(i)}$ along with its subtree is pruned if its PED $d\Big(\mathbf{s}^{(i)}\Big)$ satisfies

$$d\Big(\mathbf{s}^{(i)}\Big) > \max_{a_l \in \mathcal{A}\left(\mathbf{x}^{(i)}\right)} a_l \,. \tag{4.52}$$

Note that (4.51) and the tree-pruning criterion (4.52) correspond to that of the SISO STS-SD (see Section 4.2.3) if no a priori information is available, i.e., for $L_{i,b}^{\mathrm{A}} = 0$ $(\forall i, b)$. The soft-output STS-SD algorithm has to compute the PEDs of *all* children that do not qualify for pruning according to the criterion since these children may lead to updates of the metrics $\lambda_{i,b}^{\overline{\mathrm{ML}}}$. To perform this *leaf enumeration* procedure, the STS decoder must revisit the current node at level $i = 2$, which requires additional clock cycles and a leaf enumeration unit shown in Figure 4.19. This unit does, however, not require an additional arithmetic unit for the PED computation as it can reuse the PED computation unit in the MCU (cf. ③ in Figure 4.19).

List Administration and Tree Pruning In addition to the modifications in the MCU described above, the SO STS-SD algorithm requires the following two additional units:

List-administration unit (LAU): The LAU is responsible for maintaining and updating the list containing $\mathbf{x}^{\mathrm{ML}}$, λ^{ML}, and the $\lambda_{i,b}^{\overline{\mathrm{ML}}}$. The corresponding unit is active during the leaf-enumeration process

described above. Since the update rules implemented by the LAU require only a small number of logic operations, the silicon area of this unit is small (see Table 4.4) and is dominated by the storage space required for the metrics λ^{ML} and $\lambda_{i,b}^{\overline{\mathrm{ML}}}$.

Pruning criterion unit (PCU): The PCU is responsible for computing the reference metrics, i.e., the RHS of (4.52), required to implement the corresponding pruning criterion. From a VLSI implementation perspective, the reference metric on level i depending on the partial label $\mathbf{x}^{(i)}$ constitutes a major problem. More specifically, this dependence causes the criterion for pruning the child of a parent node on level $i+1$ to depend on the partial label $\mathbf{x}^{(i)}$ of that child. This, in turn, implies that enumeration of the children on level i in ascending order of their PEDs according to the SE criterion can, in general, not be applied, which results in the need for exhaustive-search enumeration and is thus ill-suited for VLSI implementation [33]. An adjustment of the pruning criterion in (4.51) and (4.52) solves this problem. To this end, we define

$$\begin{aligned}\mathcal{B}\Big(\mathbf{x}^{(i+1)}\Big) = \{b_l\} = &\Big\{\lambda_{j,b}^{\overline{\mathrm{ML}}} \,\Big|\, \big(j > i, \forall b\big) \wedge \big(x_{j,b} = \overline{x_{j,b}^{\mathrm{ML}}}\big)\Big\} \\ &\cup \Big\{\lambda_{j,b}^{\overline{\mathrm{ML}}} \,\Big|\, j \leq i, \forall b\Big\}\end{aligned}$$

and prune the node $\mathbf{s}^{(i)}$ along with its subtree if $d\Big(\mathbf{s}^{(i)}\Big)$ satisfies

$$d\Big(\mathbf{s}^{(i)}\Big) > \max_{b_l \in \mathcal{B}\left(\mathbf{x}^{(i+1)}\right)} b_l \,. \tag{4.53}$$

Note that the RHS of the modified pruning criterion (4.53) depends on the partial label $\mathbf{x}^{(i+1)}$ rather than on $\mathbf{x}^{(i)}$. Consequently, the enumeration of the children of a node on level $i+1$ can be carried out using the SE criterion.

Complexity Impact of List Administration and Tree Pruning

We argued above that, for a ONPC architecture, the number of visited nodes is equal to the number of clock cycles required for decoding, thus reflecting the true silicon complexity of the algorithm. However, for the proposed soft-output STS-SD architecture the number of clock

Table 4.3: Implementation results of hard-output SESD [33], soft-output LSD [127], and soft-output STS-SD ASICs in 250 nm CMOS (1P/5M) technology.

	Hard-output SESD [33]	Soft-output LSD[a] [127]	Soft-output STS-SD
Cell area [kGE[b]]	34.4	127	56.8
Core area [mm^2]	1.2	n.a.	1.9
Clock freq. [MHz]	73	170	71
Efficiency [kGE/MHz]	0.47	0.74	0.80

[a]The LSD implementation with list-size 4 and $\alpha = 2$ from [127] has been used.
[b]One GE corresponds to the area of a two-input drive-one NAND gate of size 23.76 µm^2.

cycles will be larger than the number of visited nodes shown in the numerical results in Section 4.7.2, for two reasons: First, modifying the pruning criterion (4.18) to result in (4.53) leads to less efficient pruning as

$$\max_{b_l \in \mathcal{B}\left(\mathbf{x}^{(i+1)}\right)} b_l \geq \max_{a_l \in \mathcal{A}\left(\mathbf{x}^{(i)}\right)} a_l.$$

The corresponding increase in complexity is, however, significantly smaller than what would be incurred if exhaustive search enumeration on (4.18) would be applied. The second reason for the number of clock cycles being higher than the number of visited nodes is that every time the leaf-enumeration process is performed, one additional cycle is consumed to detect the end of the enumeration process. Consequently, the proposed VLSI architecture no longer strictly follows the ONPC paradigm. The performance/complexity trade-off curves in Figure 4.20 show, however, that the impact of both effects (discussed above) leads to the number of clock cycles being only slightly higher than the number of visited nodes.

4.7.2 Implementation Results

In order to assess the true silicon complexity (in terms of circuit area and maximum achievable clock frequency) of the proposed soft-

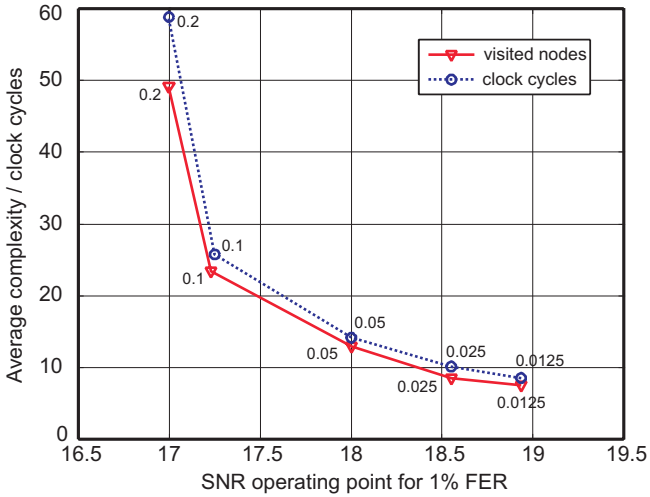


Figure 4.20: Average complexity of the soft-output STS-SD algorithm compared with the average number of clock cycles of the corresponding VLSI implementation (with MMSE-SQRD preprocessing). The numbers next to the curves correspond to normalized LLR clipping levels.

output STS sphere decoder, we implemented the VLSI architecture described in the previous section in 250 nm (1P/5M) CMOS technology for a MIMO system with $M_{\mathrm{T}} = M_{\mathrm{R}} = 4$ using 16-QAM modulation. The resulting ASIC layout is shown in Figure 4.21. The design parameters of the decoder are summarized in Table 4.3 which, for reference, also contains the design parameters of an ℓ^2-norm hard-output SESD —following the design principles employed for the ℓ^∞-norm hard-output SESD described in [33]— and implementation results of the *pipelined* soft-output LSD architecture described in [127]. We emphasize that the implementation of the LSD maintains a list with four candidate vectors and employs pipeline-interleaving, which improves the overall hardware-efficiency (in terms of kGE per clock frequency), e.g., [15, 138, 143]. Note that pipeline-interleaving can — due to structural similarities between all three architectures— also be applied for hard-output SESD (as demonstrated in [138]) and SO STS-

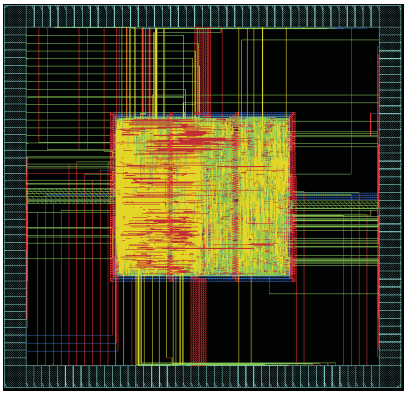


Figure 4.21: Layout of the soft-output STS-SD ASIC in 250 nm CMOS (1P/5M) technology.

SD, which would lead to dramatic improvements in terms of hardware-efficiency for such architectures as well. Hence, the advantage of the LSD implementation over the SO STS-SD implementation in terms of hardware-efficiency is due to architectural optimization and *not* due to an advantage on algorithmic level.

Detailed Area Breakdown We can see from Table 4.3 that the chip area required by soft-output STS-SD is only 58% higher than that required by a corresponding ℓ^2-norm hard-output sphere decoder. The detailed area breakdown in Table 4.4 shows that most of the area increase results from the LAU, the PCU, and the arithmetic unit that computes the LLRs. Further area increase is due to the need to store the LLRs in the output buffer of the ASIC. The additional SE enumeration unit in the MCU required for leaf enumeration adds only 1.9 kGE to the overall area. The SO STS-SD ASIC shows only slightly lower maximum clock frequency than the corresponding hard-output sphere decoder. The reason underlying this only negligible reduction in maximum clock frequency is that most of the additional logic required by the STS-SD ASIC can be kept off the critical path and has thus little influence on the maximum clock frequency.

Table 4.4: Area breakdown of the different units of the hard-output SD [33] and the SO STS-SD implementation.

	Hard-output SD		Soft-output STS-SD	
	Area [kGE]	Area [%]	Area [kGE]	Area [%]
Mem. $(\mathbf{y}, \mathbf{R})$	4.6	13.4	4.5	7.9
MCU	16.6	48.3	18.5	32.6
MEU	11.9	34.6	10.9	19.2
Output buffer	0.8	2.3	5.0	8.8
Control logic	0.5	1.6	0.5	0.9
LAU			8.4	14.7
PCU	-	-	6.4	11.3
LLR Comp.			2.6	4.6
Total	34.4	100	56.8	100

Throughput Figure 4.22 shows the performance/throughput trade-off of the reference ℓ^2-norm hard-output sphere decoder and the soft-output STS-SD ASIC described in Section 4.7.1, where the *average* throughput corresponds to

$$\Theta = \frac{RQM_T}{\mathbb{E}[C]} f_{\text{clk}} \quad [\text{bit/s}] \tag{4.54}$$

and is measured in *information*-bits per second as a function of the minimum required SNR to achieve a 1% FER. Here, f_{clk} is the maximum clock frequency of the circuit under consideration and $\mathbb{E}[C]$ denotes the average (over channel and noise realizations) number of clock cycles required to detect a symbol vector. Note that the dedicated hard-output SD implementation achieves a slightly higher throughput than that of the soft-output STS-SD implementation with[9] $L_{\max} = 0$, which is a result of the slightly higher maximum clock frequency of the corresponding hard-output SD implementation (see Table 4.3). We conclude that the circuit complexity of SO STS-SD is only 58% higher than that of the reference hard-output SESD VLSI implementation [33] and enables to deliver exact max-log soft-outputs.

[9]Recall that for $L_{\max} = 0$, the (error rate) performance of the STS-SD algorithm corresponds to that of a hard-output sphere decoder.

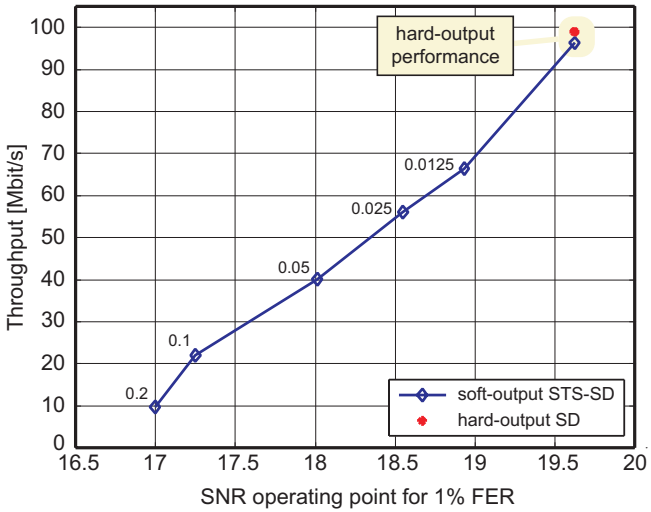


Figure 4.22: Throughput characteristics of the soft-output STS-SD and the reference hard-output SE VLSI implementation both using MMSE-SQRD preprocessing in 250 nm CMOS technology.

Chapter 5

Soft-Input Soft-Output Channel Decoding

Channel codes that provide excellent error-correction performance are of paramount importance in wireless communication systems. In iterative MIMO systems, reliability information is exchanged between the soft-input soft-output MIMO detector and the channel decoder. Hence, channel decoders must not only produce estimates of the transmitted bits, but also be able to compute soft-outputs. The design of channel codes and algorithms for SISO channel decoding is a well-studied topic in the literature, e.g., [144]. However, not much is known about the performance and VLSI-implementation complexity of SISO channel decoders for iterative MIMO decoding.

In this chapter, we compare the performance and VLSI implementation complexity of SISO decoders for three different channel codes, in view of iterative MIMO decoding. In Section 5.1, we develop architectures and VLSI implementations for high-throughput SISO decoding of convolutional codes (CCs). Architectures and VLSI implementations of SISO decoding for low-density parity check (LDPC) codes and parallel-concatenated turbo codes (PCTCs) is studied in Section 5.2 and Section 5.3, respectively. A detailed performance and complexity analysis is provided in Section 5.4.

5.1 Convolutional Codes

In 1955, CCs have been proposed by Elias [145] for forward error-correction in discrete memory-less channels. Thanks to the Viterbi algorithm [146], high-throughput (hard-output) decoding of CCs is feasible in practice at low hardware complexity. Therefore, CCs are nowadays considered in many modern (high-throughput) wireless communication standards, such as IEEE 802.11n [2] IEEE 802.16e [10], or 3GPP LTE [11].

In the remainder of this section, we compare the performance and hardware complexity associated with SISO decoding of CCs. To this end, CCs are briefly reviewed and a corresponding high-throughput SISO decoding architecture, based on the algorithm developed by Bahl *et al.* in 1974 [34], is provided. Finally, we show measurement results for 4-, 8-, 16-, 32-, and 64-state SISO channel decoder implementations.

Encoding of Convolutional Codes

In the sequel, we only consider CCs of rate $R = 1/N$ and without feedback path.[1] CCs are generated by feeding binary-valued information bits $x_k \in \{+1, -1\}$ (for $k = 1, \ldots, L$, where L denotes the number of information bits) into a shift-register of length ν. The coded bits are denoted by $c_{k,b} \in \{+1, -1\}$ ($\forall k$ and $b = 1, \ldots, N$) and generated by summations in GF(2) of well-defined values contained in the shift-registers. CCs are uniquely described by the constraint length $K = \nu + 1$ and the code generator $\mathbf{p} = [\, g_1 \ \ldots \ g_N \,]$, e.g., [147, 148]. The constraint length K denotes the maximum number of information bits that contribute to the outputs of the encoder. Each entry in $\mathbf{p}$ determines the connections to the N GF(2) adders. Figure 5.1 shows a rate $R = 1/2$ CC with $K = 3$ and generator $\mathbf{p} = [\, 5_o \ 7_o \,] = [\, 101_b \ 111_b \,]$; the subscripts $_o$ and $_b$ denote the octal and binary number format, respectively. Whenever the ith bit in p_n is equal to 1, a connection from the ith shift-register signal to the nth GF(2) adder (associated with the nth output) is made, whereas for 0 no connection is made.

[1] Other rates (such as, e.g., 2/3 or 5/6) can easily be obtained from $R = 1/2$ codes by the use of puncturing [147] as it is used in, e.g., IEEE 802.11n [2].

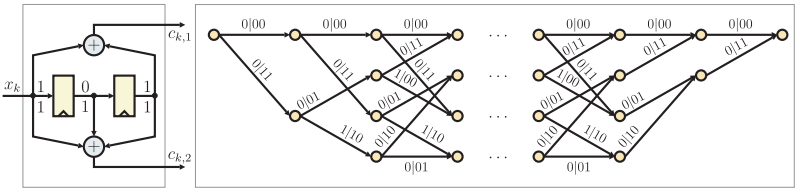


Figure 5.1: Left: convolutional encoder of the $R = 1/2$, $K = 3$, $\mathbf{p} = [\,101_b\ 111_b\,]$ CC. Right: corresponding trellis representation.

Trellis Representation

Figure 5.1 shows the trellis representation of a CC with $K = 3$. Trellises consist of states and branches. States correspond to the state of the shift-registers, denoted by $s \in \mathcal{S}_k$, where the set $\mathcal{S}_k$ contains all states at trellis step k. The maximum number of states is given by $S = 2^{\nu} = 2^{K-1}$. Branches correspond to state tuples (s', s), where $s' \in \mathcal{S}_{k-1}$ and $s \in \mathcal{S}_k$. Each branch is associated to the tuple $(x_k|c_{k,1} \cdots c_{k,N})$, i.e., with the information bit x_k and N generated code-bits at step $k = L$. In order to reach a well-defined state at the end of each code-block (i.e., at step $k = L$), the last ν information bits $k = L - \nu + 1, \ldots, L$ are used to force the encoder into a well-defined state. In Figure 5.1, for example, the code has been forced in the all-zero state at $k = L$.

Maximum Free-Distance Codes

In order to optimize the error-rate performance of CCs, it is important to choose a code that maximizes the minimum free (Hamming) distance between the all-zero codeword and any other codeword [147]. The minimum free distance of CCs is determined by the code generator $\mathbf{p}$, the code rate R, and the constraint length K. Table 5.1 shows maximum free distance code generators for $R = 1/2$ and $K \in \{3, 4, 5, 6, 7\}$. Increasing the constraint length or decreasing the code rate can improve the (error-rate) performance of CCs. However, as it will be shown in the remainder of this section, the hardware-complexity grows exponentially in K and hence, decoding of codes with large constraint length quickly becomes prohibitive.

Table 5.1: Maximum free-distance $R = 1/2$ codes [147].

K	S	$d_{\min}$	Generator $\mathbf{p}$
3	4	5	$[5_o\ 7_o]$
4	8	6	$[15_o\ 17_o]$
5	16	7	$[23_o\ 35_o]$
6	32	8	$[53_o\ 75_o]$
7	64	10	$[133_o\ 171_o]$

5.1.1 The BCJR Algorithm

CCs are usually decoded by the Viterbi algorithm (VA) [146], which efficiently[2] computes the most-likely input sequence (i.e., performs hard-output detection). For iterative MIMO systems, however, soft-outputs are required. Prominent SISO decoding algorithms for CCs providing different performance/complexity tradeoffs are the soft-output Viterbi algorithm (SOVA) [150], the LISS algorithm [151], soft-output stack algorithms, e.g., [152], or the decoding algorithm of Bahl, Cocke, Jelinek, and Raviv (BCJR) [34]. It is important to note that the SOVA, the LISS algorithm, and stack algorithms achieve, in general, sub-optimal soft-output performance, whereas the the BCJR algorithm achieves optimum soft-output performance, while being well-suited for hardware implementation. Thus, we consider the BCJR algorithm in the following.

SISO Channel Decoding

For iterative MIMO detection, the SISO decoder obtains extrinsic LLRs from the MIMO detector and computes new extrinsic LLRs for each coded bit (see Figure 2.1). Since we only focus on SISO channel decoding, we omit the superscript ^{2} (shown in Figure 2.1) to denote a priori, intrinsic, and extrinsic LLRs in the remainder of this chapter.

[2]For $K \gtrsim 10$, the VA becomes inefficient and sequential decoding [149] or the stack algorithm [121] can be used, for example.

The SISO channel decoder computes intrinsic a posteriori LLRs

$$L_{k,b}^{\mathrm{D}} \triangleq \log\left(\frac{\mathrm{P}[c_{k,b}=+1,\mathbf{L}]}{\mathrm{P}[c_{k,b}=-1,\mathbf{L}]}\right) \tag{5.1}$$

for each coded bit $c_{k,b}$, followed by computation of the extrinsic LLRs according to

$$L_{k,b}^{\mathrm{E}} = L_{k,b}^{\mathrm{D}} - L_{k,b}^{\mathrm{A}}, \quad \forall k,b \tag{5.2}$$

where $\mathbf{L}$ contains all (extrinsic) LLRs computed by the soft-input soft-output MIMO detector. In the last iteration, the SISO channel decoder additionally computes hard-output estimates for all information bits x_k. To this end, the intrinsic LLR of the information bit x_k is computed in according to

$$L_k^{\mathrm{D}} = \log\left(\frac{\mathrm{P}[x_k=+1,\mathbf{L}]}{\mathrm{P}[x_k=-1,\mathbf{L}]}\right)$$

followed by slicing to $\{+1,-1\}$ using $\hat{x}_k = \mathrm{sign}\big(L_k^{\mathrm{D}}\big)$.

The BCJR Algorithm

For the sake of simplicity of exposition, we assume $R = 1/2$ in the remainder of this section. The BCJR algorithm starts by computing the probabilities required in (5.1) from the trellis-branch probabilities

$$\mathrm{P}[c_{k,b}=\pm 1,\mathbf{L}] = \sum_{(s',s)\in\mathcal{B}_{c_{k,b}}^{(\pm 1)}} \mathrm{P}[S_{k-1}=s',S_k=s,\mathbf{L}] \tag{5.3}$$

where $\mathcal{B}_{c_{k,b}}^{(v)}$ denotes all trellis branches associated with the state tuple $s' \in \mathcal{S}_{k-1}$ and $s \in \mathcal{S}_k$ that correspond to the branch $(x_k|c_{k,1}c_{k,2})$ with $c_{k,b} = v$. The branch probabilities in (5.3) can be rewritten as [34]

$$\mathrm{P}[S_{k-1}=s',S_k=s,\mathbf{L}] = A_{k-1}(s')C_k(s',s)B_k(s) \tag{5.4}$$

where the factors[3] on the right-hand side (RHS) correspond to

$$A_{k-1}(s') = \mathrm{P}\left[S_{k-1} = s', \mathbf{L}^{(1,k-1)}\right] \tag{5.5}$$

$$C_k(s', s) = \mathrm{P}\left[S_k = s, \mathbf{L}^{(k,k)} \middle| S_{k-1} = s'\right] \tag{5.6}$$

$$B_k(s) = \mathrm{P}\left[\mathbf{L}^{(k+1,L)} \middle| S_k = s\right] \tag{5.7}$$

and $\mathbf{L}^{(i,j)}$ contains all LLRs associated with the trellis steps from i to j. Note that the factor $C_k(s', s)$ corresponds to the probability that the encoder is in state $s \in \mathcal{S}_k$ while observing $\mathbf{L}^{(k,k)} = L^{\mathrm{A}}_{k,b}$ $(\forall b)$, given the encoder was in state $s' \in \mathcal{S}_{k-1}$ at step $k-1$. Since each branch (s', s) is associated with a tuple $(x_k | c_{k,1} c_{k,2})$, the probability (5.6) can be computed with the aid of the a priori LLRs delivered by the MIMO detector according to

$$C_k(s', s) = \prod_{b=1}^{2} \frac{\exp\left(\frac{1}{2}(1 + c_{k,b}) L^{\mathrm{A}}_{k,b}\right)}{1 + \exp\left(L^{\mathrm{A}}_{k,b}\right)}. \tag{5.8}$$

If no state transition between $s' \in \mathcal{S}_{k-1}$ and $s \in \mathcal{S}_k$ exists, the branch probability is zero and we set $C_k(s', s) = 0$.

The key idea of the BCJR algorithm [34] is to compute the forward (5.5) and backward messages (5.7) *recursively*, i.e.,

$$A_k(s) = \sum_{s' \in \mathcal{S}_{k-1}} A_{k-1}(s') C_k(s', s) \tag{5.9}$$

$$B_k(s) = \sum_{s' \in \mathcal{S}_{k+1}} B_{k+1}(s') C_{k+1}(s, s'). \tag{5.10}$$

In order to reduce the computational complexity associated with computation of (5.9) and (5.10), only those branches (c', c) need to be considered for which $C_k(s', s) \neq 0$. Assuming that the code has been terminated in the zero-state (at $k = 0$ and $k = L$), the initial values of the recursion in (5.9) and (5.10) are given by

$$A_0(s) = B_L(s) = \begin{cases} 1, & s = 0 \\ 0, & \text{otherwise.} \end{cases}$$

[3] These factors are also termed as "messages" in the context of message-passing in factor graphs [153].

The BCJR algorithm performs SISO channel decoding in two phases as summarized in the following. In the first phase, all forward messages $A_k(s)$ (for all $s \in \mathcal{S}_k$) are computed from $k = 1, \ldots, L$ using (5.9) and (5.8) and stored a memory. In the second phase, all backward messages $B_k(s)$ (for all $s \in \mathcal{S}_k$) are computed from $k = L, \ldots, 1$ according to (5.9). Concurrently, the output probabilities in (5.3) are computed according to (5.4) and (5.3), using $B_k(s)$, $C_k(s', s)$, and the $A_k(s)$ stored before.

The Max-Log Approximation

Due to the presence of transcendental functions in (5.8), straightforward computation of (5.4) would lead to prohibitive computational complexity in practical applications. A common approach to avoid computation of transcendental function is to define

$$\begin{aligned} \alpha_{k-1}(s') &= \log\big(A_{k-1}(s')\big) \\ \gamma_k(s', s) &= \log\big(C_k(s', s)\big) \\ \beta_k(s) &= \log\big(B_k(s)\big) \end{aligned}$$

and to apply the max-log approximation (see Section 4.1.2 for more details) to (5.9) and (5.10) such that

$$\alpha_k(s) \approx \max_{s' \in \mathcal{S}_{k-1}} \big\{\alpha_{k-1}(s') + \gamma_k(s', s)\big\} \tag{5.11}$$

$$\beta_k(s) \approx \max_{s' \in \mathcal{S}_{k+1}} \big\{\beta_{k+1}(s') + \gamma_{k+1}(s, s')\big\}. \tag{5.12}$$

Note that (5.8) can be written as

$$\gamma_k(s', s) = \sum_{c_{k,b}=1}^{2} \left(c_{k,b} \frac{1}{2} L_{k,b}^{\mathrm{A}} + o_b\right) \tag{5.13}$$

where the offset $o_b = \frac{1}{2} L_{k,b}^{\mathrm{A}} - \log\big(1 + \exp\big(L_{k,b}^{\mathrm{A}}\big)\big)$ does not depend on $c_{k,b}$. It is important to note that o_b can safely be omitted during decoding, as only differences are considered in the computation of intrinsic a posteriori LLRs. Hence, the LLRs in (5.1) can be approximated as

$$L_{k,b}^{\mathrm{D}} \approx \max_{(s',s) \in \mathcal{B}_{c_{k,b}}^{(+1)}} \sigma_k(s', s) - \max_{(s',s) \in \mathcal{B}_{c_{k,b}}^{(-1)}} \sigma_k(s', s) \tag{5.14}$$

with

$$\sigma_k(s', s) \triangleq \alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s). \tag{5.15}$$

Application of the max-log approximation in the BCJR algorithm avoids the computation of transcendental function and multiplications, which significantly reduces the complexity of a corresponding VLSI implementation. Note that the max-log approximation entails a small performance loss, which can be be mitigated partially by multiplying the extrinsic a posteriori LLRs (5.2) by a factor of about 0.7 [137, 154].

Windowed BCJR (M-BCJR) Algorithm

The major drawback of the BCJR algorithm is the fact that during the first phase, all $|\mathcal{S}| \cdot L$ forward messages (5.11) need to be stored, which leads to a large amount of memory requirements in most practical applications. A promising solution to counter this problem is to consider a window of $0 < M \ll L$ trellis steps to compute the soft outputs, which yields—if M is chosen sufficiently large—near max-log-optimal performance.[4] This approach is known as the M-BCJR algorithm (or windowed BCJR) in the literature, e.g., [155].

The M-BCJR algorithm divides the trellis in $W = \lceil L/M \rceil$ windows of length M and computes the LLRs as follows. The forward messages are only computed for $k = i, \ldots, i + M - 1$ (where $i = 0, M, 2M, \ldots$ denotes the start index of the window), which only requires $|\mathcal{S}| \cdot M$ messages to be stored. Then, the backward messages are computed in two steps as shown in Figure 5.2. In the first step, temporary backward messages, denoted by $\beta'_{k'}(s)$, are initialized by neutral initial states[5] according to $\beta'_{i+2M-1}(s) = -\infty$ $(\forall s)$ and computed backwards in the window $i + 1$ (i.e., recursively for $k' = i + 2M - 1, \ldots, i + M$). Then, the backward messages $\beta_{k'}(s)$ are initialized with the temporary messages $\beta_{i+M-1}(s) = \beta'_{i+M-1}(s)$, $\forall s$, and computed recursively in the ith window. Concurrently to backward-message computation, the

[4] Considering only the previous $M \approx 5K$ trellis-steps to compute hard-outputs (known as the traceback length), was shown to yield a negligible performance degradation for decoding based on the VA [147].

[5] Except for those cases, where the encoder is forced into certain states.

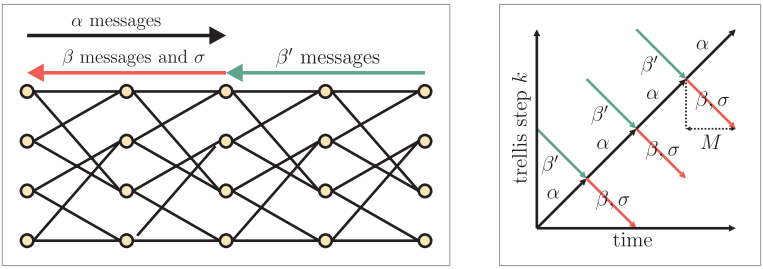


Figure 5.2: Left: illustration of the M-BCJR algorithm using $M = 2$. Right: scheduling of α, β', β, and σ message computation.

corresponding output values (5.15) and intrinsic LLRs $L^{\mathrm{D}}_{k,b}$ are computed according to (5.14) with the aid of the stored forward messages. Finally, the M-BCJR algorithm proceeds to the next window $i+1$. In contrast to the backward messages, the forward messages do not need to be re-initialized and are, therefore, optimal. It is important to note that α, β', and β' (combined with σ) can be computed concurrently in hardware, i.e., by using the schedule illustrated in Figure 5.2.

5.1.2 VLSI Architecture

The high-throughput VLSI architectures for SISO convolutional decoding described below bases on the 8-state radix-2 max-log M-BCJR architecture described in [36], which was initially designed for low-power decoding of turbo codes. In the following, we show several architectural optimizations for high-throughput SISO decoding of CCs.

Overview

Figure 5.3 shows the high-level architecture of the high-throughput max-log M-BCJR decoder. The decoder consists of two input memories (denoted by γ-memory 1 and 2), three message computation units (the α-unit, β-unit, and β'-unit), a α-message memory, and a LLR computation unit (LCU).

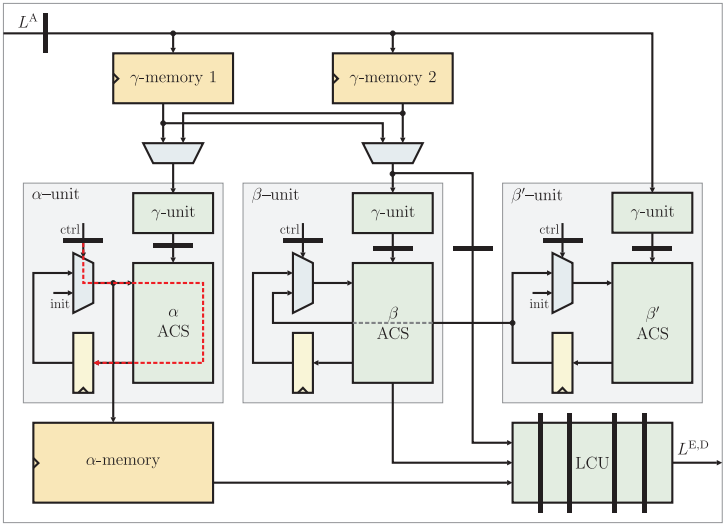


Figure 5.3: Overview of the M-BCJR architecture (the critical path in the ACS is highlighted).

Input Memories The input memories serve as an temporary buffer of the incoming LLRs and are realized by two-port SRAM macro-cells. Since the decoder operates concurrently (within three windows) on the trellis (see Figure 5.2), the LLRs for the α-unit and the β-unit need to be stored, whereas the LLRs required in the β'-unit are directly taken from the input of the decoder (see Figure 5.2).

The α-, β-, and β'-Units The α-unit computes the forward recursion in (5.11) for a window of length M and stores the $M \cdot |\mathcal{S}|$ computed state-metrics into the α-memory (realized by two-port SRAM macro-cells). The β'-unit and the β-unit perform the temporary backward recursion and the backward recursion according to (5.12), respectively.

LLR Computation Unit The LLR computation unit (LCU) computes the estimated output bits as well as the intrinsic or extrinsic max-log LLRs (depending on the operation mode) according to (5.14),

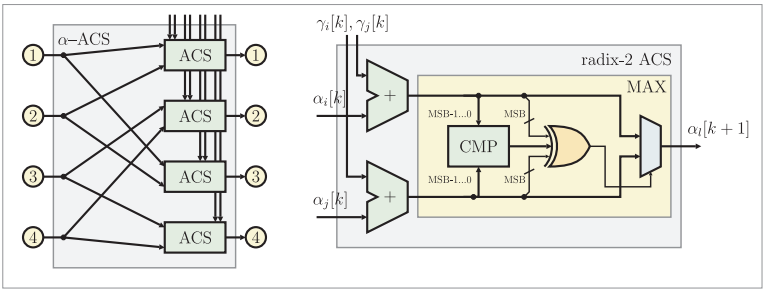


Figure 5.4: Left: architecture of a 4-state radix-2 α–ACS unit. Right: architecture of a radix-2 ACS unit with modulo normalization [36].

based on the forward messages stored in the α-memory and the actual state metrics of the β-unit. In order to shorten the critical path, the computation in the LCU is performed in a pipelined fashion such that the critical path of the resulting architecture is in the add-compare-select (ACS) units.

Add-Compare-Select Units

The forward and backward recursions in (5.11) and (5.12) are performed in the α-, β'-, and β-units of the max-log M-BCJR architecture, respectively. In order to attain high throughput, each unit contains S parallel ACS computation blocks that are connected according to the code generator polynomial (see Section 5.1). Figure 5.4 shows the α-ACS unit for the $K = 3$ code in Table 5.1. Note that this parallel architecture enables to compute one trellis step per clock cycle, which leads to a sustained decoding throughput that corresponds to the clock frequency of the circuit (in terms of information bit per second), i.e., $\Theta_{\text{BCJR}} = f_{\text{clk}}$ information bits per second. It is therefore key to reduce the critical path within the ACS unit in order to maximize the throughput of the decoder. Techniques to further improve the throughput of max-log-based M-BCJR decoders are studied in Section 5.3.2, but lead to reduced hardware-efficiency, in general.

The critical path is highlighted in Figure 5.3 and starts from a multiplexer's control (ctrl) input and ends in a state-metric register.

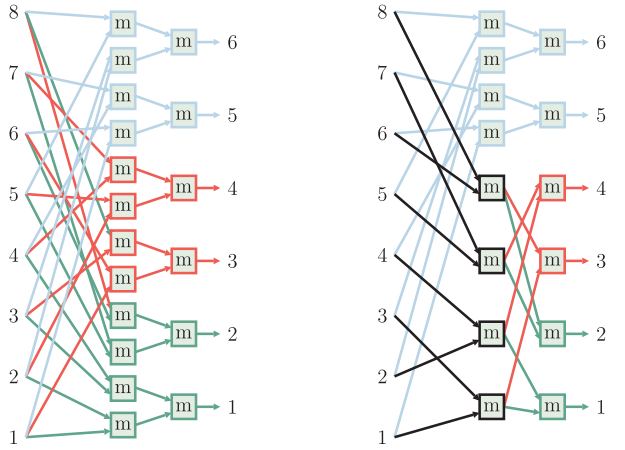
Note that the control inputs of the multiplexers are stored in flip-flops in order to minimize detrimental (in terms of timing) fan-out effects.

Modulo Normalization The forward or backward messages in trellis-based decoders grow during the recursions (5.11) and (5.12), which causes problems for fixed-point implementations (especially for large block-lengths L). In order to avoid a large dynamic range in hardware or costly re-normalization operations—both eventually result in a large critical path of the ACS unit and increased circuit area— a technique known as modulo-normalization [156] is used. This method exploits the fact that the difference between any two state-metrics at a given time instant k is upper-bounded if the dynamic range of the branch metrics in (5.8) is bounded as well. By employing two's complement numerical representation of the state metrics and by choosing sufficiently large word-lengths in the ACS, one can exploit the wrapping property of two's complement numbers, which is known as modulo normalization [156]. Note that modulo-normalization requires minor modifications in the compare-select circuitry (see in Figure 5.4) and remains optimum in a sense that the only performance loss is caused due to quantization of the branch metrics (5.13).

LLR Computation Unit

Computation of the LLRs in (5.14) as well as computation of the estimated bits, requires multiple maximization operations. This computation can be performed in a tree-like fashion in order to minimize the critical path and to reduce the circuit area. Such a straightforward approach requires, however, $(|\mathcal{S}| - 1) \cdot 6$ two-input maximization units. In order to reduce the number of maximizations, a novel technique referred to as partial-maximum sharing (PMS) has been employed. PMS reduces the amount of maximization units (and hence, the circuit area) while not enlarging the critical path of the corresponding circuit.[6] The key idea of PMS is to exploit the structure of the LCU's maximization trees by re-using partial maximization results, which resembles to partial-product sharing (e.g., [157, 158]).

[6] The fan-out of the circuit slightly increases, but synthesis results have not shown any detrimental impact on the critical path.



(a) Initial maximum tree (b) Result of PMS

Figure 5.5: Partial maximum sharing (PMS) applied to the LLR computation tree of a 4-state max-log M-BCJR decoder.

PMS is illustrated in Figure 5.5 for a 4-state max-log M-BCJR decoder. All inputs correspond to branch metrics $\sigma_k(s', s)$ as shown in (5.15). The outputs 1 and 2 are required to compute the binary-valued estimates of the transmitted bits, whereas the outputs 3 to 6 are required to compute the extrinsic or intrinsic LLRs. In this example, the initial maximization tree requires 18 two-input maximum units, whereas PMS only requires 14 units. Table 5.2 demonstrates the benefit of PMS applied to 4- up to 64-state max-log M-BCJR decoders. It can be observed that the benefit of PMS increases for large constraint lengths K and is able to yield a reduction up to 64.6% for the 64-state BCJR implementation compared to a straightforward implementation. Hence, PMS offers a significant reduction in terms of circuit area, while not lowering the critical path of the LCU.

5.1.3 Implementation Results

All techniques described above led to a high-throughput VLSI architecture. In order to assess the VLSI implementation complexity

Table 5.2: Impact of PMS to the number of 2-input maximum operations in the LCU of various max-log M-BCJR architectures.

K	Initial	PMS	Reduction
3	18	14	22.2%
4	42	22	47.6%
5	90	38	57.7%
6	186	70	62.4%
7	378	134	64.6%

associated with the max-log M-BCJR algorithm, five different architectures for the maximum free-distance codes in Table 5.1 have been designed and implemented in 180 nm (1P/6M) CMOS technology. Figure 5.6 shows the two resulting ASICs. The first ASIC contains the 4- to 32-state radix-2 max-log M-BCJR implementations, whereas the second ASIC contains the 64-state M-BCJR decoder only. We emphasize that the 64-state variant is compliant with IEEE 802.11n [2] and—to the best of our knowledge—the first of its kind reported in the literature.

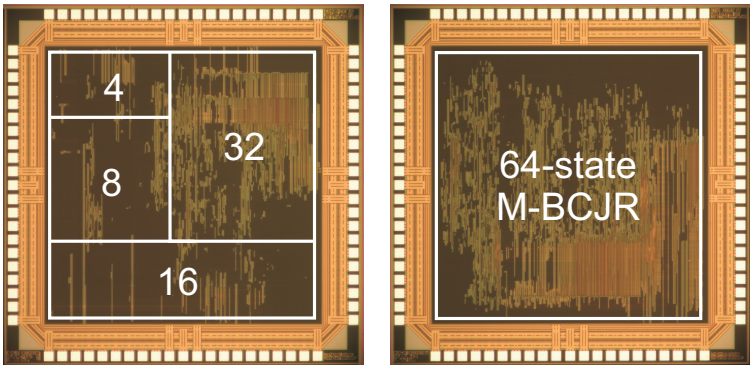
Implementation Results

The implementation results are summarized in Table 5.3. In order to enable comparison, an equal target clock frequency has been used for all implementations.[7] Additionally, the window size has been set to $M = 32$ for all cores, which is favors the results of the 64-state M-BCJR implementation, since smaller window sizes could be used for all implementations with a lower number of states.[8] The maximum achievable throughput (obtained from post place-and-route timing constraints) corresponds to 375 Mbps.

As it can be seen in Table 5.3, the cell area increases proportionally to the number of decoder states, since the decoder area is dominated by the ACS units and the α-memory (see Section 5.1.2). Note that the

[7] Note that the implementations with less than 64-states require lower ACS word-lengths and would, hence, be able to achieve higher clock frequencies.

[8] Note that $M \approx 5K$ has been shown to be sufficient for near-optimum (error-rate) performance, e.g., [147].



(a) 4- to 32-state M-BCJRs units. (b) 64-state M-BCJR unit.

Figure 5.6: ASIC micrographs of high-throughput radix-2 max-log M-BCJR decoders in 180 nm (1P/6M) CMOS technology.

Table 5.3: Implementation results of max-log M-BCJR decoders (radix-2, $M = 32$) for 180 nm (1P/6M) CMOS technology.

States	4	8	16	32	64
Cell area[a] [kGE]	23.1	37.3	60.0	123.8	243.5
Core area [mm^2]	0.22	0.36	0.58	1.20	2.36
Clock freq. [MHz]	375	375	375	375	375
Throughput [Mbps]	375	375	375	375	375
Word-width[b] [bit]	8	9	9	10	10
α–Memory [kbit]	1	2.25	4.5	10	20
Efficiency [kGE/Mbps]	0.06	0.10	0.16	0.33	0.64
E per bit[c] [nJ/bit]	0.35	0.57	0.99	1.96	3.65

[a]One gate equivalent corresponds to the size of a two-input drive-one NAND gate of size 9.7 µm^2.

[b]Corresponds to the number of bits required in the ACS units to enable modulo-normalization [156].

[c]Energy (E) per information bit at 1.8 V core supply with a clock frequency of 150 MHz. The input LLRs are generated from an AWGN channel using BPSK modulation operating at 3 dB SNR.

Table 5.4: Measured energy per information bit [nJ/bit] for max-log M-BCJR decoders with different number of states.

S	ACS Units	α–Memory	LCU	Total
4	0.20 (57.2%)	0.09 (26.4%)	0.06 (16.4%)	0.35 (100%)
8	0.30 (52.2%)	0.17 (30.2%)	0.10 (17.6%)	0.57 (100%)
16	0.47 (47.0%)	0.32 (32.2%)	0.21 (20.8%)	0.99 (100%)
32	0.91 (46.4%)	0.71 (36.3%)	0.34 (17.3%)	1.96 (100%)
64	1.72 (46.9%)	1.33 (36.3%)	0.62 (16.8%)	3.66 (100%)

same behavior is observed for AT-efficiency (in kGE/Mbps) as well as for the energy-efficiency (measured in nJ per information bit), i.e., twice the number of states results in about twice the AT-product and twice the energy required per bit. Hence, high-throughput max-log M-BCJR implementations for codes with large constraint length quickly become prohibitive in terms of circuit area and power consumption.

Energy-Efficiency

Table 5.4 shows energy-efficiency measurement results associated with each max-log M-BCJR decoder in Table 5.3 separately for the main computation units. All measurements have been conducted using clock-gating in the LCU and in the α-memory. The LLRs are generated from a additive white Gaussian noise (AWGN) channel using BPSK modulation at 3 dB SNR.

We can see that the ACS units consume approximately half of the total energy. Moreover, increasing number of states leads to a dramatic energy-efficiency reduction in the α-memory, which is caused by the fact that the memory area scales with the number of states *and* with the number of ACS bits required for modulo-normalization.[9]

Comparison with Hard-Output Viterbi Decoding

Table 5.5 compares the complexity of the proposed 64-state M-BCJR (cf. Table 5.3) with a reference 64-state radix-4 Viterbi decoder de-

[9] As it has been mentioned above, the word-length in the presence of modulo normalization also depends on the constraint length of the code.

Table 5.5: Comparison of the proposed 64-state radix-2 M-BCJR implementation with a 64-state radix-4 Viterbi decoder.

Algorithm	M-BCJR	Viterbi [16]
CMOS process [nm]	180	130
Cell area [kGE]	243.5	68
Clock frequency [MHz]	375	160
Throughput [Mbps]	375	320
Efficiency [kGE/Mbps]	0.64	0.29[a]

[a]In order to account for differences in process technology, the hardware efficiency has been scaled by a factor of 180/130.

scribed in [16]. Both decoders are compliant to IEEE 802.11n [2]. We can see that the M-BCJR implementation is 2.2× less efficient (in terms of kGE/Mbps) compared to that of a Viterbi decoder; this observation enables us to conclude that SISO capability entails about a two-fold increase in terms of silicon complexity (for a given throughput) compared to that of hard-output decoding of CCs.

5.2 Low-Density Parity Check Codes

Low-density parity check LDPC codes have been developed by Gallager in 1962 [159] and are able to attain excellent error-correction performance. However, decoding of LDPC codes was not feasible in practice at this time. In 1999, LDPC codes have been rediscovered by MacKay [160] and Richardson *et al.* [161] and since then, belong to the most promising error-correcting codes. In particular, quasi-cyclic (QC) LDPC codes [162, 163] offer excellent error-correction capability while enabling high decoding throughput at low implementation complexity, e.g., [164–166]. Hence, QC-LDPC codes have been considered many modern wireless communication standards, e.g., DVB-S2 [167], IEEE 802.16 [10], and IEEE 802.11n [2].

In the remainder of this section, we describe a configurable high-throughput architecture for SISO decoding of QC-LDPC codes. The architecture has been optimized for IEEE 802.11n [2] and integrated

in 180 nm CMOS technology. Performance and complexity measurements and comparison with existing decoders conclude this section.

5.2.1 Quasi-Cyclic LDPC Codes and Decoding

LDPC codes are linear block codes satisfying

$$\mathbf{H}\mathbf{x} = \mathbf{0}_{M\times 1} \tag{5.16}$$

in GF(2), where $\mathbf{x}$ is the N-dimensional binary-valued code vector, $\mathbf{0}_{M\times 1}$ denotes an M-dimensional all-zero vector, and $\mathbf{H}$ is a sparse binary-valued $M \times N$ parity check matrix [159]. QC-LDPC codes are defined by a $M_p \times N_p$ LDPC matrix prototype $\mathbf{H}_p$. The parity check matrix $\mathbf{H}$ in (5.16) is constructed from the LDPC matrix prototype by replacing each entry in $\mathbf{H}_p$ by a $Z \times Z$ cyclic-shift matrix $\mathbf{P}^c$ (i.e., $M = M_p Z$ and $N = N_p Z$), where c is equal to the corresponding entries in $\mathbf{H}_p$. The cyclic-shift matrices are defined as $\mathbf{P}^c = \prod_{i=1}^{c} \mathbf{P}^1$ (for $c > 0$) with

$$\left[\mathbf{P}^1\right]_{i,j} = \begin{cases} 1, & (i \bmod Z) + 1 = j \\ 0, & \text{otherwise} \end{cases}$$

$\mathbf{P}^0 = \mathbf{I}_Z$, and $\mathbf{P}^- = \mathbf{0}_{Z\times Z}$. The entries of $\mathbf{H}_p$ satisfy $[\mathbf{H}_p]_{m,n} < Z$ $(\forall m, n)$. For example, the rate-5/6, $Z = 81$ QC-LDPC matrix prototype of the IEEE 802.11n standard [2] is defined as

$$\mathbf{H}_p = \begin{bmatrix} 13 & 48 & 80 & 66 & 4 & 74 & 7 & 30 & 76 & 52 & 37 & 60 & - & 49 & 73 & 31 & 74 & 73 & 23 & - & 1 & 0 & - & - \\ 69 & 63 & 74 & 56 & 64 & 77 & 57 & 65 & 6 & 16 & 51 & - & 64 & - & 8 & 9 & 48 & 62 & 54 & 27 & - & 0 & 0 & - \\ 51 & 15 & 0 & 80 & 24 & 25 & 42 & 54 & 44 & 71 & 71 & 9 & 67 & 35 & - & 58 & - & 29 & - & 53 & 0 & - & 0 & 0 \\ 16 & 29 & 36 & 41 & 44 & 56 & 59 & 37 & 50 & 24 & - & 65 & 4 & 65 & 52 & - & 4 & - & 73 & 52 & 1 & - & - & 0 \end{bmatrix}.$$

The parity check in (5.16) can be represented as a bipartite graph consisting of N variable nodes (denoted by v) and M check nodes (denoted by c). Variable nodes are associated with the LLRs of the transmitted bits $L_v = L(x_v)$, $v = 1, 2, \ldots, N$, and check nodes with parity checks, i.e., the mth parity check node is connected to the nth variable node if $[\mathbf{H}]_{m,n} = 1$.

Layered Decoding of LDPC Codes

LDPC decoding can be represented as message passing (MP) on the bipartite graph [153]. The "standard" MP schedule consists of two

phases (known as flooding MP [168]). In the first phase, all messages, from the variable to the check nodes (denoted by $Q_{v,c}$) are computed. In the second phase, all messages from check to variable nodes (denoted by $R_{c,v}$) are computed. Then, the intrinsic output LLRs L_v^{D} $(\forall v)$ can be computed and the process is repeated until the algorithm converges to a valid code-word or until a maximum number of repetitions have been performed. Flooding MP is frequently used in the literature, e.g., [169, 170], but has three major disadvantages: i) both phases require a different set of arithmetic operations, ii) both message types ($Q_{v,c}$ and $R_{c,v}$) as well as the a priori LLRs need to be stored, and iii) the convergence behavior is rather slow [168].

Layered LDPC decoding [168,171] avoids the drawbacks of flooding MP. The algorithm only requires to store $R_{c,v}$ messages and Q_v-values. The latter are initialized by $Q_v = L_v^{\mathrm{A}}$ $(\forall v)$, which avoids separate storage for the LLRs. Layered MP computes [168, 171]

$$R_{c,v}^{\mathrm{new}} = 2\tanh^{-1}\left(\prod_{v'\in\mathcal{V}(c)\setminus v}\tanh\left(\frac{Q_v - R_{c,v'}}{2}\right)\right) \tag{5.17}$$

where the set $\mathcal{V}(v)$ contains all variable nodes that are connected with the check node c. Then, the Q_v-values are being updated according to

$$Q_v^{\mathrm{new}} = Q_v - R_{c,v} + R_{c,v}^{\mathrm{new}}. \tag{5.18}$$

This procedure is repeated until a valid codeword has been found (by checking whether the hard-output estimates $\hat{x}_v = \mathrm{sign}(Q_v^{\mathrm{new}})$ satisfy the parity check in (5.16)) or a maximum number of iterations has been performed. In order to avoid computation of transcendental functions in hardware, the update rule (5.17) can be simplified using min-sum MP [153]

$$\begin{aligned} |R_{c,v}^{\mathrm{new}}| &= \min_{v'\in\mathcal{V}(v)\setminus v}|Q_v - R_{c,v'}| \\ \mathrm{sign}\big(R_{c,v}^{\mathrm{new}}\big) &= \prod_{v'\in\mathcal{V}(v)\setminus v}\mathrm{sign}(Q_v - R_{c,v'}) \end{aligned} \tag{5.19}$$

which can further be improved by refining (5.19) according to the

offset min-sum (OMS) MP rule [172]

$$R_{c,v}^{\text{new}} \leftarrow \max\left\{R_{c,v}^{\text{new}} - \beta, 0\right\} \cdot \text{sign}(Q_v - R_{c,v'})\,. \tag{5.20}$$

Setting the offset parameter to $\beta = 0.15$ in (5.20) has shown to yield a near-optimum performance for the codes used in IEEE 802.11n [2], for example.

Decoding Algorithm for QC-LDPC codes

The algorithm employed in the VLSI architecture exploits the value-reuse properties of OMS as described in [165,166] and employs a novel technique called message clipping (MC), which allows to trade memory requirements for performance. The decoding algorithm is summarized in Algorithm 2. The memory requirements corresponds to the Z-dimensional vectors $\mathbf{q}_n^i$ (i.e., containing all Q_v-values) and the $R_{c,v}$-messages from check to variable nodes $\mathbf{r}_{m,n}^i$. All vector operations in Algorithm 2 are performed element-wise, except for the cyclic shifts.

Initialization On lines 1-2 of Algorithm 2, the $\mathbf{q}_n^0$ $(\forall n)$ values are initialized by the a priori LLRs (where $L(x_{(n-1)Z+1})$ denotes the LLR associated with the bit $x_{(n-1)Z+1}$) and all messages $\mathbf{r}_{m,n}^0$ $(\forall m, n)$ initialized with zero. The algorithm performs up to I iterations (lines 3-19) and operates row-wise (for each row m of $\mathbf{H}_p$). Message computation is then performed iteratively for the columns $n \in \mathcal{N}_m$ of $\mathbf{H}_p$ for which $\left[\mathbf{H}_p\right]_{m,n} \neq {}'-'$ (see line 6 and 12 Algorithm 2), i.e.,

$$\mathcal{N}_m = \left\{n \in \mathbb{Z} \,|\, \left[\mathbf{H}_p\right]_{m,n} \neq {}'-'\right\}. \tag{5.21}$$

Message computation of layered MP can be divided into two phases: the first is called MIN phase (lines 6-11) and the second is called SEL phase (lines 12-17).

MIN Phase The $\mathbf{r}_{m,n}^{i-1}$-message is subtracted from a cyclically-shifted version (by $c = \left[\mathbf{H}_p\right]_{m,n}$) of $\mathbf{q}_n^{i-1}$ and stored in a temporary vector $\mathbf{t}_n$ (cf. line 6). OMS [172] and its value-reuse properties [165, 166] only require to compute the minimum $\mathbf{m}_1$ and the second minimum $\mathbf{m}_2$ for each entry of $\mathbf{t}_n$ $(\forall n)$. Both minima are being

Algorithm 2 Layered-OMS QC-LDPC Decoder

1: $\mathbf{r}_{m,n}^{0} \leftarrow \mathbf{0}_{Z\times 1}$, $n = 1, 2, \ldots, N_b$, $m = 1, 2, \ldots, M_b$
2: $\mathbf{q}_{n}^{0} \leftarrow \left[L(x_{(n-1)Z+1}) \; L(x_{(n-1)Z+2}) \; \cdots \; L(x_{nZ}) \right]^{T}$, $\forall n$
3: **for** $i = 1, 2, \ldots, I$ **do**
4: **for** $m = 1, 2, \ldots, M_b$ **do**
5: $\mathbf{m}_1 \leftarrow \infty \cdot \mathbf{I}_{Z\times 1}$, $\mathbf{m}_2 \leftarrow \infty \cdot \mathbf{I}_{Z\times 1}$, $\mathbf{s} \leftarrow \mathbf{I}_{Z\times 1}$
6: **for** $n \in \mathcal{N}_m$ **do**
7: $c = \left[\mathbf{H}_p\right]_{m,n}$, $\mathbf{t}_n \leftarrow \mathbf{P}^{c}\mathbf{q}_{n}^{i-1} - \mathbf{r}_{m,n}^{i-1}$
8: $[\mathbf{m}_1, \mathbf{x}, \mathbf{v}] \leftarrow \min\left\{\mathbf{m}_1, |\mathbf{t}_n|\right\}$
9: $\mathbf{m}_2 \leftarrow \min\{\mathbf{m}_2, \mathbf{x}\}$
10: $\mathbf{s} \leftarrow \mathbf{s} \cdot \mathrm{sign}(\mathbf{t})$
11: **end for**
12: **for** $n \in \mathcal{N}_m$ **do**
13: $c = \left[\mathbf{H}_p\right]_{m,n}$, $\mathbf{t}_n \leftarrow \mathbf{P}^{c}\mathbf{q}_{n}^{i-1} - \mathbf{r}_{m,n}^{i-1}$
14: $\mathbf{m} \leftarrow \mathrm{sel}(\mathbf{m}_1, \mathbf{m}_2, \mathbf{v}, n)$
15: $\mathbf{r}_{m,n}^{i} \leftarrow \mathrm{clip}\big(\mathbf{s} \cdot \mathrm{sign}(\mathbf{t}_n) \cdot \max\left\{\mathbf{m} - \underline{\beta}, 0\right\}, \mathbf{t}_n\big)$
16: $\mathbf{q}_{n}^{i} \leftarrow \mathbf{P}^{c'}\left(\mathbf{t}_n + \mathbf{r}_{m,n}^{i}\right)$, $c' = Z - \left[\mathbf{H}_p\right]_{m,n}$
17: **end for**
18: **end for**
19: **end for**

computed iteratively as shown on lines 8 and 9, where the minimum on line 8 returns the minimum of $\mathbf{m}_1$ or $\mathbf{t}_n$, a temporary vector $\mathbf{x}$ which contains the values not corresponding to the minimum (i.e., the second-lowest values), and an index vector where $[\mathbf{v}]_k$ is set to the current index n if a new minimum (i.e., corresponding to $[\mathbf{t}_n]_k$) has been found. The vector $\mathbf{s}$ contains Z signs, which are used later in the SEL phase.

SEL Phase The new Q and R messages are computed iteratively for $n = 1, \ldots, N_b$. To this end, the temporary vector $\mathbf{t}_n = \mathbf{P}^{c}\mathbf{q}_{n}^{k-1} - \mathbf{r}_{m,n}^{k-1}$ is computed first (line 14). Then, the sel-function (line 14) compares each of the Z indices $[\mathbf{v}]_k$ (for $k = 1, \ldots, Z$) with n and yields the corresponding entry of $[\mathbf{m}_1]_k$ if $\mathbf{v}_k = n$ and $[\mathbf{m}_2]_k$ otherwise. The new $\mathbf{r}_{m,n}^{i}$-vector is computed by using OMS MP [172]

(cf. line 15) with the vector $\underline{\beta} = \beta \cdot \mathbf{I}_{Z \times 1}$, where only one offset value β is used for all Z entries. The new $\mathbf{q}^i_{m,n}$ vector is computed on line 17 and cyclically shifted using the inverse rotation $c' = Z - c$, such that $\mathbf{P}^c \mathbf{P}^{c'} = \mathbf{I}_{Z \times Z}$.

Message Clipping

Since layered LDPC decoding performs updates of the $\mathbf{q}^i_{m,n}$-vector using the results of the previous iteration (cf. line 16) according to (5.18), the dynamic range of the Q_v-messages can get very large, in general. In order to reduce the amount of bits required to store all $\mathbf{q}^i_{m,n}$-vectors $(\forall n, m)$, the maximum magnitude of the Q_v-messages can be clipped. Unfortunately, such a straightforward approach yields poor error-rate performance. In order to combat this problem, a novel approach, referred to as MC, is described below. Instead of clipping the Q_v-values, the $R_{c,v}$-messages are clipped (on line 15) according to

$$\text{clip}(\mathbf{r}, \mathbf{t}) = \max \Big\{ \min \big\{ \mathbf{r}, Q_{\max} - \mathbf{t} \big\}, -Q_{\max} - \mathbf{t} \Big\}$$

which ensures that that $\left| [\mathbf{q}^i_n]_k \right| \leq Q_{\max}$ $(\forall n, k)$ and $Q_{\max}$ denotes the MC parameter; this offers to trade-off memory consumption (to store the Q_v-messages) for error-rate performance (corresponding simulation results are shown in Section 5.2.2).

5.2.2 VLSI Architecture

Figure 5.7 outlines the architecture of the QC-LDPC decoder. The decoder consists of two memories, i.e., one for the Q_v-values and one for the $R_{c,v}$-messages, a cyclic shifter (CS), and a pool of node computation units (NCUs). To enable reconfigurability for different LDPC matrix prototypes, the architecture contains a configurable control unit and a sequence (SEQ) memory.

In order to achieve high throughput, the decoder operates on Z rows of $\mathbf{H}$ in a parallel manner (using Z NCUs). The architecture computes one entry of $\mathbf{H}_p$ per clock cycle, so that at most N_b clock cycles are required per row of the LDPC matrix prototype. The throughput is further increased by parallel computation of the MIN and SEL

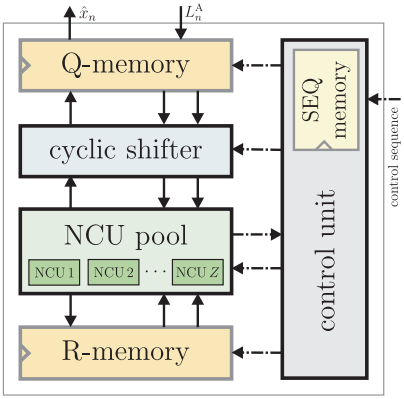


Figure 5.7: QC-LDPC decoder architecture overview.

phases (line 6-12 and 12-17, respectively) of Algorithm 2. This parallel computation requires that the MIN and SEL phases do not access the same data at the same time. To avoid memory access contention, the MIN phase is performed on row $m+1$, whereas the SEL phase operates on row m. Furthermore, the SEL phase never accesses the nth column before it will be used in the MIN phase, to maintain convergence of layered LDPC decoding. These memory access constraints are enforced by the control unit (see Section 5.2.2).

Node Computation Units

In order to enable parallel computation of the SEL and MIN phases, each of the Z NCUs are divided into a SEL unit and MIN unit by an intermediate pipeline stage as shown in Figure 5.8.

MIN Unit The kth MIN unit ($k = 1, \ldots, Z$) iteratively computes $[\mathbf{m}_1]_k$, $[\mathbf{m}_2]_k$, $[\mathbf{v}]_k$, and $[\mathbf{s}]_k$ for all $n \in \mathcal{N}_{m+1}$ on row $m+1$ (cf. lines 5-11 of Algorithm 2) by using the inputs $\left[\mathbf{r}_{m+1,n}^{i-1}\right]_k$ and $\left[\mathbf{P}^c \mathbf{q}_n^{i-1}\right]_k$. When the MIN phase of row $m+1$ has finished, both minima (i.e., $\mathbf{m}_1$ and $\mathbf{m}_2$), the index vector $\mathbf{v}$, and the sign-vector $\mathbf{s}$ are passed from the MIN to the SEL units.

SEL Unit The SEL unit iteratively computes the new clipped output messages $\mathbf{r}_{m,n}^{i}$ and $\mathbf{t}+\mathbf{r}_{m,n}^{i}$ as shown on lines 12-17 of Algorithm 2 using $\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{v}$, and $\mathbf{s}$ from the MIN unit and $\mathbf{r}_{m+1,n}^{i-1}$ as well as $\mathbf{P}^{c}\mathbf{q}_{n}^{i-1}$ from the R- and Q-memory, respectively.

Partial Parity Check During computation of the new Q_v-values and $R_{c,v}$-messages, each SEL unit performs a partial parity check (PPC) of (5.16), i.e., checks whether $\mathbf{h}_{m'}^{T}\hat{\mathbf{x}} = 0$, where $\mathbf{h}_{m'}^{T}$ stands for the m'th row (for $m' = 1, 2, \ldots, M$) of $\mathbf{H}$ in (5.16) and $\hat{\mathbf{x}}$ corresponds to the binary-valued estimates obtained from the sign-bits of $\mathbf{q}_{n}^{i}$. After processing all m' rows of $\mathbf{H}$, the decoding procedure is terminated prematurely if all PPCs are satisfied. It is important to note that combining all PPCs does, in general, not correspond to the parity check in (5.16), since the evaluation of the PPCs are done sequentially and some estimates might change during computation of all rows (since they result from the $\mathbf{q}_{n}^{i}$-values). We emphasize, however, that the proposed approach *never* terminates the decoder if (5.16) is *not* satisfied. However, situations occur, where (5.16) is satisfied but all PPCs are not. Hence, the presented approach is slightly sub-optimal (in terms of the number of performed iterations), but the average number of iterations is still reduced significantly for medium to high SNRs. Hence, the PPC approach still leads to an improvement in terms of energy-efficiency at high SNR (corresponding measurement results are provided in Section 5.2.3).

Cyclic Shifter

Cyclic shifts according to the entries of $\mathbf{H}_p$ are required when the MIN and SEL units read data from the Q-memory and when the SEL unit writes new data to the Q-memory (see Figure 5.8). These cyclic shifts are performed in the CS. Since $Z \leq Z_{\max}$ (where $Z_{\max}$ denotes the maximum sub-block size supported by the architecture) can be different for each LDPC matrix prototype (i.e., depending on the code-block size) a flexible unit is required that is able to perform an arbitrary cyclic shift $0 \leq c < Z$. Several architectures that perform cyclic shifts of a subset of $Z_{\max}$ have been described in the literature. A corresponding survey can be found in [173]. However, most of the

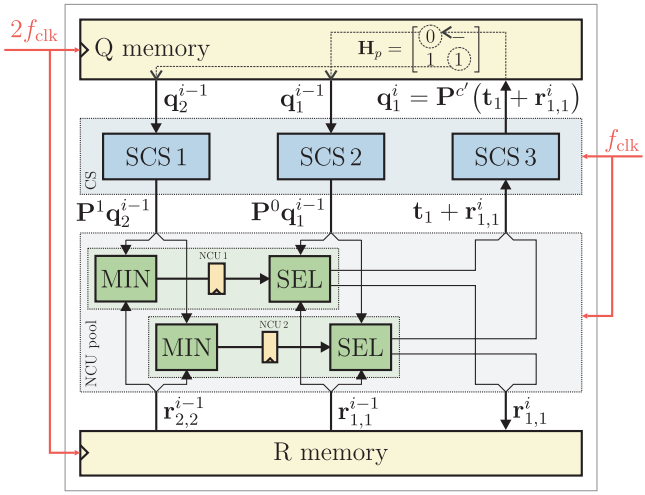


Figure 5.8: Architectural details and decoding schedule of the QC-LDPC decoder for $Z = 2$ and $N_b = M_b = 2$.

proposed architectures lead to either irregular structures, result in large circuit and interconnection area, or are only suited for a small number of different Z-values.

In order to maintain reconfigurability, we propose a simple but highly-efficient subset cyclic shifter (SCS), which is able to perform shifts for *any* subset of $Z_{\max}$ by an arbitrary shift amount $0 \leq c < Z$. The SCS architecture is depicted in Figure 5.9 and consists of a shifter control (SC) unit, two barrel rotators (denoted by BR 1 and BR 2), and an output multiplexer stage. The first barrel rotator (BR 1) performs a cyclic left-shift by $c_1 = Z_{\max} - Z + c$, while BR 2 left-shifts the input by $c_2 = c$. The shift amounts c_1 and c_2 are computed in the SC unit. The output multiplexer stage selects the outputs $1, \ldots, Z-c$ from BR 2 and the outputs $Z - c + 1, \ldots, Z$ from BR 1.

The key advantages of the proposed SCS are i) the low circuit area and ii) its fast operation. The total number of 2-to-1 multiplexers required in each SCS corresponds to

$$B_q\big(2Z_{\max}\lceil\log_2(Z_{\max})\rceil + Z_{\max}\big)$$

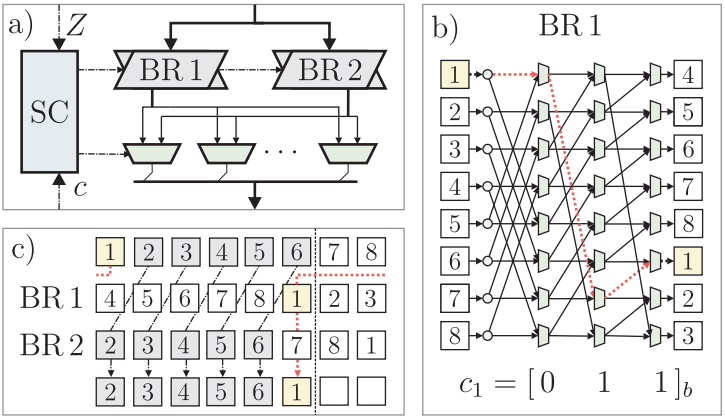


Figure 5.9: Subset cyclic shifter architecture: a) overview, b) barrel rotator, and c) operation principle for $Z_{\max} = 8$, $Z = 6$, and $c = 1$.

where B_q denotes the number of bits required for each entry of $\mathbf{q}$. Furthermore, each SCS only consists of $\lceil \log_2(Z_{\max}) \rceil + 1$ multiplexer stages, which, thanks to its regular structure, enables pipelining and hence allows to achieve high throughput.

Memories for Q- and R-Messages

Since the Q-memory and the R-memory require two write operations and one read operation per clock cycle for all Z messages, the memories have been designed such that one (read or write) address corresponds to $Z_{\max}$ messages. To increase the memory bandwidth without (noticeably) increasing the area, we used a double-clocking strategy, i.e., both memories operate at twice the clock frequency of the remaining logic, i.e., $f_{\text{mem}} = 2f_{\text{clk}}$, which is illustrated in Figure 5.8. Since the timing characteristics of the available memories were sufficiently fast, double-clocking offers a two-fold increase in terms of memory bandwidth without leading to a significant area overhead.

Configurable Control Unit

The control unit consists of the SEQ memory and a simple controller, which generates all control signals for the decoder architecture. During the initialization phase of the LDPC decoder, the control unit can be configured with the OMS scaling parameter β, the sub-block size Z, and the maximum number of iterations I. Additionally, a control sequence needs to be loaded into the sequence (SEQ) memory prior to decoding. This control sequence contains all necessary information on $\mathbf{H}_p$ in combination with the decoding schedule (i.e., required memory addresses, cyclic shifts etc.). During operation, the control words are being translated to corresponding signals and addresses. The key advantages of the proposed approach is that the decoder remains fully configurable, i.e., all possible QC-LDPC matrices that fit into the allocated memories can be processed, and the complexity of the control unit remains low.

Architectural Optimizations for IEEE 802.11n

The architecture described above has been optimized for IEEE 802.11n [2] in order to determine the design parameters for the ASIC implementation shown in Section 5.2.3.

Architectural Optimizations The decoder has been designed to support at least all QC-LDPC matrix prototypes of IEEE 802.11n [2] while achieving a decoding throughput higher than 600 Mbit/s. Since the standard only requires $Z \in \{27, 54, 81\}$, 81 NCUs have been instantiated. To reduce the power consumption, the NCU pool has been divided in three blocks consisting of 27 NCUs. Each of the three NCU blocks can be switched off using clock gating, which is used to reduce the power consumption for the modes $Z \in \{27, 54\}$.

Numeric Precision Optimization Since IEEE 802.11n supports many different modulation and coding schemes, the numeric precision requirements have been evaluated using frame error rate (FER) simulations in an AWGN channel. Figure 5.10 compares the FER of layered message passing using the floating-point (fp) sum-product algorithm (SPA) with layered OMS ($\beta = 0.15$). Note that OMS only

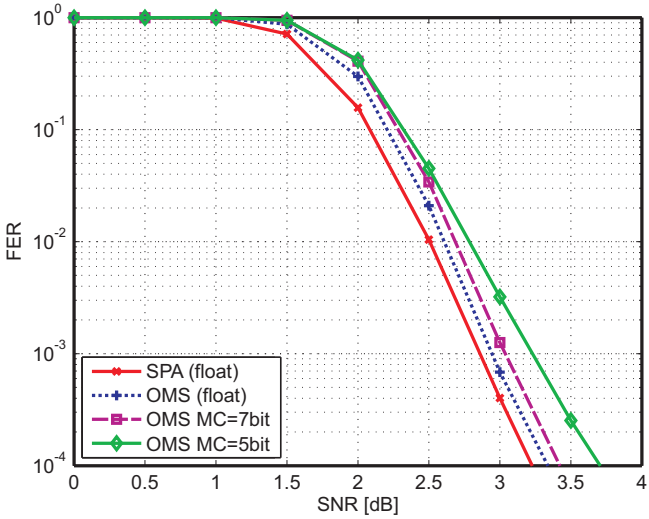


Figure 5.10: Frame error rate (FER) comparison using the $R = 1/2$, $Z = 81$ code [2] in an AWGN channel using BPSK modulation.

loses approximately 0.2 dB to the SPA. The impact of MC is shown by using 7 bits and 5 bits. Note that both OMS simulations include early termination based on the PPCs described in Section 5.2.2. MC to 5 bits leads to a 0.45 dB SNR loss compared to MC using 7 bits (at 10^{-3} FER) but reduces the amount of bits required in the Q-memory by 28%. Thus, we decided to limit the Q-values to $B_q = 5$ bit. The R-messages require $B_r = 5$ bit and the input LLRs have been quantized to 5 bit according to [172]. The curve associated with OMS 5 bit in Figure 5.10 corresponds to the FER performance of the final implementation.

Q- and R-Memory Design IEEE 802.11n requires at most $24 \cdot 81$ Q_v-values and the R-memory requires at most $88 \cdot 81$ $R_{c,v}$-messages, where 88 corresponds to the maximum number of sub-blocks not equal to '-' in IEEE 802.11n [2]. Therefore, we instantiated eight 64×102 single-port SRAM modules (total 52'224 bit) for the R-memory and three 32×135 two-port SRAMs (total 12'960 bit) for the Q-memory.

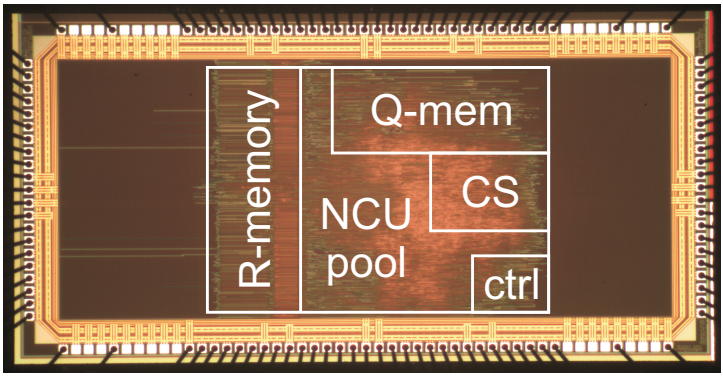


Figure 5.11: ASIC photo of the QC-LDPC decoder in 180 nm CMOS technology (there is unused circuit area on both sides of the ASIC).

The number of bits per sequence word is $B_s = 42$ bit and memory for a maximum of 128 sequence words has been allocated.

5.2.3 Implementation Results

Figure 5.11 shows a ASIC photo of the fabricated QC-LDPC decoder in 180 nm (1P/6M) CMOS technology. The key figures of the QC-LDPC decoder implementation are summarized in Table 5.9. The implementation requires 3.39 mm^2 core area and achieves 208 MHz and 416 MHz for the slow logic-clock and the fast memory-clock, respectively. Table 5.6 shows a detailed area breakdown of the QC-LDPC decoder. It can clearly be observed that most of the circuit area is occupied by the Q- and R-memories. The NCU Pool and the cyclic shifter require approximately a third of the decoder's area and the circuit complexity of the configurable control unit is low.

Table 5.7 shows the sequence lengths for all possible codes in IEEE 802.11n as well as the resulting throughput for $f_{\text{clk}} = 208$ MHz using a maximum number of five iterations. The throughput can be computed according to

$$\Theta = \frac{Z \cdot N_b \cdot R}{L \cdot I + 32} f_{\text{clk}}$$

Table 5.6: Detailed area breakdown of the QC-LDPC decoder implementation in 180 nm (1P/6M) CMOS technology.

Unit	mm^2	kGE[a]	%
Control unit	0.10	10.3	3.0
NCU pool	0.59	60.8	17.4
Q memory	0.83	85.6	24.5
R memory	1.16	119.6	34.2
Cyclic shifter	0.45	46.4	13.3
Miscellaneous[b]	0.26	26.8	7.6
Total	3.39	349.5	100

[a] One gate equivalent (GE) corresponds to the area of a two-input drive-one NAND gate of size 9.7 μm^2.

[b] Denotes remaining logic, i.e., pipeline registers, logic required for the input/output interface, etc.

where 32 additional clock cycles are required to load and flush the pipeline. The maximum achievable throughput for the $R = 5/6$, $Z = 81$ code corresponds to 780 Mbit/s and exceeds the throughput requirements of IEEE 802.11n.

Energy-Efficiency

Table 5.8 shows energy-efficiency measurement results of the implemented ASIC. The impact of early-termination using PPC, as well as the impact of clock gating is shown at different SNRs for different $R = 1/2$ codes [2]. For higher SNRs, the energy-efficiency improves significantly, since it becomes more likely that the PPCs are satisfied and the decoder can be terminated early. For small code blocks (i.e., $Z = 27$), the power consumption can be dramatically reduced if clock gating is used, since part of the NCUs can be disabled during decoding of the smaller block-sizes (i.e., for $Z = \{27, 54\}$). Note that for $Z = 27$, clock-gating combined with PPC improves the energy-efficiency up to 63% at high SNRs, whereas in the low-SNR regime up to 48% improvement can be achieved.

Table 5.7: Sequence length L and throughput Θ for all QC-LDPC codes of IEEE 802.11n.

Z	Rate	L	Θ [Mbps]
27	1/2	94	134
	2/3	88	190
	3/4	89	212
	5/6	95	222
54	1/2	88	286
	2/3	90	373
	3/4	90	419
	5/6	89	471
81	1/2	91	415
	2/3	89	565
	3/4	86	656
	5/6	80	780

Comparison

Table 5.9 compares the proposed implementation with dedicated QC-LDPC decoders for IEEE 802.11n. Due to differences in process technologies, fair area, speed, and power comparisons are difficult to state. However, the area of all four designs is comparable when considering technology scaling. Note that the implementation described above achieves the lowest clock frequency, which is mainly due to the process technology and the detrimental input/output timing of the off-chip drivers. Nevertheless, our implementation achieves a throughput of up to 780 Mbps, which is sufficient to fulfill the requirements of the standard. Additionally, the achieved throughput is comparable to that of [164] and even higher than that of [174].

5.3 Turbo Codes

In 1993, Berrou *et al.* [175,176] first described turbo codes which represented a breakthrough in channel coding due to the capability to achieve near-Shannon-capacity with practical encoding and decoding

Table 5.8: Energy-efficiency for rate-1/2 QC-LDPC codes.

Z	SNR	no power save		clk–gate		clk–gate & PPC	
		nJ/bit	%	nJ/bit	%	nJ/bit	%
27	1	11.5	100	6.0	52	6.0	52
	3	11.9	100	6.2	52	5.6	47
	7	11.6	100	6.1	53	3.1	27
54	1	5.6	100	4.4	79	4.4	79
	3	5.8	100	4.5	79	4.2	73
	7	5.6	100	4.4	79	2.2	40
81	1	3.8	100	3.8	100	3.8	100
	3	4.0	100	4.0	100	3.7	92
	7	3.9	100	3.9	100	1.9	49

Table 5.9: Comparison of QC-LDPC Decoders for IEEE 802.11n.

	This work	[166]	[164][a]	[174][b]
Technology [nm]	180	130	130	65
Core area [mm^2]	3.39	1.85	1.9	0.74
Memory area [mm^2]	1.99	1.04	n.a.	0.26
Cell area [kGE]	144.3[c]	99.9	195	217
Max. clock freq. [MHz]	208	500	400	240
Max. throughput [Mbps]	780	1618	1000[d]	410

[a] Corresponding to the pipelined version in [164].
[b] Corresponding to the $K = 4$, $N_{\mathrm{c2v}} = 5$, and $N_{\mathrm{SO}} = 8$ decoder in [174].
[c] The memory area has been excluded (cf. Table 5.6).
[d] The maximum throughput is given for 2.2 dB SNR.

schemes. So far, turbo codes have been adopted in various wireless communication standards such as the high-speed downlink packet access (HSDPA) standard [139] by the 3GPP consortium and also in the upcoming LTE standard [139], which bases on MIMO technology.

Most of the turbo decoders proposed in the literature were designed with mobile and low-rate applications in mind and hence, achieve a throughput in the order of several ten Mbps, e.g., [36, 37]. MIMO systems usually require a throughput ranging from several hundred Mbps up to Gbps. Interleaving it is known to be one of the main bottlenecks for high-throughput turbo decoding, since parallel and interleaved access to memories is difficult, in general. Recently, the concept of contention-free interleavers has been proposed, e.g., [177,178], which aims at alleviating the interleaver bottleneck by enabling parallel access to interleaved data. Contention-free interleavers are key to enable high-throughput turbo decoding in practical systems.

In this section, we briefly review the basics of turbo decoding and describe a high-throughput turbo decoder architecture for the 3GPP LTE standard. The proposed turbo decoder has been implemented in 130 nm CMOS technology and is compared to reference implementations in terms of throughput and hardware-efficiency.

5.3.1 Decoding of Turbo Codes

Systematic parallel-concatenated turbo codes (PCTCs) are usually generated from two recursive CCs (RCC) and an interleaver (see Section 5.1). The binary-valued data bits x_k (for $k = 1, \ldots, N$) are directly fed to the output (referred to as systematic bits) as well as into the first RCC which generates parity bits $c_{k,1}$ ($\forall k$). The second RCC is fed by an interleaved version of x_k and generates parity bits $c_{k,2}$, $\forall k$. Note that systematic PCTCs naturally have rate 1/3; higher rates can be derived by the use of puncturing [147]. The encoding procedure is shown in Figure 5.12. In the ensuing discussion, we focus on the 8-state code with feed-forward generator $p_{\text{ff}} = 15_o$ and feedback generator $p_{\text{fb}} = 13_o$ as defined in 3GPP LTE standard [11].

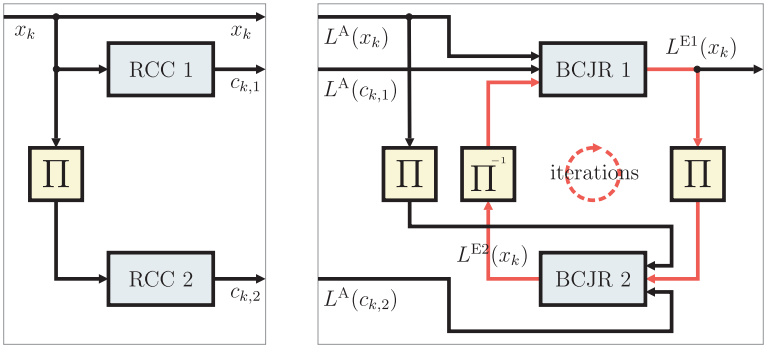


Figure 5.12: Left: turbo encoder consisting of two recursive convolutional codes (RCCs) and an interleaver (denoted by Π). Right: turbo decoder principle (de-interleaving is denoted by Π^{-1}).

The Principle of Turbo Decoding

Decoding of PCTC corresponds to iteratively exchanging extrinsic LLRs between two SISO decoders [175]. SISO decoding is usually performed using the BCJR algorithm (see Section 5.1.1). The decoding procedure is shown in Figure 5.12 and corresponds to repeatedly performing half-iterations. In the first half-iteration, the first BCJR unit (denoted by BCJR 1) computes extrinsic LLRs $L^{\mathrm{E1}}(x_k)$ based on the a priori LLRs of the systematic bits $L^{\mathrm{A}}(x_k)$, the parity bits resulting from the first RCC $L^{\mathrm{A}}(c_{k,1})$, and a de-interleaved version of the extrinsic LLRs generated by the second decoder (in the previous half iteration) denoted by $L^{\mathrm{E2}}(x_{\pi^{-1}(k)})$. The function $k = \pi(k')$ performs a bijective mapping between k' and k according to the used interleaver. Note that in the first iteration $L^{\mathrm{E2}}(x_k) = 0$ $(\forall k)$. The branch metrics of the first max-log BCJR decoder (5.13) correspond to [175]

$$\gamma_k(s', s) = \frac{1}{2}\Big(x_k\big(L^{\mathrm{A}}(x_k) + L^{\mathrm{E2}}(x_{\pi^{-1}(k)})\big) + c_{k,1}L^{\mathrm{A}}(c_{k,1})\Big).$$

During the next-half iteration, the second SISO decoder (i.e., BCJR 2) computes new extrinsic LLRs $L^{\mathrm{E2}}(x_k)$, based on $L^{\mathrm{A}}(c_{k,2})$ and on interleaved versions of the systematic bits $L^{\mathrm{A}}(x_{\pi(k)})$ and extrinsic LLRs

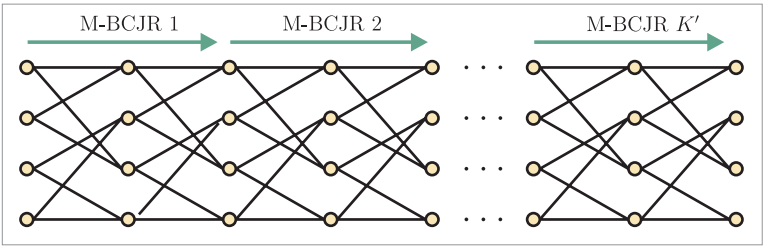


Figure 5.13: Example of high-throughput convolutional decoding using K' M-BCJR decoders operating in parallel.

computed by BCJR 1, i.e., $L^{\mathrm{E1}}(x_{\pi(k)})$. Hence, the branch metrics of the second BCJR correspond to

$$\gamma_k(s', s) = \frac{1}{2}\Big(x_k\big(L^{\mathrm{A}}(x_{\pi(k)}) + L^{\mathrm{E1}}(x_{\pi(k)})\big) + c_{k,1}L^{\mathrm{A}}(c_{k,2})\Big).$$

The turbo decoder now continues to perform iterations between the first and second SISO decoder until a maximum number of I_{PCTC} iterations (corresponding to $2I_{\mathrm{PCTC}}$ half-iterations) has been reached. During the last half-iteration, the active component decoder computes *intrinsic* LLRs instead of extrinsic ones, which are used to obtain estimates of the transmitted bits.[10]

Parallel Trellis Decoding

The decoding throughput of turbo decoders depends on the maximum number of iterations I_{PCTC} as well as the time that is required to perform one half-iteration. Since I_{PCTC} determines the error-rate performance of turbo decoding, the only possibility to increase the throughput is to reduce the decoding time per half-iteration. One approach is to unroll the loop in Figure 5.12, which, unfortunately, requires multiple copies of the a priori LLRs and the extrinsic memories, which quickly becomes inefficient in practice.

A promising alternative is to replicate the M-BCJR decoders and to perform parallel decoding on the trellis. Figure 5.13 illustrates this

[10] If the decoding process is stopped during an even half-iteration, the intrinsic LLRs need to be de-interleaved first.

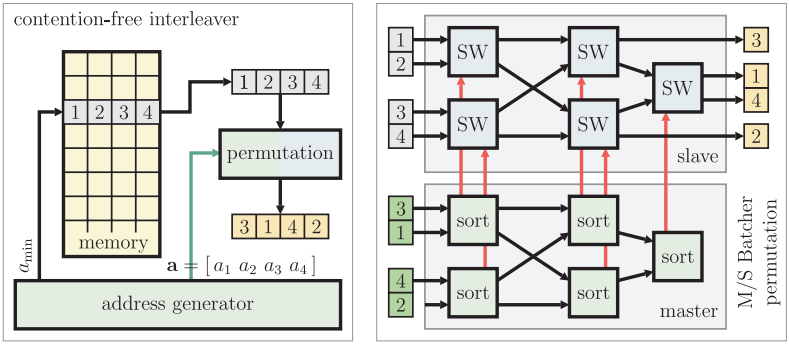


Figure 5.14: Left: generalized hardware structure of contention-free interleavers. Right: implementation of the interleaver based on a master/slave (M/S) Batcher network (with $K' = 4$).

approach using K' parallel M-BCJR decoder cores. Note that parallel decoding leads to an almost K'-fold increase in terms of throughput.[11] The key problem of this approach is the increased memory bandwidth, which gets further aggravated due to the presence of an interleaver. This bottleneck is inherently present for most of the available interleaver-types (e.g., the interleaver used in the HSDPA standard [139]) and hence, inhibits to achieve high throughput for most turbo decoders.

Contention-Free Interleaving from a Hardware Perspective

The interleaver bottleneck can be alleviated by choosing an appropriate interleaver rule and by designing a corresponding interleaver architecture. Contention-free interleavers are able to alleviate the interleaver bottleneck [178]. In the following discussion, we describe a general structure for contention-free interleavers; the corresponding structure is depicted in Figure 5.14.

The interleaver memory of a contention-free interleaver with block-length N consists of $N' = N/K'$ words (i.e., K' and N' must be an

[11] The latency of M-BCJR architectures inhibits linear (in the number of instances K') scaling of the decoding throughput.

integer factor of the block-length N), each of it containing K' LLR-values (or bits). The interleaver memory is initialized column-wise. An address generator computes the K' trellis positions that need to be accessed concurrently according to $a_i = \pi(x_i)$, $i = 1, \ldots, K'$, where x_i corresponds to the current non-interleaved trellis step of the ith decoder, i.e., the first M-BCJR decoder reads from trellis position a_1, the second from a_2 etc., and $\pi(\cdot)$ denotes the interleaving operation. The key idea of contention-free interleavers is the following: If each M-BCJR unit can be separated by N' trellis steps, i.e., $x_i = x_j + N'k$ ($k \in \mathbb{Z}$), all required soft-values to be accessed reside on the same row in the $N' \times K'$-dimensional interleaver memory (which implies that all interleaved addresses satisfy $a_i = a_j + N'k$ with $k \in \mathbb{Z}$). Note, however, the K' values on each row are not necessarily in the right order.

Interleaving can now performed in two steps (see Figure 5.14). The first step corresponds to selecting the word in the memory, which is associated to the smallest address, i.e., $a_{\min} = \min_{i=1,\ldots,K'}\{a_i\}$. The second step corresponds to extracting the assignment of memory contents to the BCJR decoders. This assignment is obtained through all generated addresses $\mathbf{a} = [\, a_1 \ \cdots \ a_{K'} \,]$, i.e., by sorting the address vector $\mathbf{a}$ and by storing the corresponding permutation in a vector $\mathbf{p}$. Now, all the K' values at address $a_{\min}$ simply need to be permuted with $\mathbf{p}$. For example, if the ith M-BCJR accesses the lowest address $a_i = a_{\min}$ and unit j accesses the second lowest, unit i needs to access the first of the K' words and unit j the second one. If the memory content needs to be accessed in a non-interleaved fashion, the word addresses simply correspond to $a_i = x_i$ ($\forall i$) and no permutation is performed, i.e., $\mathbf{p} = [\, 1 \ \cdots \ K' \,]$. We emphasize that this two-step interleaving process allows us to read out K' (interleaved) values at once, without leading to any contentions in the interleaver memory.

Quadratic Polynomial Permutation (QPP) Interleaver

A promising choice for high-throughput turbo decoding are quadratic polynomial permutation (QPP) interleavers [178, 179]. The QPP interleaver rule is defined according to

$$\pi(x) = \left(f_1 x + f_2 x^2 \right) \mod N \tag{5.22}$$

where N denotes the block-length and f_1 and f_2 are suitably chosen interleaver parameters (also depending on N). It has been shown that QPP interleavers exhibit good interleaving properties, e.g., [178,179], and enable (for any two $N', K \in \mathbb{Z}$ such that $N'K' = N$) contention-free access to K' entries in the interleaver's memory. Hence, up to K' parallel M-BCJR decoder instances can be employed, which renders this interleaver well-suited for high-throughput turbo decoding. Turbo codes based on QPP interleaving are, for example, considered in the 3GPP LTE standard [11] and support contention-free memory access for up to $K' = 8$ units for all block-sizes.

In practical applications, it is often desirable to have the interleaved addresses $\pi(x)$ in a certain well-defined order, such as, e.g., for $x = 0, 1, \ldots, N-1$. As it has been noted in [180], address generation for QPP interleaving according to (5.22) can efficiently be performed in recursive manner, i.e.,

$$\begin{aligned} \pi(x+1) &= \big(\pi(x) + \delta(x)\big) \mod N \\ \delta(x+1) &= \big(\delta(x) + b\big) \mod N \end{aligned}$$

with $\pi(0) = 0$, $\delta(0) = f_1 + f_2$, and $b = 2f_2$. Note that this recursion can efficiently been performed in hardware as it only requires additions and modulo operations; the latter can be carried out with simple compare-select circuits. In addition, it is possible to write a recursion for other increments (i.e., are smaller than N and positive), which essentially leads to different $\delta(0)$ and b values.

5.3.2 VLSI Architecture

The high-throughput turbo decoder architecture for the 3GPP LTE standard [11] presented below, bases on the low-power architecture for HSDPA described in [36]. In order to achieve high-throughput, the max-log M-BCJR architecture described in Section 5.1.2 has been optimized for throughput and eight parallel instances are used. In addition, a high-throughput architecture for contention-free interleavers has been designed in order to keep the critical path within the M-BCJR units.

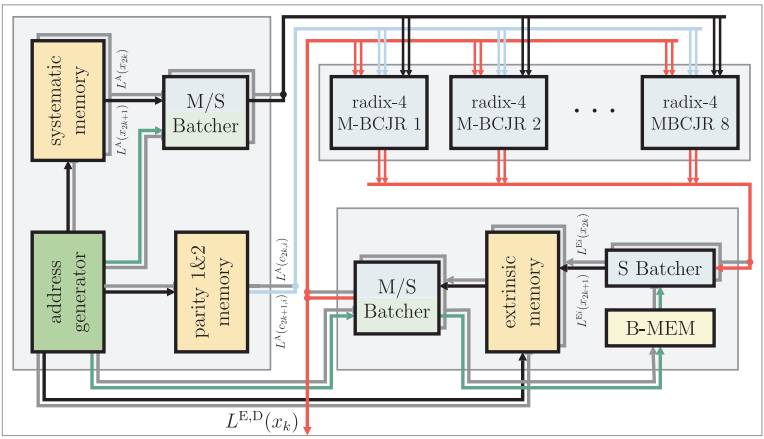


Figure 5.15: High-throughput turbo decoder architecture for 3GPP LTE employing eight radix-4 max-log M-BCJR decoders.

Turbo Decoder Architecture Overview

The turbo decoder architecture is shown in Figure 5.15 and consists of a systematic LLR memory followed by an interleaver, a memory containing the LLRs associated with the parity bits 1 and 2, and an extrinsic LLR memory, which is able to perform (de-)interleaving at the input as well as at the output. Each half-iteration is being decoded in a parallel fashion using eight throughput-optimized M-BCJR instances.

Radix-4 M-BCJR Architecture

Prominent approaches to increase the throughput of trellis-based detectors, such as Viterbi decoders or BCJR algorithms, are pipeline-interleaving of the ACS unit, e.g., [14, 47], or employing higher-radix ACS units [181]. Pipeline interleaving can be used to decode two or more separate codes in a single unit by the insertion of pipeline registers into the ACS unit. It was shown in [47] that pipeline-interleaving is a solution for high-throughput implementation on FPGAs [47]. However, pipeline-interleaving tends to very high clock frequencies

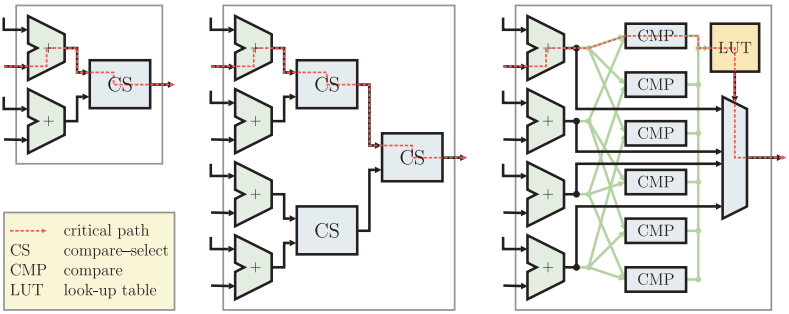


Figure 5.16: Left: radix-2 ACS. Middle: slow radix-4 ACS. Right: fast radix-4 ACS. The critical paths are indicated with dashed lines.

when implemented on ASICs, which renders it difficult to keep the critical path within the ACS units. In contrary, higher-order radix ACS units perform multiple trellis steps per clock cycle, e.g., two trellis steps in a radix-4 architecture, which potentially lower the clock frequency, while achieving approximately twice the throughput compared to a radix-2 architecture. Hence, employing radix-4 ACS units for the high-throughput turbo decoder seems to be advantageous.

Figure 5.3 shows three different ACS architectures. The radix-2 architecture performs one trellis step per clock cycle and requires two adders and one compare-select (CS) unit. A straightforward (slow) radix-4 implementation consists of four adders and three CS units. Compared to the radix-2 ACS, the critical path is enlarged by one CS unit ultimately resulting in a lower clock frequency. Comparison between all adder outputs using parallel comparison units and employing an evaluation logic in order to determine the maximum element of all four inputs only slightly increases the critical path compared to a radix-2 ACS implementation. The fast radix-4 ACS leads —thanks to performing two trellis steps per clock cycle— to a much higher throughput (see, e.g., [47]). Figure 5.17 compares all three ACS variants using the β-unit of a 8-state BCJR decoder (see Section 5.1.2), i.e., including eight ACS units and the required LLR-to-branch-metric computation logic. Note that the radix-4 implementation achieves higher throughput while being less efficient in terms of area divided

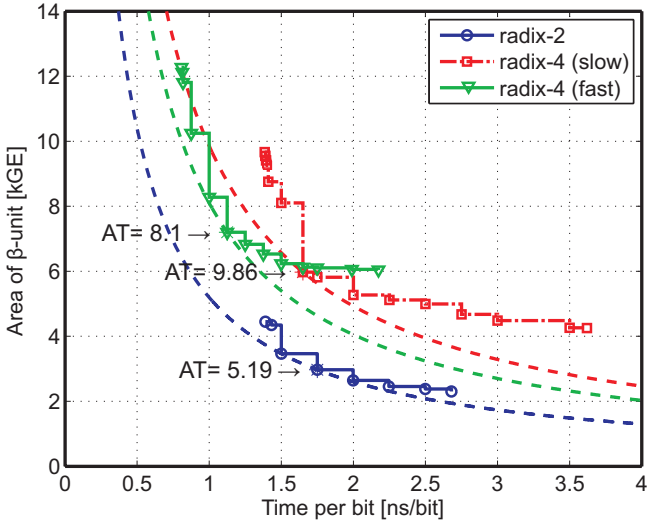


Figure 5.17: Area-time tradeoff comparison of radix-2 and radix-4 β-unit ACS in 180 nm CMOS technology. The dashed lines represent constant area-time (AT)-product.

by the time per bit (approximately 56% for the fast and 90% for the slow implementation). With high-throughput in mind, the slightly degraded hardware-efficiency was found to be acceptable.

Table 5.3 compares implementation results of a high-throughput 8-state radix-4 max-log M-BCJR and the radix-2 implementation described in Section 5.1.3. Note that the maximum achievable clock frequency of the radix-4 implementation is only 7.1% lower compared to that of the radix-2 counterpart, which ultimately leads to a 86% higher throughput. Even though the radix-4 ACS requires one bit more than a radix-2 ACS (since two branch metrics are being added for radix-4), the α-memory is considerably smaller as only the forward state-metrics for every second trellis step need to be stored. In summary, the radix-4 max-log M-BCJR implementation is approximately 20% less efficient (in terms of time per bit multiplied by circuit area) compared to that of the radix-2 M-BCJR implementation.

QPP Interleaver Architecture

Since eight M-BCJR instances are used, data for eight M-BCJR decoders need to be read out at once from the interleaver memory per clock cycle. Due to the presence of a radix-4 M-BCJR implementation (which performs two trellis steps per clock cycle) each M-BCJR decoder additionally requires data for the even and the odd trellis steps. Hence, 16 interleaved values need to be read from the systematic and extrinsic memories per clock cycle.

To this end, an address generator computes all 16 interleaved addresses per clock cycle using the recursive formulation outlined in Section 5.3.1. Then, all eight addresses that belong to the even trellis steps shown for $K' = 4$ in Figure 5.14.[12] are being fed into a Batcher sorting network [182]. Since the smallest of all eight (interleaved) addresses does not exceed N', it can directly be used as the word address $a_{\min}$ of the interleaver's memory. Sorting and permutation, as described in Section 5.3.1, is performed within a unit referred to as master/slave (M/S) Batcher network (see Figure 5.14). The master Batcher network mainly performs sorting of the input addresses (in ascending order) with the aid of two-input sort units (denoted as "sort" in Figure 5.14). These two-input sorting units additionally produce a control signal whether the inputs have been swapped or not. Using these control signals, the sort-order computed in the master network can now be applied to the LLR-values within the slave network, which only consists of two-input switches (denoted by "SW"). We emphasize that this architecture enables concurrent and hardware-efficient sorting of all K' inputs and enables pipelining, which is essential for a high-throughput implementation of contention free interleavers. For more details on the architecture, we refer to [183].

5.3.3 Implementation Results

The high-throughput turbo decoder for 3GPP LTE described above has been implemented in 130 nm (1P/8M) CMOS technology. The ASIC layout is shown in Figure 5.18.

[12] The same procedure has to be performed for the odd trellis steps.

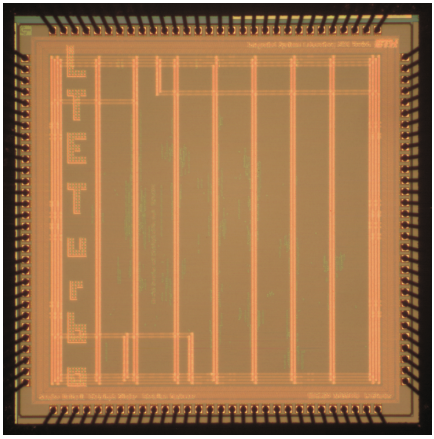


Figure 5.18: ASIC micrograph of the high-throughput LTE turbo decoder in 130 nm (1P/8M) CMOS technology.

Area Breakdown

Table 5.11 provides a detailed area breakdown of the turbo decoder implementation. The total circuit area corresponds to 3.57 mm^2, which corresponds to 697 kGE (including all memories). More than 65% of area is occupied by the eight radix-4 max-log M-BCJR units. Note that the M-BCJR units have not been pushed to their performance limits (in terms of the critical path) and hence, are slightly smaller than the implementation shown in Table 5.10. The interleaver (consisting of memories, address generators, and the M/S batcher network) only requires 24%. Hence, we conclude that the circuit area and the critical path of the turbo decoder implementation is dominated by the area and the critical path of the M-BCJR units, respectively Hence, it is essential to optimize the max-log M-BCJR units in order to attain a hardware-efficient turbo decoder implementation.

Implementation Results and Comparison

Table 5.12 provides post-layout implementation results of the 3GPP-LTE turbo decoder and compares its performance to two reference

Table 5.10: Radix-2 max-log M-BCJR compared to a radix-4 implementation (8-states, $M = 32$) in 180 nm CMOS technology.

Radix	2	4
Cell area [kGE]	37.3	80.7
Core area [mm^2]	0.36	0.76
Max. clock frequency [MHz]	375	350
Throughput [Mbps]	375	700
ACS word-width [bit]	9	10
α-memory [kbit]	2.25	1.25
Efficiency [kGE/Mbps]	0.10	0.12

Table 5.11: Detailed area breakdown of the 3GPP LTE turbo decoder ASIC in 130 nm (1P/9M) CMOS technology.

Unit	mm^2	kGE[a]	%
M-BCJR 1 to 8	8×0.29	8×57.3	65.7
– α-unit	0.037	7.23	
– β-unit	0.042	8.15	
– β'-unit	0.037	7.22	
– LCU	0.080	15.6	
– α-memory	0.045	8.8	
– miscellaneous[b]	0.049	10.3	
Interleaver	0.87	170	24.3
– extrinsic memory	0.28	55.4	
– systematic memory	0.14	27.1	
– address generator	0.32	63.4	
– M/S batcher network	0.03	6.17	
– miscellaneous[c]	0.10	17.9	
Parity memory	0.31	61.5	8.8
Miscellaneous	0.07	7.1	1.2
Total	3.57	697	100

[a] One GE corresponds to a two-input drive-one NAND gate of size 5.12 µm^2.
[b] Contains γ-memories, the FSM, and other top-level logic.
[c] Contains the FSM and top-level logic (e.g., buffers and pipeline registers).

Table 5.12: Post-layout implementation results of the LTE turbo decoder and comparison with turbo decoders for HSDPA.

	This work	Benkeser *et al.* [36]	Bickerstaff *et al.* [37]
Standard	LTE	HSDPA	HSDPA
Technology [nm]	130	130	180
Radix/parallelism	4/8	2/1	4/1
Core area [mm^2]	3.57	1.2	14.5
Total memory [kbit]	129.1	120	450
Cell area [kGE]	553[a]	44.1	410
Max. clock freq. [MHz]	400	246	145
Max. throughput[b] [Mbps]	520 (5.5)	20.2 (5.5)	33.4[c] (6)
Efficiency[d] [kGE/Mbps/it]	0.19	0.40	2.05

[a] Corresponding to the area without systematic, parity, and extrinsic memories.
[b] The number within the braces correspond to the number of full iterations.
[c] The throughput has been scaled by 1.39 to consider technology scaling from 180 nm to 130 nm CMOS technology.
[d] The hardware-efficiency is normalized to the number of iterations (it).

implementations for the HSDPA standard, namely the low-power implementation by Benkeser *et al.* [36] and the high-throughput radix-4 implementation of Bickerstaff *et al.* [37].

The high-throughput 3GPP LTE implementation presented in this section requires 126 kbit of memory[13] and an active core area of 3.57 mm^2. The throughput of the decoder is approximately[14]

$$\Theta \approx \frac{2N}{\left(3M' + N/K'\right) 2 I_{\mathrm{PCTC}}} f_{\mathrm{clk}}$$

million bits per second (Mbps), where $K' = 8$ corresponds to the number of M-BCJR units and $M' = 15$ denotes the number of cycles required per window. Note that the maximum block-length of the

[13] This figure excludes the memories required in the M-BCJR units, since they have been realized with latch-arrays instead of using SRAM macro-cells
[14] The exact number of clock cycles per half-iteration slightly depends on the block-length N, which has been neglected in throughput computation.

3GPP LTE standard corresponds to $N = 6144$, which leads to a maximum throughput of 520 Mbps if using 5.5 iterations and running it at the maximum clock frequency of 400 MHz. From Table 5.12 one can observe that the LTE decoder attains a 25 times higher throughput than that of the reference HSDPA implementation [36] and is twice as efficient in terms of kGE per Mbps per iteration. We emphasize that such a high throughput and good hardware-efficiency can only be achieved by alleviating the interleaver bottleneck using the master-slave Batcher network, in combination with eight highly-optimized radix-4 max-log M-BCJR units.

5.4 Performance/Complexity Tradeoff for SISO Channel Decoding

The presented SISO channel decoders exhibit either excellent error-correction capability or low implementation complexity. It is therefore essential to compare both, the (error-rate) performance *and* the complexity in order to identify which channel code/decoder-pairs exhibit a better tradeoff.

In this section, we compare the performance/complexity tradeoff associated with the SISO channel decoders described in this chapter. The tradeoff comparison below bases on VLSI implementation results and only considers their error-correcting capabilities. The soft-input soft-output performance is characterized in Section 6 jointly with the MIMO detector.

5.4.1 Performance and Complexity Measures

In order to enable a fair comparison in terms of performance and hardware-complexity, several assumptions have been made. For all considered SISO channel decoders, a similar code-block length of approximately 1000 information bits is used. Note that the exact number of information bits depends on the code and the required tail-bits. All codes use rate $R = 1/2$ (i.e., resulting in approximately 2000 coded bits). In order to account for technology scaling [184, 185], all VLSI implementation results are normalized to 180 nm CMOS technology,

i.e., we scale the throughput of the 130 nm CMOS implementations by a factor of 130/180.

In the following, all assumptions and modifications for each considered SISO channel decoder are listed in detail.

- The 4-, 8-, 16-, 32-, and 64-state radix-2 max-log M-BCJR implementations described in Section 5.1 are used. In order to consider the interleaver and the associated LLR memory for 1952 coded bits (including tail bits), we add the circuit area of two single-port macro-cell memories with storage capacity for 976 LLRs (each consisting of 5 bit) requiring a total area of 16 kGE. In addition, we include 3 kGE, which corresponds to the area of two QPP-interleaver address generators (see Section 5.3.1).

- For the 8-state radix-4 max-log M-BCJR decoder described in Section 5.3.2, we note that twice the memory bandwidth (compared to a radix-2 implementation) is required. Hence, we consider four QPP address generators (6 kGE) and four memories (each consisting of 488×5 bit) of 26 kGE total area.

- As a reference, we consider the *hard-output* Viterbi decoder described in [16], which has been implemented in 130 nm CMOS technology. This implementation requires 136 kGE and consists of two parallel radix-4 Viterbi cores, each achieving 320 Mbps. The 180 nm-equivalent throughput corresponds to 462 Mbps. As it has been done for the radix-4 M-BCJR, we include the area of four QPP address generators and four memories.

- The QC-LDPC decoder implementation described in Section 5.2 is used in the comparison. Since the decoder is already designed for 1944 coded bits no special assumptions are made. We use the IEEE 802.11n QC-LDPC code with rate 1/2 and $Z = 81$ (see Table 5.7).

- The memory area of the high-throughput 3GPP LTE turbo decoder in Section 5.3 is reduced to account for a block length of 976 information bits (resulting in 1964 coded and tail bits). To this end, we subtract the area of systematic, parity, and extrinsic memories (equivalent to 144 kGE) the decoder's total area and replace it with the area corresponding to memories (with

equivalent functionality) for the target block-length of 1964 bits (total area: 51.5 kGE). Since the implementation is for 130 nm CMOS technology, the throughput is reduced by 130/180 to account for technology scaling. In order to attain a code rate of 1/2, the coded bit stream is punctured appropriately.

Complexity Measure Complexity is measured in terms of kGE per throughput (in million bits per second), which indicates the hardware resources (in Mbps) spent to achieve a certain throughput. This measure corresponds to the area-time (AT) product in [14].

Performance Measure Performance is characterized by the SNR operating point, i.e., the minimum SNR required to achieve a target bit error rate (BER) of 10^{-5}. The error-rate has been simulated in a single-input single-output OFDM system ($M_{\mathrm{T}} = M_{\mathrm{R}} = 1$) with an i.i.d. (across tones) Rayleigh fading channel model using BPSK modulation. The decoders are assumed to perform all computations in floating-point, i.e., fixed-point precision effects have been neglected in this comparison. Note that approximations on algorithmic level (such as, e.g., the max-log approximation or OMS-MP) have been employed.

5.4.2 Tradeoff Comparison and Conclusions

Figure 5.19 summarizes the results of this chapter by showing the performance/VLSI implementation complexity tradeoff for the radix-2 and radix-4 M-BCJR decoders, the Viterbi decoder, the QC-LDPC decoder, and the LTE turbo decoder. We can draw the following conclusions:

- In the low-complexity (i.e., hardware-efficient region), the M-BCJR decoders with small constraint length (i.e., $K = 3, 4, 5$) are Pareto-optimal in terms of the performance/complexity tradeoff. However, the M-BCJR algorithm scales badly with the constraint-length and hence, to achieve a low SNR operating point quickly leads to an inefficient decoding architecture. For example, the the 64-state ($k = 7$) M-BCJR implementation

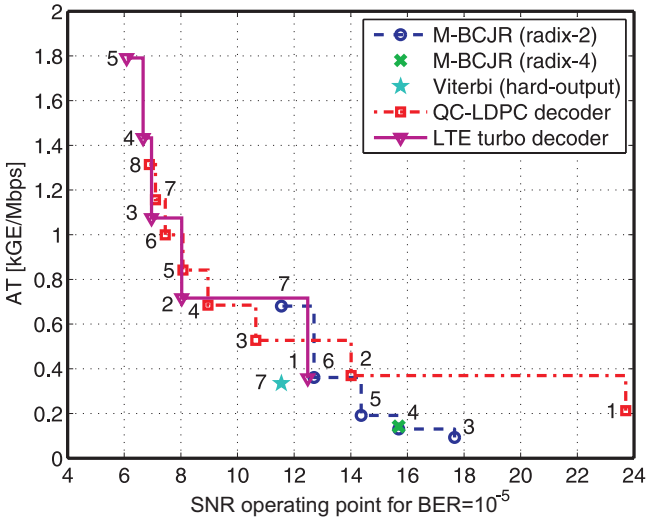


Figure 5.19: Hardware-based performance/complexity tradeoff for SISO channel decoders. The numbers next to the M-BCJR decoder corresponds to the constraint length. The numbers next to the QC-LDPC and LTE turbo decoder denote the number of iterations.

is approximately twice less efficient than its 32-state counterpart, while the improvement in terms of SNR operating point is rather low.

- The 8-state radix-4 M-BCJR detector attains the same performance as its radix-2 counterpart, but is slightly less efficient (due to the less efficient ACS units and the increased memory bandwidth, see Table 5.10).

- The reference hard-output Viterbi detector of [16] achieves the same performance as the 64-state M-BCJR decoder, but is approximately two times less complex. Hence, if no SISO-decoding capability is required, implementation of the Viterbi algorithm for detection of convolutional codes is preferable.

- In the medium-to-low-SNR regime, the QC-LDPC decoder and

the LTE turbo decoder attain a similar performance/complexity tradeoff. Thus, no clear advantage for either of the two decoders can be identified. Note that important advantages of the QC-LDPC decoder implementation (which can not be seen from Figure 5.19) are the fact that it offers high degree of *flexibility* (e.g., in terms of reconfigurability for various block-lengths and codes), covers a large tradeoff region by adjusting the number of iterations, and its architecture offers scalability for even higher throughput.

- At very-low SNR, the LTE turbo decoder slightly outperforms the QC-LDPC decoder. Thus, if a system needs to operate reliably at very low-SNRs, the LTE turbo decoder seems to be the best choice (among the considered codes and decoders). Note, however, that the performance gap to the QC-LDPC decoder is very small (and the proposed turbo decoder architecture is less flexible than the QC-LDPC decoder).

In summary, Viterbi decoding is well-suited for low-complexity hard-output decoding of CCs. If SISO-capability is required, the M-BCJR algorithm is suitable for decoding of CCs having small constraint length. QC-LDPC codes and turbo codes seem to be well-suited for medium to low SNR operating points, where QC-LDPC decoders can, in general, be designed to achieve a significantly higher degree of flexibility than turbo decoders. We therefore believe that the flexibility and scaling advantages render QC-LDPC codes to be the most promising scheme for future high-performance (wireless) communication standards.

Chapter 6

Performance/Complexity Tradeoffs

The goal of this chapter is to analyze the performance and VLSI implementation complexity of iterative MIMO decoding. To this end, we consider the performance/complexity tradeoffs associated with (soft-input) soft-output MIMO detection in combination with SISO channel decoding. The tradeoff analysis bases on numerical (error-rate) performance simulations and on the VLSI implementation results provided in Chapters 3–5.

We start in Section 6.1 by introducing the performance and complexity measures. In Section 6.2, we show corresponding performance/complexity tradeoff results and discuss our findings.

6.1 Performance and Complexity Measures

In order to enable a fair comparison in terms of performance and complexity, the choice of the right performance measures is of paramount importance. Moreover, assumptions on the system model that accurately reflect real-world aspects need to be made. In the remainder of this section, we describe the MIMO system model and detail the assumptions that have been used in the subsequent tradeoff analysis.

6.1.1 Performance Measure

The performance measure employed in the remainder of this chapter corresponds to the SNR operating point (cf. Section 4.6.3), i.e., the minimum required SNR to achieve a target FER of 1%.

These SNR operating points are simulated in a coded MIMO-OFDM system that is similar to the 40 MHz bandwidth mode of the IEEE 802.11n standard [2]. We use $M_{\mathrm{T}} = M_{\mathrm{R}} = 4$, 16-QAM symbol constellation with Gray labeling as defined in [2], 122 OFDM tones, and assume a TGn type C channel model [12]. All codes used in the ensuing discussion have rate 1/2. One frame consists of approximately 1000 information bits, which leads to about twice the number of coded bits for each (spatial) OFDM symbol.[1] All (error-rate) performance simulations are averaged over 640'000 channel realizations and for each channel realization a single noise realization has been generated. Since fixed-point simulations of iterative MIMO decoding are time-consuming and all implementations show a small implementation loss, the simulations results base on floating-point arithmetic (i.e., fixed-point effects have been neglected). Note, however, that all approximations performed on algorithmic level (such as the max-log approximation) have been considered in our simulations.

6.1.2 Throughput and Area Measures

Throughput is given in million *information* bits per second (Mbps). In order to account for the different process technologies, all throughput figures have been normalized to 90 nm CMOS technology using technology scaling rules. In particular, we assume that the throughout scales linearly with the maximum clock frequency of the circuit, where the maximum clock frequency scales approximately inversely proportional with the process technology [184, 185].[2] Hence, the throughput for 90 nm CMOS technology (denoted by Θ_{90}) is assumed to be $\Theta_{90} = \frac{s}{90}\Theta_s$ where s stands for the technology node (in nm) of the implementation to be scaled (with corresponding throughput Θ_s).

[1] The exact number of information/code bits depends on the employed code.

[2] Note that this assumption was shown to be valid for CMOS technologies down to 90 nm [186]. In addition, based on implementation results of various MIMO circuits, it was demonstrated in [187] that technology scaling is surprisingly accurate for CMOS technologies ranging from 250 nm to 130 nm.

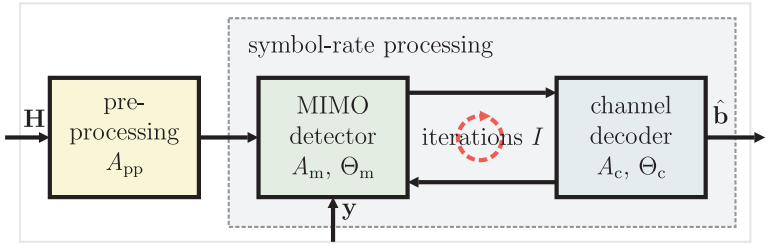


Figure 6.1: Iterative MIMO decoder with area and throughput figures.

A natural throughput measure for SISO detectors for iterative MIMO systems and SISO channel decoders is million LLR-values per second. Since $R = 1/2$ is considered in the remainder of this chapter, two LLR values are generated per information bit. Hence, we divide the throughput (in terms of LLRs per second) by two and provide the information rate in terms of Mbps instead.

Circuit Area is given in terms of gate equivalents (GEs), where one GE refers to a two-input drive-one NAND gate in the corresponding technology.

6.1.3 Efficiency and Complexity Measures

An iterative MIMO decoder (see Figure 6.1) consists, in general, of a preprocessing unit, a SISO detector for MIMO systems, and a SISO channel decoder. The corresponding hardware-efficiency is denoted by AT (where A and T refer to the circuit area and the time per data item, respectively), which corresponds to the size-time product [14]. This measure indicates the amount of resources (in terms of circuit area) that are required to attain a given throughput.

In the following, we consider an iterative MIMO decoding schedule, where the soft-input soft-output MIMO detector and the SISO channel decoder are *always* active. This can, for example, be achieved by processing two OFDM symbols concurrently and in an interleaved fashion, i.e., the MIMO detector performs detection of even OFDM symbols, while the channel decoder operates with odd OFDM sym-

bols. After completion, both units exchange their data. Note that the slower unit determines the time instant where data is exchanged. We emphasize that the considered schedule is rather optimistic since, for example, latency constraints (present in practical systems) can lead to the case where not enough data can be delivered to the decoder in a given time interval. Nevertheless, we compute the hardware-efficiency associated to this schedule and the system depicted in Figure 6.1 according to

$$AT \triangleq \frac{A_{\mathrm{pp}} + (A_{\mathrm{m}} + A_{\mathrm{c}})I}{\min\left\{\Theta_{\mathrm{m}}, \Theta_{\mathrm{c}}\right\}} \tag{6.1}$$

where the subscripts $_{\mathrm{pp}}$, $_{\mathrm{m}}$, and $_{\mathrm{c}}$, refer to preprocessing, to the soft-input soft-output MIMO decoder, and to the SISO channel decoder, respectively. The definition in (6.1) results from the fact that the smaller throughput limits the overall system throughput and the larger the number of iterations I is, the more circuit area is required (for MIMO detection and channel decoding) in order to achieve a certain throughput. Note that in (6.1), it has been assumed that the preprocessing unit is neither throughput-critical nor affected by the number of iterations.[3]

Throughput Matching

Since we are comparing VLSI implementation results from different CMOS processes, the technology-scaled throughput figures of the MIMO detector and the channel decoder do not necessarily match. This mismatch evidently leads to poor overall hardware-efficiency, when considering the minimum of Θ_{m} and Θ_{c} in (6.1). In practice, however, the two units are designed such that both meet the target throughput of the MIMO decoder; this can —up to a certain extent— be achieved by application of architectural transformations. Hence, we assume that the throughput of the MIMO detector and the channel decoder are both matched to the *target throughput* Θ, by the use of architectural transformations that yield constant AT-product (e.g.,

[3]The latency associated with preprocessing is critical in many wireless communication standards, such as in, e.g., IEEE 802.11n [2]. However, latency aspects are not considered in this thesis for the sake of simplicity.

using replication or time sharing [14]). In particular, we derive area and throughput figures associated with new (hypothetical) architectures (denoted by the superscript $'$) according to

$$A'_{\mathrm{m}} \triangleq A_{\mathrm{m}} \frac{\Theta}{\Theta_{\mathrm{m}}} I, \quad \Theta'_{\mathrm{m}} \triangleq \Theta \tag{6.2}$$

$$A'_{\mathrm{c}} \triangleq A_{\mathrm{c}} \frac{\Theta}{\Theta_{\mathrm{c}}} I, \quad \Theta'_{\mathrm{c}} \triangleq \Theta. \tag{6.3}$$

which preserve the hardware-efficiency, i.e., $AT'_{\mathrm{m}} = AT_{\mathrm{m}}$ and $AT'_{\mathrm{c}} = AT_{\mathrm{c}}$, where $AT_{\mathrm{m}} = A_{\mathrm{m}}/\Theta_{\mathrm{m}}$ and $AT_{\mathrm{c}} = A_{\mathrm{c}}/\Theta_{\mathrm{c}}$ correspond to the hardware-efficiencies of the initial MIMO detector and channel decoder, respectively. With (6.2) and (6.2), we can now compute the hardware-efficiency of the throughput-matched iterative MIMO decoder AT' according to (6.1)

$$AT' \triangleq \frac{A_{\mathrm{pp}}}{\Theta} + (AT_{\mathrm{m}} + AT_{\mathrm{c}})I. \tag{6.4}$$

Note that if the target throughput Θ is low, the preprocessing area dominates the overall hardware-efficiency AT', whereas for large values of Θ or a large number of iterations I, preprocessing complexity becomes negligible.

We emphasize that throughput matching and hence, the resulting hardware-efficiency in (6.4) can only be justified for circuits, which allow to trade throughput for circuit area (by means of architectural transformations) over a large region and in fine steps. In the remainder of this thesis, we assume that these properties apply to the considered MIMO detectors and channel decoders circuits.

Silicon Complexity

The hardware-efficiency expression obtained by using throughput matching can be used to *estimate* the total area A' required to implement an iterative MIMO decoder for a given target throughput Θ and number of iterations I. This area-estimate corresponds to

$$A' \triangleq AT'\Theta = A_{\mathrm{pp}} + (AT_{\mathrm{m}} + AT_{\mathrm{c}})I\Theta \tag{6.5}$$

gate equivalents and will be referred to as "silicon complexity" in the remainder of this chapter. We can see from (6.5) that increasing the

number of iterations I leads to a *linear increase* in terms of circuit area. Hence, in order to achieve low silicon complexity, a small number of iterations I is necessary. In the ensuing discussion, we will assume a target throughput of $\Theta = 250\,\text{Mbps}$.[4] Note that memories for storage of channel matrices and of received vectors are not included in these estimates.

Symbol-Rate Efficiency

Some tasks in an iterative MIMO decoder are executed at symbol-rate, i.e., need to be carried out for each received vector and each iteration. Preprocessing comprises those tasks that only need to be computed when the channel state changes. In WLAN systems, the channel state often remains constant over a long period (e.g., as the Doppler spread is small) and hence, preprocessing needs to be executed rarely. Hence, the tradeoff analysis shown in the remainder of this chapter will also compare the hardware-efficiency associated with the symbol-rate tasks, i.e.,

$$AT_\text{s} \triangleq (AT_\text{m} + AT_\text{c})I. \tag{6.6}$$

This measure is referred to as "symbol-rate efficiency" in the sequel. Note that (6.6) corresponds to the size-time product of the symbol-rate tasks of an iterative MIMO decoder performing I iterations and indicates the amount of GEs required to perform all symbol-rate tasks for a certain throughput.

6.2 Performance/Complexity Tradeoffs

This section provides performance/complexity tradeoff results for iterative MIMO decoding based on the MIMO detectors and channel decoders described in Chapters 3 to 5. In Section 6.2.1, we analyze the tradeoffs associated with the SISO MMSE PIC detector and different channel codes. In Section 6.2.2, we compare the tradeoffs realized by the SISO MMSE PIC algorithm and soft-output STS SD.

[4] The throughput achieved in IEEE 802.11n [2] for $M_\text{R} = M_\text{T} = 4$, 16-QAM, rate 1/2, 108 OFDM tones, 40 MHz bandwidth, and 400 ns guard interval (which is similar to the considered system model) corresponds to 240 Mbps.

We emphasize that the results provided in the following correspond to a snapshot for the considered system parameters and the implementations provided in the previous chapters. In addition, the silicon-complexity measure in (6.5) only provides *optimistic estimates* of the true silicon complexity required for iterative MIMO decoding. For real-world MIMO systems, the results might differ. We believe, however, that the results obtained in the following analysis enable to draw conclusions that are valid for real-world MIMO systems.

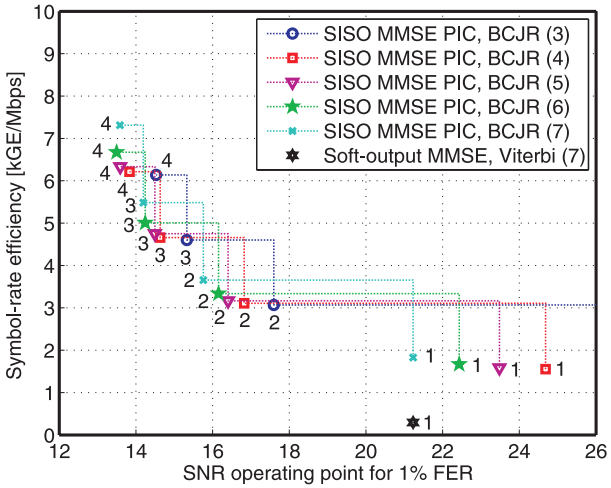
6.2.1 Impact of Channel Codes to Performance/Complexity Tradeoff

Impact of Convolutional Codes

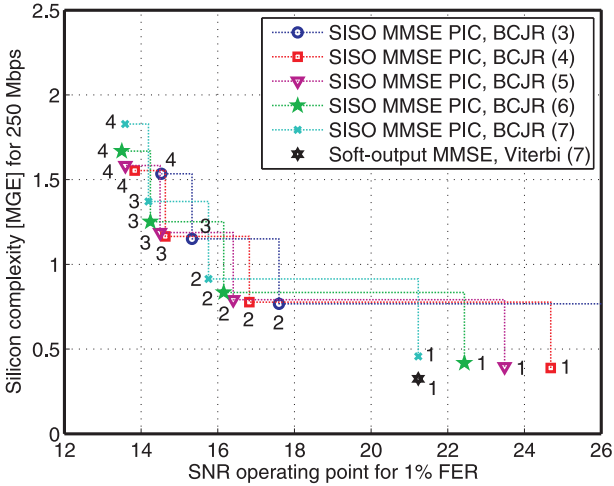
Figure 6.2(a) and Figure 6.2(b) compare the performance/symbol-rate efficiency tradeoff and the performance/silicon complexity tradeoff, respectively. The numbers given in the legend correspond to the constraint length of the CC. The numbers next to the curves correspond to the number of iterations I.

Performance vs. Symbol-Rate Efficiency Figure 6.2(a) shows that for $I = 1$, soft-output MMSE detection in combination with the Viterbi decoder in [16] outperforms SISO MMSE PIC using the $K = 7$ M-BCJR decoder (in terms of symbol-rate efficiency) by a factor of six; this is due to the fact that preprocessing needs to be performed at symbol-rate for the SISO MMSE PIC algorithm, whereas preprocessing for linear SO MMSE detection is not contained in this measure, because it is not a symbol-rate task. We can furthermore see that for $I = 1$, the constraint length of CCs has a strong impact on the SNR operating point of the MIMO system.

By increasing the number of iterations, the SNR operating point of the MIMO system is reduced significantly, i.e., from the first to the second iteration, performance improvements from 6 dB to 9 dB (depending on the constraint-length of the code) can be observed. For $I > 4$, the performance starts to saturate. Hence, using an even larger number of iterations does not seem to be efficient. In addition, for $I > 2$ the impact of the channel code on the performance is small. Hence, smaller constraint-lengths may lead to the same performance



(a) Performance vs. symbol-rate efficiency.



(b) Performance vs. silicon complexity.

Figure 6.2: Performance/complexity tradeoffs for SISO MMSE PIC with M-BCJR decoding compared to linear soft-output MMSE detection with Viterbi decoding.

as more sophisticated CCs, while being more efficient. This implies that CCs with low constraint-length can be beneficial in terms of the symbol-rate efficiency.

Performance/Silicon Complexity Tradeoff Figure 6.2(b) shows that the silicon complexity (in MGE) of linear SO MMSE detection in combination with a Viterbi decoder is similar (i.e., only 1.4 times lower) to that of SISO MMSE PIC with M-BCJR decoding (for $K = 7$). The reason for this is that the preprocessing area given in [16] is included in the area estimate of linear SO MIMO detection.

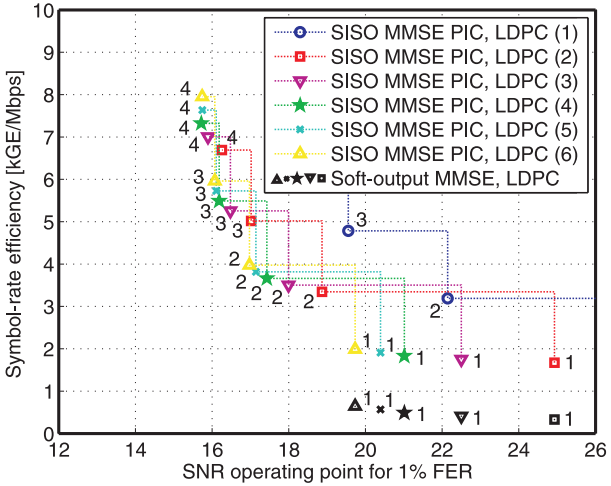
Increasing the number of iterations increases the silicon complexity of iterative MIMO decoding (of SISO MMSE PIC with M-BCJR decoding). Performing two iterations, for example, requires approximately 750 MGE to 950 MGE. We, therefore, conclude that achieving the gains offered by iterative MIMO decoding comes at the cost of (often significantly) increased silicon complexity. Further increasing the number of iterations (i.e., $I > 2$) leads to smaller performance gains.

Impact of LDPC Codes

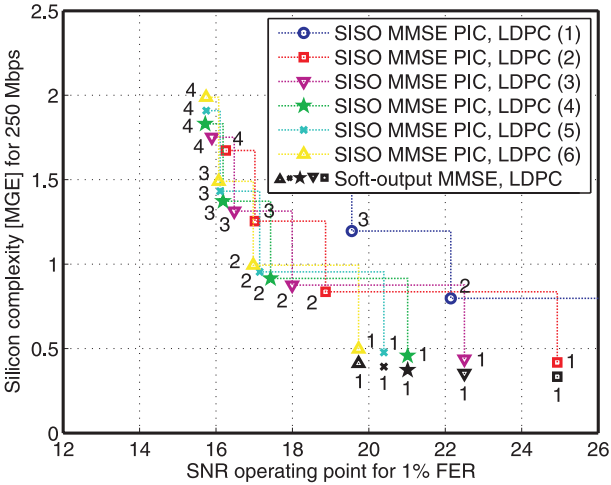
Figure 6.3 illustrates the tradeoffs of iterative MIMO decoding using SISO MMSE PIC with the QC-LDPC decoder described in Section 5.2. The rate-1/2, $Z = 81$ LDPC code from [2] is used and the numbers given in the legend correspond to the number of iterations performed in the LDPC-decoder (i.e., I_{LDPC}).

Performance vs. Symbol-Rate Efficiency From Figure 6.3(a), we see that the performance in the first iteration heavily depends on the number of LDPC-internal iterations I_{LDPC}. Increasing I_{LDPC} above 6 iterations does not substantially improve the performance. By increasing I, the performance improves, but starts to saturate at approximately 15 dB SNR (for $I > 2$). Since, the symbol-rate efficiency substantially degrades for $I > 1$, we conclude that LDPC codes seem to be better suited for soft-output-only MIMO decoding.

Performance/Silicon Complexity Tradeoff Figure 6.3(b) shows that the linear soft-output MMSE detector from [16] outperforms



(a) Performance vs. symbol-rate efficiency.



(b) Performance vs. silicon complexity.

Figure 6.3: Performance/complexity tradeoffs for SISO MMSE PIC and linear soft-output MMSE detection (both with LDPC codes).

the SISO MMSE PIC implementation (using M-BCJR decoding) for $I = 1$ and requires approximately 20% less silicon complexity. Iterative MIMO decoding with $I = 2$ requires approximately 1 MGE. Further increasing I does not significantly improve the performance.

Impact of Turbo Codes

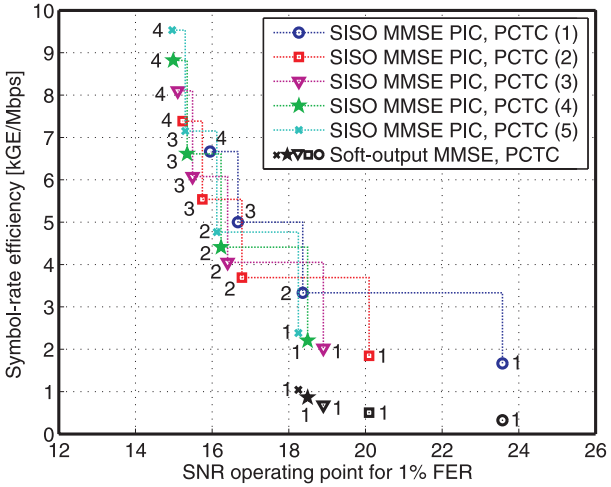
Figure 6.4 shows the tradeoff results using PCTCs. The numbers in the legend correspond to the number of (full) iterations in the turbo-decoder (denoted by I_{PCTC}). We are using the turbo code from the 3GPP LTE standard [11] with 976 information bits per code-block (cf. Section 5.3) and employ puncturing to obtain rate 1/2.

Performance vs. Symbol-Rate Efficiency The tradeoff-curves shown in Figure 6.4(a) are similar to that realized by QC-LDPC codes (see Figure 6.3(a)). Linear soft-output MMSE detection using $I_{\mathrm{PCTC}} = 4$ performs well and approaches 18 dB SNR. Increasing the number of iterations $I > 1$ (and using the SISO MMSE PIC algorithm) quickly degrades the symbol-rate efficiency, while the performance improvements are rather low. Hence, using even a small number of iterations is inefficient in this scenario.

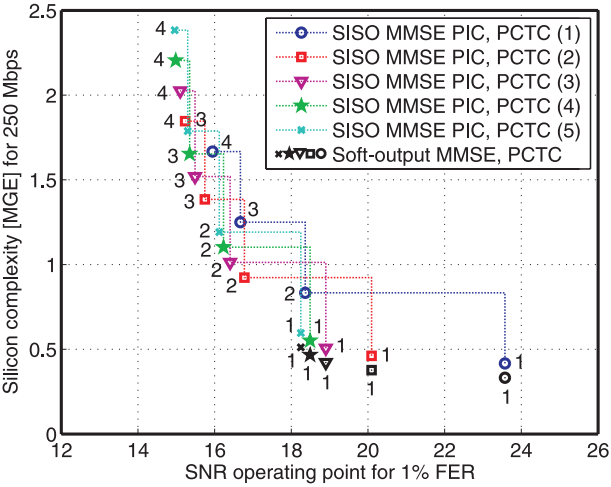
Performance/Silicon Complexity Tradeoff In Figure 6.4(b), we observe that non-iterative MIMO decoding using $I = 1$ based on turbo codes requires approximately 400 kGE to 600 kGE and the performance ranges between 18 dB to 23.5 dB SNR (depending on I_{PCTC}). It seems to be better (in terms of the performance/complexity tradeoff) to use a larger number of iterations within the turbo decoder (i.e., I_{PCTC}) than increasing I. In particular, performing more than one iteration substantially increases the silicon complexity of the iterative MIMO decoder and requires more than 2 MGE for $I = 4$. We therefore conclude that PCTCs are better suited for non-iterative MIMO decoding (e.g., using linear soft-output MMSE detection).

Summary

Figure 6.5 summarizes the tradeoff results realized by the SISO MMSE PIC and soft-output linear MMSE detection in combination with

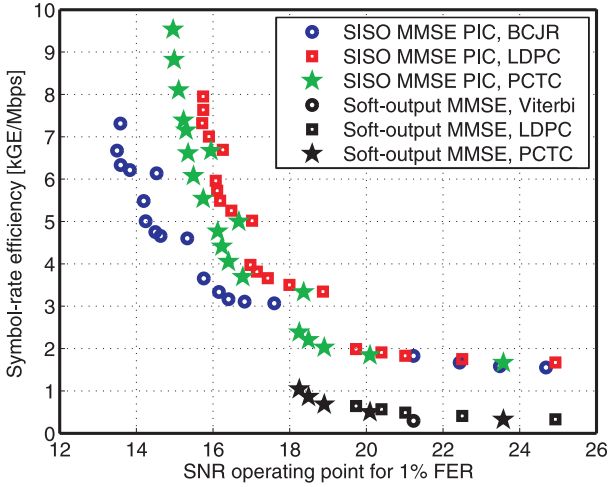


(a) Performance vs. symbol-rate efficiency.

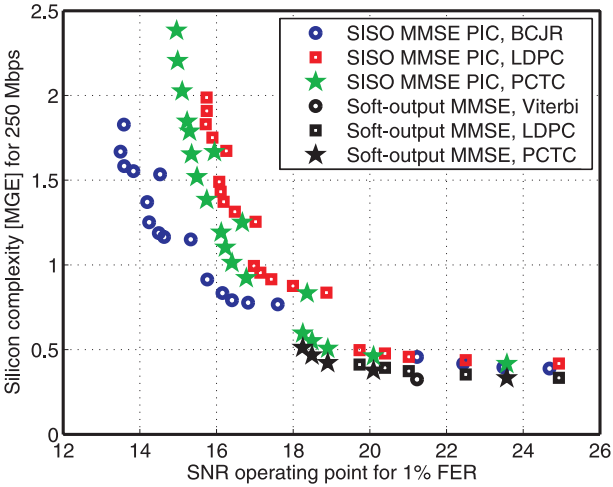


(b) Performance vs. silicon complexity.

Figure 6.4: Performance/complexity trade-offs for SISO MMSE PIC and linear soft-output MMSE detection (both using PCTCs.)



(a) Performance vs. symbol-rate efficiency.



(b) Performance vs. silicon complexity.

Figure 6.5: Tradeoff summary for SISO MMSE PIC and linear soft-output MMSE detection using different channel decoders.

(SISO) decoding of CCs, LDPC codes, and PCTCs.

In Figures 6.5(a) and 6.5(b), we can see that linear soft-output MMSE detection in combination with Viterbi decoding is Pareto-optimal to an SNR operating point of about 21.5 dB. In order to achieve better performance with non-iterative MIMO decoding, LDPC or turbo codes need to be employed. Note that for a given silicon complexity (or symbol-rate efficiency) the performance of LDPC codes is slightly worse than that of turbo codes (i.e., about 0.5 dB).

Iterative MIMO decoding is required to further improve the performance. Surprisingly, convolutional codes outperform (in terms of performance and complexity) turbo codes and LDPC codes. The reason for a lower SNR operating point can be addressed to using small block-lengths and correlation among frequency and space (see, e.g., [140]). In addition, our simulation results also indicate that the performance strongly depends on the employed channel code; we observed that the inter-play between MIMO detector and channel code seems to be important for the performance of iterative MIMO decoding.[5]

Conclusions

It was shown that iterative MIMO decoding is able to improve the performance by more than 8 dB SNR, while increasing the silicon complexity by approximately 1.5 MGE. Hence, iterative MIMO decoding is able to offer tremendous performance improvements in practical systems (compared to that of non-iterative MIMO decoding) at the cost of approximately two to four times higher silicon area. For the considered scenario, CCs are better suited than LDPC or turbo codes. For non-iterative decoding, LDPC and turbo codes yield superior performance and complexity than if using CCs.

6.2.2 SISO MMSE PIC vs. Soft-Output STS-SD

In this section, we compare the tradeoffs realized by SISO MMSE PIC and soft-output (SO) STS-SD for different channel codes. For the

[5] Recall, e.g., the observation made in [78]. Therein it was shown that anti-Gray mapping leads to better performance than Gray mapping, if iterative detection and decoding is performed in AWGN channels using 16-QAM. Hence, simply replacing a component in iterative systems with better performance in non-iterative scenarios does not necessarily lead to better performance in iterative systems.

SISO MMSE PIC algorithm, the same parameters as in the previous section have been used. For the SO STS-SD algorithm, the following assumptions have been made.

SO STS-SD Complexity Since the SO STS-SD algorithm described in Section 4.7 has a variable complexity, we employ early termination with MF scheduling (see Section 4.4.2). Hence, we consider —in contrast to the average throughput shown in (4.54)— the *guaranteed* throughput of the SO STS-SD implementation

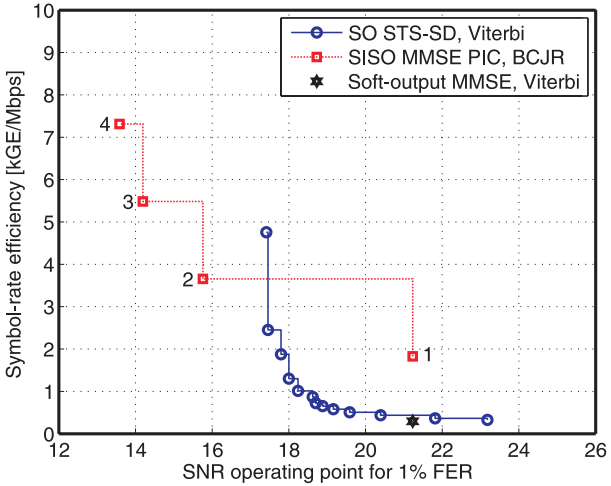
$$\Theta_{\text{STS}} = \frac{RQM_T}{D_{\text{avg}}} \tilde{f}_{\text{clk}} \quad [\text{bit/s}]$$

with the average run-time constraint D_{avg} (using a margin of $M = M_{\text{T}}$) and $\tilde{f}_{\text{clk}}$ corresponds to the technology-scaled clock frequency of the SO STS-SD implementation in 90 nm CMOS technology. The hardware-efficiency of SO STS-SD is $AT_{\text{STS}} = A_{\text{STS}}\Theta_{\text{STS}}$ using the the implementation results from Table 4.3, i.e., $A_{\text{STS}} = 56.8\,\text{kGE}$. The technology-scaled clock frequency is $\tilde{f}_{\text{clk}} = \frac{250}{90} f_{\text{clk}} = 200\,\text{MHz}$ in a 90 nm CMOS process (72 MHz for 250 nm CMOS) and the preprocessing area corresponds to that of the MMSE-QRD implementation described in [16] and is $A_{\text{pp}} = 251\,\text{kGE}$.

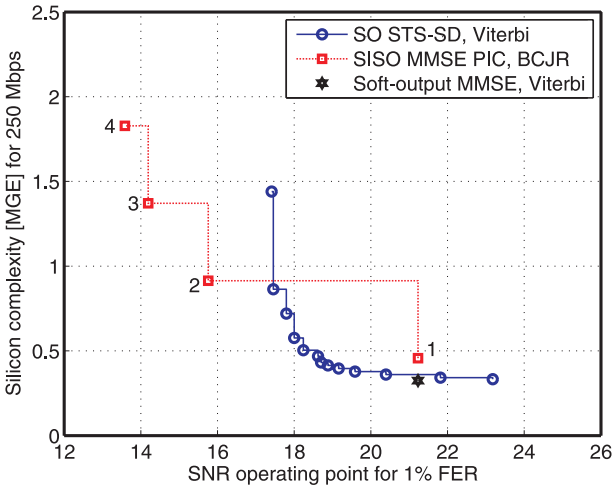
SO STS-SD Performance In order to optimize the SNR operating point of the SO STS-SD, the LLR clipping parameter $L_{\max}$ is chosen in accordance with the average run-time constraint D_{avg} (see Section 4.6.5). Further improvements in terms of the SNR operating point has been achieved by employing column-sorting, channel-matrix regularization (MMSE-SQRD), and LLR correction. The LLR correction approach used here corresponds to using two different LLR correction curves: The extrinsic LLRs are divided by a factor of two if the detector has been terminated early and no scaling is performed otherwise. Simulations have shown that this simple LLR correction scheme substantially improves the overall performance.

Tradeoff for Convolutional Codes

Figure 6.6 compares the tradeoffs for CCs using constraint length 7. For linear soft-output MMSE detection and SO STS-SD, Viterbi de-



(a) Performance vs. symbol-rate efficiency.



(b) Performance vs. silicon complexity.

Figure 6.6: Tradeoffs: SISO MMSE PIC, SO STS-SD, and linear soft-output MMSE detection for convolutional codes.

coding has been considered. To this end, the implementation results described in [16] have been used for the Viterbi decoder. For the SISO MMSE-PIC algorithm, we considered the 64-state M-BCJR implementation described in Section 5.1.3. The numbers next to the curve of SISO MMSE PIC corresponds to the number of iterations I.

Both trade-off figures show that the SO STS-SD is able to cover a large tradeoff region, which is parametrized by D_{avg} and $L_{\max}$. The linear SO MMSE detector requires slightly less silicon-complexity compared to that of SO STS-SD (at the same SNR operating point). The SISO MMSE PIC detector is the least efficient detector in the first iteration (caused by the fact that an M-BCJR decoder is used and preprocessing is performed at symbol rate). However, the SISO MMSE PIC is able to attain significantly better performance for $I > 1$.

Tradeoff for LDPC Codes

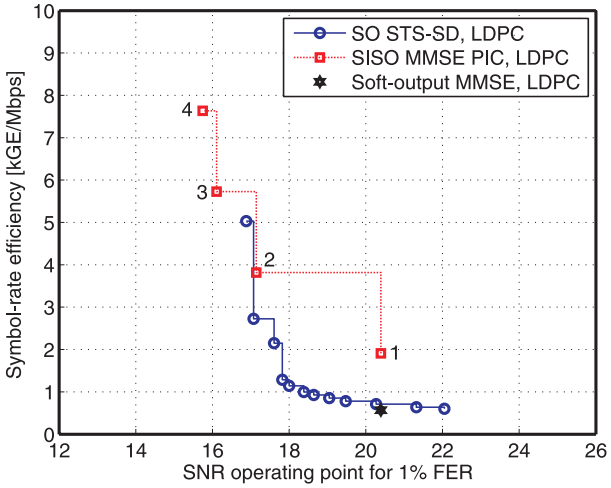
Figure 6.7 compares the tradeoff results for QC-LDPC codes with $I_{\text{LDPC}} = 5$. The SO STS-SD allows to cover a large tradeoff region and is even able to attain similar performance and complexity as the SISO MMSE PIC detector with $I = 2$. Hence, iterative MIMO decoding with the SISO MMSE PIC algorithm seems to be only beneficial if using more than two iterations.

Tradeoff for Turbo Codes

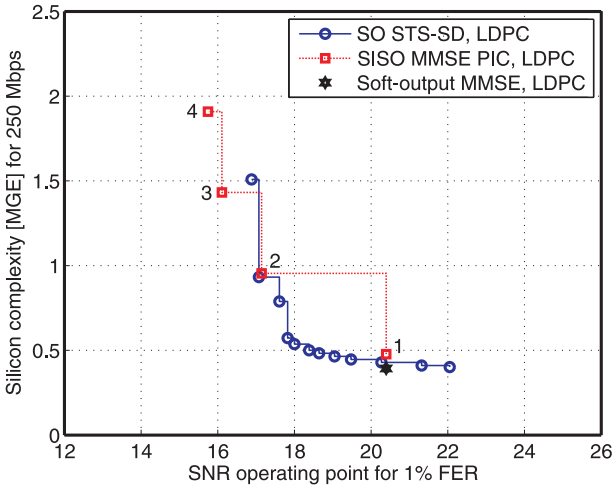
The results in Figure 6.8 show the tradeoff curves for turbo codes (using $I_{\text{PCTC}} = 5$). Note that the behavior is similar to that of LDPC codes. The SISO MMSE PIC algorithm seems to be only beneficial for $I > 2$. Hence, we conclude that for more sophisticated channel codes (such as LDPC and turbo codes), iterative MIMO decoding requires a larger number of iterations in order to outperform the SO STS-SD and entails an (often significant) overhead in terms of silicon complexity.

Summary and Conclusions

From the tradeoff results provided in Figures 6.6, 6.7, and 6.8, we see that the SO STS-SD algorithm enables to cover a large tradeoff region

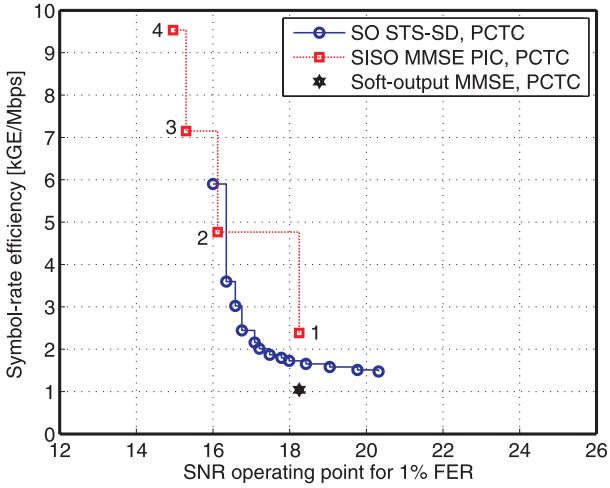


(a) Performance vs. symbol-rate efficiency.

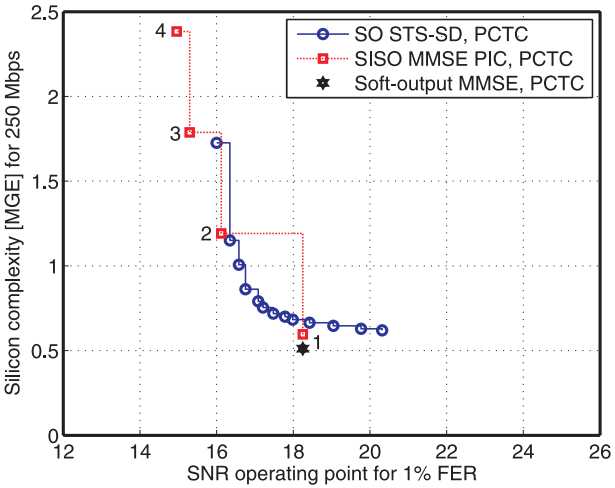


(b) Performance vs. silicon complexity.

Figure 6.7: Tradeoffs: SISO MMSE PIC, SO STS-SD, and linear soft-output MMSE detection for LDPC codes.



(a) Performance vs. symbol-rate efficiency.



(b) Performance vs. silicon complexity.

Figure 6.8: Tradeoffs: SISO MMSE PIC, SO STS-SD, and linear soft-output MMSE detection for turbo codes.

by tuning of the LLR clipping parameter and the (average) run-time constraint. In particular, the SO STS-SD covers a performance range of approximately 5 dB SNR and in combination with sophisticated channel codes (such as QC-LDPC or turbo codes) can even outperform iterative MIMO decoding with SISO MMSE PIC and $I = 2$. If CCs are used and more than one iteration can be afforded, iterative MIMO decoding using the SISO MMSE PIC algorithm yields better performance than soft-output detection using the SO STS-SD; this improvement, however, entails an (often significant) increase in terms of circuit area, i.e., the required circuit area grows linearly with the number of iterations.

Chapter 7

Summary, Conclusion, and Outlook

7.1 Summary

Iterative MIMO decoding was shown to be key to approach the fundamental performance limits of MIMO wireless communication systems. However, practical implementations of iterative MIMO decoding is—due to the high signal processing complexity required for soft-input soft-output MIMO detection—extremely challenging, even when targeting state-of-the-art process technology nodes. It is therefore of paramount importance to jointly optimize algorithm- and VLSI implementation-aspects in order to enable iterative MIMO decoding in practical systems.

Previous work on implementation aspects of MIMO decoding mainly focused on techniques that require low implementation complexity and hence, achieve rather poor performance. In this thesis, we investigated efficient MIMO decoding schemes that are able to approach optimum performance. To this end, we developed and optimized key algorithms for iterative MIMO decoding (for soft-input soft-output MIMO detection and for SISO channel decoding) and proposed corresponding VLSI architectures. ASIC implementation results and analysis of the associated performance/complexity trade-

offs are used to demonstrate that iterative MIMO decoding is feasible in practical systems.

Soft-Input Soft-Output MMSE PIC

The soft-input soft-output minimum mean-square error (MMSE) parallel interference cancellation (PIC) algorithm developed by Wang and Poor in 1999 [20] has been studied. In order to enable implementation of SISO detection in practical systems, we proposed a novel method on algorithmic level that substantially reduces the number of required matrix inversions. This method was shown to be key for economic hardware implementation. In order to further reduce the complexity of the algorithm, various existing techniques have been applied, which ultimately led to a low-complexity high-performance SISO detection algorithm. The performance of the resulting algorithm was shown to be able to approach the (error-rate) performance of the optimum SISO detector in systems employing iterative MIMO decoding.

In order to evaluate the performance and complexity based on implementation results, a reference VLSI architecture has been designed. To this end, a systolic network of dedicated processing units has been developed, which is able to perform the required tasks efficiently in hardware. Matrix inversion has been identified as the most critical component of the decoder. Thus, a custom high-performance matrix inversion unit based on the LU-decomposition has been designed. The resulting architecture is able to perform high-throughput SISO MMSE PIC in MIMO systems using four spatial streams and supports modulation schemes ranging from BPSK to 64-QAM, while achieving close-to floating-point performance.

The architecture has been implemented in 90 nm CMOS technology and corresponding post-layout ASIC implementation results demonstrate that SISO detection based on the proposed SISO MMSE PIC algorithm is able to achieve more than 820 Mbps per iteration at only 410 kGE (including the complexity of preprocessing). Moreover, comparison with state-of-the-art hard-output and soft-output MIMO detection schemes has shown that SISO capability entails a twofold reduction in hardware efficiency, but enables to obtain tremendous performance gains in practical systems. Based on these results, it was demonstrated that low-complexity and high-performance SISO detec-

tion for iterative MIMO decoding is feasible in practical systems.

SISO Single Tree-Search Sphere Decoding

A novel soft-input soft-output MIMO detector based on single tree-search sphere decoding (STS-SD) has been proposed. Key for low complexity of the proposed algorithm are tightening of the tree-pruning criterion, clipping of the extrinsic LLRs built into the tree search, and a novel method for incorporating compensation of self-interference in LLRs—caused by channel-matrix regularization—into the tree search. Finally, we proposed an LLR correction method, which was demonstrated to achieve substantial performance improvements at low additional computational complexity. Our simulation results showed that the SISO STS-SD algorithm offers a wide range of performance/complexity tradeoffs and clearly outperforms state-of-the-art SISO detectors for MIMO systems. In addition, SISO STS-SD achieves close-to-optimal—in the sense of outage capacity—performance. The algorithm is able to realize an entire family of detectors with (error-rate) performance ranging from exact max-log SISO performance to low-complexity successive interference cancellation, which renders the detector interesting for practical applications.

In addition, we described an architecture of the soft-output (SO) STS-SD. Corresponding VLSI implementation results demonstrate that the proposed soft-output STS-SD algorithm is only 58% larger than that of a reference hard-output sphere decoder implementation; this enables us to conclude that SO STS-SD is well-suited for near-optimal soft-output MIMO detection in practical systems.

Soft-Input Soft-Output Channel Decoding

In order to complete the picture of iterative MIMO decoding, we analyzed the performance and implementation complexity associated with SISO channel decoding. To this end, VLSI architectures for SISO decoding of convolutional codes (CCs), quasi-cyclic (QC) low-density parity check (LDPC) codes, and the turbo codes described in the 3GPP LTE standard [11] have been developed.

Convolutional Codes The BCJR algorithm [34] has been considered for SISO decoding of convolutional codes (CCs). A high-throughput VLSI architecture for a windowed variant of the BCJR algorithm, known as the M-BCJR algorithm, has been designed. The architecture bases on the design developed in [36] and was optimized for hardware-efficiency and throughput. In order to evaluate the performance/complexity tradeoff with SISO decoding of CCs, decoders supporting 4-, 8-, 16-, 32-, and 64-trellis-states have been implemented. Corresponding ASIC implementation results in 180 nm CMOS technology demonstrate that SISO decoding of CCs with the M-BCJR algorithm is able to achieve 375 Mbps with circuit area ranging from 23.1 kGE to 243.5 kGE (depending on the number of trellis-states). Comparison with a reference Viterbi decoder [16] has shown that SISO capability entails a twofold increase in terms of circuit complexity. The 64-state M-BCJR decoder is the first of its kind and compliant to the IEEE 802.11n standard [2].

QC-LDPC Codes For SISO decoding of QC-LDPC codes, a fully reconfigurable VLSI architecture has been designed, which is able to decode virtually any QC-LDPC code that fits into the allocated memories. To this end, a novel cyclic shifter has been developed and memory requirements have been reduced significantly by clipping of the decoder-internal messages. The QC-LDPC decoder has been optimized for the codes proposed in IEEE 802.11n [2] and fabricated in 180 nm CMOS technology. Measurement results and comparison with dedicated (not reconfigurable) LDPC decoders for IEEE 802.11n demonstrate that flexibility of the architecture does not lead to a performance penalty in terms of area, throughput, and power. The decoder is compliant to IEEE 802.11n and achieves up to 780 Mbps while requiring only 349.5 kGE. The decoder is able to compute soft-outputs and hence, suitable for iterative MIMO decoding.

Turbo Codes A high-throughput turbo decoder for the 3GPP LTE standard [11] has been developed. The codes specified in this standard enable contention-free access to the memories, which substantially alleviates the interleaver bottleneck. In order to achieve high throughput, we proposed a suitable high-performance interleaver architecture.

Decoding is performed by a high-throughput radix-4 M-BCJR architecture. The turbo decoder employs eight M-BCJR instances, has been implemented in 130 nm CMOS technology, and achieves up to 520 Mbps. Comparison to reference designs shows that the proposed architecture is more efficient (in terms of area per throughput) and achieves the highest turbo-decoding throughput reported in the literature. This design demonstrates that iterative MIMO decoding based on turbo codes is feasible in practice.

Performance/Complexity Tradeoff The tradeoffs underlying the implemented SISO channel decoders have been investigated.

- It was shown that M-BCJR algorithm achieves high throughput in practice and is suitable for low-complexity SISO decoding in systems where rather weak coding is required. For stronger CCs, the associated VLSI implementation complexity quickly gets prohibitive in terms of silicon area, which asks for more sophisticated coding schemes with corresponding efficient channel decoders.
- QC-LDPC codes offer a excellent error-correction performance, while supporting a high degree of flexibility in terms of achievable throughput, code-rate, and block-lengths. Implementation results demonstrate that the performance is, in general, better than that of the M-BCJR algorithm (for a given complexity).
- Turbo codes based on contention-free interleavers provide excellent error-rate performance. The associated implementation complexity is, in general, higher than that of the M-BCJR algorithm or QC-LDPC decoders for the same throughput.

In summary, a clear advantage for one of the three coding/decoding schemes could not be identified. This observation implies that system designers need to decide which of the above mentioned properties apply for the problem at hand.

Performance/Complexity Tradeoffs

In order to compare the performance and VLSI-implementation complexity of iterative MIMO decoding, two complexity measures have

been introduced. The first measure indicates the hardware-efficiency of all tasks that need to be executed at symbol-rate. The second measure estimates the silicon complexity required for iterative MIMO decoding to meet a given target throughput; this measure shows that the circuit area of iterative MIMO decoding scales linearly in the number of iterations.

Impact of Channel Code If soft-output (i.e., non-iterative) MIMO detection is considered, linear SO MMSE detection in combination with Viterbi decoding was shown to be the best choice to achieve low complexity. For improved performance, LDPC and turbo codes were shown to outperform CCs by approximately 2 dB to 3 dB SNR, while requiring 20% to 60% more circuit area (compared to Viterbi decoding).

Tradeoff comparisons based on SISO MMSE PIC with the M-BCJR algorithm have shown that iterative MIMO decoding (for more than one iteration) outperform non-iterative schemes. We finally observed that iterative MIMO decoding is able to improve the performance (compared to non-iterative decoding) by 5 dB to 8 dB SNR, while the circuit area increases (approximately) by a factor of 2.5 to 5, respectively.

SISO MMSE PIC vs. SO STS-SD We compared the performance and complexity of the SISO MMSE PIC algorithm (using iterative MIMO decoding) with SO STS-SD. Thanks to the tunability offered by SO STS-SD, a large tradeoff region can be covered with this algorithm. In combination with more sophisticated channel codes (such as LDPC or turbo codes), the SO STS-SD algorithm is even able to (slightly) outperform SISO MMSE PIC using two iterations. The SISO MMSE PIC algorithm used for iterative MIMO decoding is, however, able to outperform SO STS-SD for more than two iterations.

7.2 Conclusion

Based on the provided results, it was demonstrated that iterative MIMO decoding is feasible in practical systems. For an IEEE 802.11n-like MIMO system, iterative MIMO detection and channel decoding

entails a complexity which ranges (approximately) between 0.5 MGE and 2 MGE for two iterations. The improvements in terms of SNR operating point are substantial, i.e., the SNR operating point can be improved by more than 8 dB SNR in some situations. Hence, iterative MIMO decoding is a viable solution for improved link reliability, longer range, and higher throughput in practical MIMO systems.

A clear advantage for either the SISO MMSE PIC detector or the SO STS-SD algorithm could not be identified. The SO STS-SD algorithm is tunable, provides max-log optimal performance or low-computational complexity, and can be extended with max-log optimal soft-input capability. The SISO MMSE PIC is not tunable, but offers —in combination with a SISO channel decoder and iterative MIMO decoding— significant performance gains. In summary, if the complexity associated with iterative MIMO decoding is tolerable (e.g., if the resulting silicon complexity, latency, power consumption etc. is tolerable), the SISO MMSE PIC algorithm is a viable competitor to the SO STS-SD algorithm. However, if optimal (soft-output) performance and tunability of the algorithm to different performance and complexity requirements are the key requirements, the SO STS-SD is the better choice.

7.3 Outlook

In view of the obtained results throughout this thesis, we briefly outline remaining research topics for iterative MIMO decoding in the following two paragraphs.

Theory and Algorithms

- Radio-frequency (RF) impairments, such as, I/Q imbalance, phase-noise, power amplifier non-linearities etc., were recently shown to pose significant problems for SD-based MIMO detection algorithms [188]. Detailed analysis of the impact on performance of iterative MIMO decoding in the presence of RF-impairments is an open research topic that is of practical relevance. Furthermore, low-complexity techniques to mitigate detrimental effects should be investigated as well.

- Analytical tools to characterize the complexity distribution of hard-output SD have recently become available [136]. So far, not much is known about the complexity distribution of soft-output SD or SISO STS-SD. A detailed complexity analysis of those algorithms remains an open topic.

- Throughout this thesis, no special assumptions on the structure of the channel matrix has been made. Channel matrices of ISI channels, for example, exhibit a well-defined (Toeplitz) structure. Intuitively, exploitation of this structure in tree-search-based detection algorithms should be able to reduce the complexity. Interesting results in this direction have recently become available for fast decoding of STBCs, e.g., [46].

VLSI Implementation Aspects

- Since no VLSI implementation of the SISO STD-SD algorithm has been designed, soft-input extension of the SO STS-SD architecture is still an open problem. Schnorr-Euchner enumeration in the presence of priors is one of the most challenging aspects.

- No complete iterative MIMO decoder has been integrated in this thesis (i.e., we provided implementation results for the most critical components). Hence, the design of an iterative MIMO decoder and integration into a physical layer of a MIMO transceiver is certainly an interesting and challenging engineering task.

- Power-consumption-aspects have not been considered in this thesis. However, portable wireless devices require energy-efficient implementations to maximize battery lifetime. Hence, all considered algorithms and implementations have potential to be optimized with respect to energy-efficiency.

Appendix A

Mathematical Derivations for SISO MMSE PIC

A.1 The MMSE Filter Vector

The MMSE filter vector for interference suppression on the ith stream (after parallel interference cancellation) satisfies

$$\tilde{\mathbf{w}}_i^H = \underset{\tilde{\mathbf{w}}^H \in \mathbb{C}^{1\times M_\mathrm{T}}}{\arg\min}\ \mathbb{E}\Big[\big|\tilde{\mathbf{w}}^H \hat{\mathbf{y}}_i - s_i\big|^2\Big]. \tag{A.1}$$

Setting $\frac{\partial}{\partial \tilde{\mathbf{w}}_i^H}\mathbb{E}\Big[\big|\tilde{\mathbf{w}}_i^H \hat{\mathbf{y}}_i - s_i\big|^2\Big] = 0$ leads to the well-known orthogonality principle [42]

$$\mathbb{E}\Big[\hat{\mathbf{y}}_i\big(\tilde{\mathbf{w}}_i^H \hat{\mathbf{y}}_i - s_i\big)^*\Big] = \mathbf{0}_{M_\mathrm{R}\times 1} \tag{A.2}$$

where expectation is over the noise $\mathbf{n}$, the soft-symbol estimation errors e_j $(\forall j)$ of (3.2), and the transmit symbols s_j $(\forall j)$. By noting that (A.2) can be rewritten to $\mathbb{E}\big[\hat{\mathbf{y}}_i \hat{\mathbf{y}}_i^H\big]\,\mathbf{w}_i = \mathbb{E}[\hat{\mathbf{y}}_i s_i^*]$ and using the definition of the interference-canceled received vector in (3.6) we ob-

tain [73]

$$\left(E_{\mathrm{s}}\mathbf{h}_i\mathbf{h}_i^H + \sum_{j\neq i} E_j\mathbf{h}_j\mathbf{h}_j^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{R}}}\right)\tilde{\mathbf{w}}_i = \mathbf{A}\tilde{\mathbf{w}}_i = E_s\mathbf{h}_i \qquad \text{(A.3)}$$

Left-multiplication of (A.3) with $\mathbf{A}^{-1}$ followed by conjugate transposition, leads to the MMSE filter vector

$$\tilde{\mathbf{w}}_i^H = E_s\mathbf{h}_i^H\left(E_{\mathrm{s}}\mathbf{h}_i\mathbf{h}_i^H + \sum_{j\neq i} E_j\mathbf{h}_j\mathbf{h}_j^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{R}}}\right)^{-1}$$

which can be written more compactly as [42]

$$\tilde{\mathbf{w}}_i^H = E_{\mathrm{s}}\mathbf{h}_i^H\left(\mathbf{H}\widetilde{\mathbf{\Lambda}}_i\mathbf{H}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{R}}}\right)^{-1} \qquad \text{(A.4)}$$

with $\widetilde{\mathbf{\Lambda}}_i$ being a $M_{\mathrm{T}} \times M_{\mathrm{T}}$ real-valued diagonal matrix

$$\tilde{\Lambda}_{j,j} = \begin{cases} E_j, & j \neq i \\ E_s, & j = i \end{cases}$$

where the variances E_j are defined in (3.3).

A.2 Efficient MMSE Filter Computation

A.2.1 Single Matrix Inversion

To reduce the number of matrix inversions required to compute the M_{T} MMSE filter vectors, we reformulate the "standard" MMSE filter formulation [42] given in (3.9)

$$\tilde{\mathbf{w}}_i^H = E_{\mathrm{s}}\mathbf{h}_i^H\left(\mathbf{H}\widetilde{\mathbf{\Lambda}}_i\mathbf{H}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{R}}}\right)^{-1} = E_{\mathrm{s}}\mathbf{h}_i^H\mathbf{A}_i^{-1} \qquad \text{(A.5)}$$

where the matrix $\mathbf{A}_i$ depends on the stream index i through $\widetilde{\mathbf{\Lambda}}_i$. In order to avoid dependence on the ith stream in the matrix that needs to be inverted, we define a new filter vector

$$\mathbf{w}_i^H = \mathbf{h}_i^H\left(\mathbf{H}\mathbf{\Lambda}\mathbf{H}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{R}}}\right)^{-1} = \mathbf{h}_i^H\mathbf{A}^{-1} \qquad \text{(A.6)}$$

with $\mathbf{\Lambda}$ being a $M_{\mathrm{T}} \times M_{\mathrm{T}}$ real-valued diagonal matrix with entries $\Lambda_{i,i} = E_i$ $(\forall i)$. We emphasize that the matrix $\mathbf{A}$ to be inverted in (A.6) does not longer depend on i. Note that if this new filter (A.6) can be used for MMSE equalization, all M_{T} MMSE filter vectors can be computed from a *single* matrix inversion. In the remainder of the ensuing discussion, we show that the new MMSE filter vector $\mathbf{w}_i^H$ in (A.6) is simply a scaled version of $\tilde{\mathbf{w}}_i^H$ in (A.5). Then, we prove that scaling of the filter vectors by a constant does not affect the performance of the SISO MMSE PIC algorithm.

We start by using the matrix inversion lemma [87]

$$\left(\mathbf{A}_i + \mathbf{u}\mathbf{v}^H\right)^{-1} = \mathbf{A}_i^{-1} - \frac{\mathbf{A}_i^{-1}\mathbf{u}\mathbf{v}^H\mathbf{A}_i^{-1}}{1 + \mathbf{v}^H\mathbf{A}_i^{-1}\mathbf{u}}. \tag{A.7}$$

and set $\mathbf{u}\mathbf{v}^H = \mathbf{h}_i(E_i - E_{\mathrm{s}})\mathbf{h}_i^H$ where $\mathbf{u} = \mathbf{h}_i$. It is important to realize that the left-hand side (LHS) of (A.7) corresponds to $\mathbf{A}^{-1}$ defined in (A.6). Using (A.7), we can write

$$\begin{aligned}\mathbf{w}_i^H &= \mathbf{h}_i^H\left(\mathbf{A}_i + \mathbf{u}\mathbf{v}^H\right)^{-1} \\ &= \mathbf{h}_i^H\mathbf{A}_i^{-1} - \frac{\mathbf{h}_i^H\mathbf{A}_i^{-1}\mathbf{h}_i(E_i - E_{\mathrm{s}})\mathbf{h}_i^H\mathbf{A}_i^{-1}}{1 + (E_i - E_{\mathrm{s}})\mathbf{h}_i^H\mathbf{A}_i^{-1}\mathbf{h}_i}\end{aligned} \tag{A.8}$$

and use the definition given in (A.5) to rewrite (A.8) according to

$$\begin{aligned}\mathbf{w}_i^H &= {E_{\mathrm{s}}}^{-1}\tilde{\mathbf{w}}_i^H - \frac{{E_{\mathrm{s}}}^{-1}\tilde{\mathbf{w}}_i^H\mathbf{h}_i(E_i - E_{\mathrm{s}}){E_{\mathrm{s}}}^{-1}\tilde{\mathbf{w}}_i^H}{1 + (E_i - E_{\mathrm{s}}){E_{\mathrm{s}}}^{-1}\tilde{\mathbf{w}}_i^H\mathbf{h}_i} \\ &= {E_{\mathrm{s}}}^{-1}\tilde{\mathbf{w}}_i^H\left(\frac{1 + (E_i - E_{\mathrm{s}}){E_{\mathrm{s}}}^{-1}\tilde{\mathbf{w}}_i^H\mathbf{h}_i - (E_i - E_{\mathrm{s}}){E_{\mathrm{s}}}^{-1}\mathbf{h}_i^H\tilde{\mathbf{w}}_i}{1 + (E_i - E_{\mathrm{s}}){E_{\mathrm{s}}}^{-1}\tilde{\mathbf{w}}_i^H\mathbf{h}_i}\right).\end{aligned}$$

From (A.4) follows that $\tilde{\mathbf{w}}_i^H\mathbf{h}_i = \mathbf{h}_i^H\tilde{\mathbf{w}}_i$ and hence, both terms are real-valued; this property enables to establish the following relation between the two MMSE filter vectors in (A.5) and (A.5)

$$\mathbf{w}_i^H = \tilde{\mathbf{w}}_i^H\left(\frac{1}{E_{\mathrm{s}} + (E_i - E_{\mathrm{s}})\tilde{\mathbf{w}}_i^H\mathbf{h}_i}\right) = \tilde{\mathbf{w}}_i^H c_i \tag{A.9}$$

where the constants $c_i \in \mathbb{R}$ $(\forall i)$ only depend on the ith stream.

Now, we show that scaling of the MMSE filter vectors does *not* affect the a posteriori LLRs resulting from the SISO MMSE PIC algorithm. The only terms in (3.17) that depend on the MMSE filter vector and influence a posteriori LLR computation correspond to $z_i = \tilde{\mathbf{w}}_i^H \hat{\mathbf{y}}_i$ in (3.10), $\mu_i = \tilde{\mathbf{w}}_i^H \mathbf{h}_i$ in (3.11), and

$$\nu_i^2 = \tilde{\mathbf{w}}_i^H \left(\sum_{j \neq i} E_j \mathbf{h}_j \mathbf{h}_j^H + N_\mathrm{o} \mathbf{I} \right) \tilde{\mathbf{w}}_i$$

of (3.12). Using $\mathbf{w}_i^H = \tilde{\mathbf{w}}_i^H c_i$ instead of $\tilde{\mathbf{w}}_i^H$ in (3.17) leads to

$$\begin{aligned} \frac{|z_i - \mu_i a|^2}{\nu_i^2} &= \frac{\left| \mathbf{w}_i^H (\hat{\mathbf{y}}_i - \mathbf{h}_i a) \right|^2}{\mathbf{w}_i^H \left(\sum_{j \neq i} E_j \mathbf{h}_j \mathbf{h}_j^H + N_\mathrm{o} \mathbf{I} \right) \mathbf{w}_i} \\ &= \frac{\left| c_i \tilde{\mathbf{w}}_i^H (\hat{\mathbf{y}}_i - \mathbf{h}_i a) \right|^2}{c_i \tilde{\mathbf{w}}_i^H \left(\sum_{j \neq i} E_j \mathbf{h}_j \mathbf{h}_j^H + N_\mathrm{o} \mathbf{I} \right) \tilde{\mathbf{w}}_i c_i^*} \\ &= \frac{c_i^2 \left| \tilde{\mathbf{w}}_i^H (\hat{\mathbf{y}}_i - \mathbf{h}_i a) \right|^2}{c_i^2 \tilde{\mathbf{w}}_i^H \left(\sum_{j \neq i} E_j \mathbf{h}_j \mathbf{h}_j^H + N_\mathrm{o} \mathbf{I} \right) \tilde{\mathbf{w}}_i} \end{aligned}$$

and hence the constant c_i cancels out in the LLR computation, i.e., the output of the SISO MMSE PIC algorithm remains *unchanged* if a scaled version of the MMSE filter vector is used. Thus, the low-complexity MMSE filter vectors obtained in (A.6) are equivalent to the ones in (A.5).

The final step to attain low computational complexity corresponds to realizing that (A.6) can be used to compute all M_T MMSE filter vectors *concurrently* as follows

$$\mathbf{W}^H = \mathbf{H}^H \left(\mathbf{H} \mathbf{\Lambda} \mathbf{H}^H + N_\mathrm{o} \mathbf{I}_{M_\mathrm{R}} \right)^{-1} \tag{A.10}$$

where the column $\mathbf{h}_i^H$ has been replaced with $\mathbf{H}^H = [\, \mathbf{h}_1 \ \cdots \ \mathbf{h}_{M_\mathrm{T}} \,]^H$. Hence, $\mathbf{W}^H = [\, \mathbf{w}_1 \ \cdots \ \mathbf{w}_{M_\mathrm{T}} \,]^H$ contains *all* MMSE filter vectors on its rows and requires to compute only a *single* matrix inversion.

A.2.2 Further Methods for Complexity Reduction

The key drawback of computing (A.10) is that—if more receive than transmit antennas are used—it requires to invert a $M_{\mathrm{R}} \times M_{\mathrm{R}}$ matrix, i.e., $\mathbf{H}\mathbf{\Lambda}\mathbf{H}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{R}}}$ is of dimension $M_{\mathrm{R}} \times M_{\mathrm{R}}$. In this paragraph, we show how the MMSE filter matrix (A.10) can be computed by inverting a matrix of dimension $M_{\mathrm{T}} \times M_{\mathrm{T}}$. We start the simplification by defining

$$\widetilde{\mathbf{H}} = \mathbf{H}\mathbf{\Lambda}^{\frac{1}{2}}.$$

where $\mathbf{\Lambda}^{\frac{1}{2}}$ is a real-valued $M_{\mathrm{T}} \times M_{\mathrm{T}}$ diagonal matrix with $\Lambda_{i,i}^{\frac{1}{2}} = \sqrt{E_i}$ for $i = 1, \ldots, M_{\mathrm{T}}$ and $\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{\Lambda}^{\frac{1}{2}} = \mathbf{\Lambda}$. The MMSE filter matrix in (A.10) can now be rewritten as

$$\mathbf{W}^H = \mathbf{\Lambda}^{-\frac{1}{2}}\widetilde{\mathbf{H}}^H\left(\widetilde{\mathbf{H}}\widetilde{\mathbf{H}}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{R}}}\right)^{-1}. \tag{A.11}$$

The derivation shown in the following bases on the singular value decomposition (SVD) $\widetilde{\mathbf{H}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$, where $\mathbf{U}$ is of dimension $M_{\mathrm{R}} \times M_{\mathrm{T}}$ and $\mathbf{U}\mathbf{U}^H = \mathbf{I}_{M_{\mathrm{R}}}$, $\mathbf{\Sigma}$ is an $M_{\mathrm{T}} \times M_{\mathrm{T}}$-dimensional diagonal matrix which contains the singular values σ_i $(i = 1, \ldots, M_{\mathrm{T}})$ of $\widetilde{\mathbf{H}}$ on its main diagonal, i.e., $\Sigma_{i,i} = \sigma_i$ $(\forall i)$, and $\mathbf{V}$ is of dimension $M_{\mathrm{T}} \times M_{\mathrm{T}}$ and unitary, i.e., $\mathbf{V}\mathbf{V}^H = \mathbf{I}_{M_{\mathrm{T}}}$ [189]. Application of the SVD to the RHS of (A.11) and omitting the matrix $\mathbf{\Lambda}^{-\frac{1}{2}}$ leads to

$$\begin{aligned}\widetilde{\mathbf{H}}^H\left(\widetilde{\mathbf{H}}\widetilde{\mathbf{H}}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{R}}}\right)^{-1} &= \widetilde{\mathbf{H}}^H\left(\mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^H\mathbf{U}^H + N_{\mathrm{o}}\mathbf{U}\mathbf{U}^H\right)^{-1} \\ &= \widetilde{\mathbf{H}}^H\mathbf{U}\left(\mathbf{\Sigma}\mathbf{\Sigma}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{T}}}\right)^{-1}\mathbf{U}^H. \end{aligned} \tag{A.12}$$

Note that the matrix in (A.12) to be inverted is now of dimension $M_{\mathrm{T}} \times M_{\mathrm{T}}$. Further manipulations yield

$$\widetilde{\mathbf{H}}^H\mathbf{U}\left(\mathbf{\Sigma}\mathbf{\Sigma}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{T}}}\right)^{-1}\mathbf{U}^H = \mathbf{V}\mathbf{\Sigma}^H\left(\mathbf{\Sigma}\mathbf{\Sigma}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{T}}}\right)^{-1}\mathbf{U}^H.$$

Since $\mathbf{\Sigma}^H$ and $\left(\mathbf{\Sigma}\mathbf{\Sigma}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{T}}}\right)^{-1}$ are diagonal matrices, we can exploit commutativity and write

$$\mathbf{V}\mathbf{\Sigma}^H\left(\mathbf{\Sigma}\mathbf{\Sigma}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{T}}}\right)^{-1}\mathbf{U}^H = \mathbf{V}\left(\mathbf{\Sigma}\mathbf{\Sigma}^H + N_{\mathrm{o}}\mathbf{I}_{M_{\mathrm{T}}}\right)^{-1}\mathbf{\Sigma}^H\mathbf{U}^H.$$

This expression can now be simplified to

$$\begin{aligned}\mathbf{V}\left(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^H + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}}\right)^{-1}\boldsymbol{\Sigma}^H\mathbf{U}^H &= \mathbf{V}\left(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^H + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}}\right)^{-1}\mathbf{V}^H\mathbf{V}\boldsymbol{\Sigma}^H\mathbf{U}^H \\ &= \mathbf{V}\left(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^H + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}}\right)^{-1}\mathbf{V}^H\widetilde{\mathbf{H}}^H \\ &= \left(\mathbf{V}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^H\mathbf{V}^H + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}}\right)^{-1}\widetilde{\mathbf{H}}^H = \\ &= \left(\widetilde{\mathbf{H}}^H\widetilde{\mathbf{H}} + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}}\right)^{-1}\widetilde{\mathbf{H}}^H.\end{aligned}$$

By replacing $\widetilde{\mathbf{H}}$ by $\mathbf{H}\boldsymbol{\Lambda}^{\frac{1}{2}}$ and by using $\left(\boldsymbol{\Lambda}^{\frac{1}{2}}\right)^H = \boldsymbol{\Lambda}^{\frac{1}{2}}$, the following algebraic transformations lead to a more efficient formulation of the MMSE filter matrix

$$\begin{aligned}\left(\widetilde{\mathbf{H}}^H\widetilde{\mathbf{H}} + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}}\right)^{-1}\widetilde{\mathbf{H}}^H &= \left(\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{H}^H\mathbf{H}\boldsymbol{\Lambda}^{\frac{1}{2}} + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}}\right)^{-1}\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{H}^H \\ &= \left(\boldsymbol{\Lambda}^{\frac{1}{2}}\left(\mathbf{H}^H\mathbf{H} + N_\mathrm{o}\boldsymbol{\Lambda}^{-1}\right)\boldsymbol{\Lambda}^{\frac{1}{2}}\right)^{-1}\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{H}^H \\ &= \boldsymbol{\Lambda}^{-\frac{1}{2}}\left(\mathbf{H}^H\mathbf{H} + N_\mathrm{o}\boldsymbol{\Lambda}^{-1}\right)^{-1}\mathbf{H}^H.\end{aligned}$$

Finally, the MMSE filter representation in (A.11) can be written in more compact form as

$$\mathbf{W}^H = \boldsymbol{\Lambda}^{-1}\left(\mathbf{H}^H\mathbf{H} + N_\mathrm{o}\boldsymbol{\Lambda}^{-1}\right)^{-1}\mathbf{H}^H \tag{A.13}$$

which only requires to invert a $M_\mathrm{T} \times M_\mathrm{T}$-dimensional matrix instead of $M_\mathrm{R} \times M_\mathrm{R}$ matrices when using the representation in (A.10).

Unfortunately, the MMSE filter matrix representation in (A.13) poses significant problems in practice due to poor numerical stability. Consider the case where near-perfect a priori information is available, i.e., all variances are very small $E_i \approx 0$ $(\forall i)$. In this case, the entries of the matrix $\boldsymbol{\Lambda}^{-1}$ can get arbitrarily large and hence, computation of the MMSE filter matrix would require a prohibitively large dynamic range. In order to reduce numerical problems, we rewrite (A.13) as

$$\begin{aligned}\mathbf{W}^H &= \boldsymbol{\Lambda}^{-1}\left(\mathbf{H}^H\mathbf{H} + N_\mathrm{o}\boldsymbol{\Lambda}^{-1}\right)^{-1}\mathbf{H}^H \\ &= \left(\mathbf{H}^H\mathbf{H}\boldsymbol{\Lambda} + N_\mathrm{o}\mathbf{I}_{M_\mathrm{T}}\right)^{-1}\mathbf{H}^H.\end{aligned} \tag{A.14}$$

Thus, computation of all M_T MMSE filter vectors according to (A.14) requires less computational complexity than (A.5) and can, in general, be computed with high numerical stability.

A.3 Efficient NPI-Variance Computation

Computation of the NPI-variance in (3.12) requires high computational complexity. Dejonghe and Vandendorpe [73] proposed a method to efficiently compute the corresponding variances. In the following discussion, we combine this idea with our low-complexity MMSE filter vectors given in Appendix A.2.

To this end, we start by writing the variances in (3.12) using the (low-complexity) MMSE filter vectors $\mathbf{w}_i^H$ $(\forall i)$ as

$$\nu_i^2 = \mathbf{w}_i^H \left(\sum_{j \neq i} E_j \mathbf{h}_j \mathbf{h}_j^H + N_\mathrm{o} \mathbf{I}_{M_\mathrm{R}} \right) \mathbf{w}_i. \tag{A.15}$$

Using the definition of the matrix $\mathbf{\Lambda}$ in (A.6), we can rewrite the variance in (A.15) to

$$\begin{aligned} \nu_i^2 &= \mathbf{w}_i^H \left(\mathbf{H} \mathbf{\Lambda} \mathbf{H}^H - E_i \mathbf{h}_i \mathbf{h}_i^H + N_\mathrm{o} \mathbf{I}_{M_\mathrm{R}} \right) \mathbf{w}_i \\ &= \mathbf{w}_i^H \left(\mathbf{H} \mathbf{\Lambda} \mathbf{H}^H + N_\mathrm{o} \mathbf{I}_{M_\mathrm{R}} \right) \mathbf{w}_i - E_i \big(\mathbf{w}_i^H \mathbf{h}_i \big)^2 \end{aligned} \tag{A.16}$$

where $\mathbf{w}_i^H \mathbf{h}_i \in \mathbb{R}$ due to the fact that $c_i \tilde{\mathbf{w}}_i^H \mathbf{h}_i$ is real-valued as well (see Eq. A.9). From (A.6) it can be seen that

$$\mathbf{w}_i^H \left(\mathbf{H} \mathbf{\Lambda} \mathbf{H}^H + N_\mathrm{o} \mathbf{I}_{M_\mathrm{R}} \right) = \mathbf{h}_i^H$$

and hence, we can rewrite (A.16) to

$$\nu_i^2 = \mathbf{h}_i^H \mathbf{w}_i - E_i \big(\mathbf{w}_i^H \mathbf{h}_i \big)^2 = \mathbf{w}_i^H \mathbf{h}_i - E_i \big(\mathbf{w}_i^H \mathbf{h}_i \big)^2 \tag{A.17}$$

where the second equality follows from the fact that $\mathbf{h}_i^H \mathbf{w}_i$ is real-valued. We emphasize that computation of (A.17) requires significantly less complexity than computing (A.15).

List of Figures

List of Tables

Bibliography

[1] S. Cherry, "Edholm's law of bandwidth," *IEEE Spectrum*, vol. 41, no. 7, pp. 58–60, July 2004.

[2] *IEEE Draft Standard; Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications; Amendment 4: Enhancements for Higher Throughput*, P802.11n/D3.0, Sept. 2007.

[3] A. Paulraj, R. Nabar, and D. Gore, *Introduction to Space-Time Wireless Communications*. Cambridge Univ. Press, 2003.

[4] H. Bölcskei, D. Gesbert, C. Papadias, and A. J. van der Veen, Eds., *Space-Time Wireless Systems: From Array Processing to MIMO Communications*. Cambridge Univ. Press, 2006.

[5] D. Tse and P. Viswanath, Eds., *Fundamentals of Wireless Communication*. Cambridge Univ. Press, 2005.

[6] V. Tarokh, H. Jafarkhani, and A. R. Calderbank, "Space-time block codes from orthogonal designs," *IEEE Trans. on Inf. Th.*, vol. 45, no. 5, pp. 1456–1467, July 1999.

[7] ——, "Space-time block coding for wireless communications: performance results," *IEEE Journal on Sel. Areas in Comm.*, vol. 17, no. 3, pp. 451–460, Mar. 1999.

[8] S. M. Alamouti, "A simple transmit diversity technique for wireless communications," *IEEE Journal on Sel. Areas in Comm.*, vol. 16, no. 8, pp. 1451–1458, Oct. 1998.

[9] İ. E. Telatar, "Capacity of multi-antenna Gaussian channels," *European Trans. on Telecomm.*, vol. 10, no. 6, pp. 585–596, Sept. 1999.

[10] *IEEE Standard for Local and metropolitan area networks; Part 16, Air Interface for Fixed and Mobile Broadband Wireless Access Systems; Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed bands*, P802.16e, Dec. 2005.

[11] *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 9)*, 3GPP Organizational Partners TS 36.212, Rev. 8.3.0, May 2008.

[12] V. Erceg, L. Schumacher, P. Kyritsi, A. Molisch, D. S. Baum, A. Y. Gorokhov, C. Oestges, Q. Li, K. Yu, N. Tal, B. Dijkstra, A. Jagannatham, C. Lanzl, V. J. Rhodes, J. Medbo, D. Michelson, M. Webster, E. Jacobsen, D. Cheung, C. Prettie, M. Ho, S. Howard, B. Bjerke, L. Jengx, H. Sampath, S. Catreux, S. Valle, A. Poloni, A. Forenza, and R. W. Heath, *TGn channel models*, May 2004, IEEE 802.11 document 03/940r4.

[13] P. Fertl, J. Jaldén, and G. Matz, "Performance assessment of MIMO-BICM demodulators based on system capacity," *submitted to IEEE Trans. on Sig. Proc.*, 2009.

[14] H. Kaeslin, *Digital Integrated Circuit Design: from VLSI Architectures to CMOS Fabrication.* Cambridge Univ. Press, 2008.

[15] A. Burg, "VLSI circuits for MIMO communication systems," Ph.D. dissertation, ETH Zürich, Switzerland, 2006.

[16] A. Burg, S. Häne, M. Borgmann, D. Baum, T. Thaler, F. Carbognani, S. Zwicky, L. Barbero, C. Senning, P. Greisen, T. Peter, C. Foelmli, U. Schuster, P. Tejera, and A. Staudacher, "A 4-stream 802.11n baseband transceiver in 0.13 µm CMOS," in *Proc. of 2009 Symposium on VLSI Circuits*, Kyoto, Japan, June 2009, pp. 282–283.

[17] Y. Palaskas, A. Ravi, S. Pellerano, B. R. Carlton, M. A. Elmala, R. Bishop, G. Banerjee, R. Nicholls, S. K. Ling, N. Dinur, S. S. Taylor, and K. Soumyanath, "A 5-GHz 108-Mb/s 2×2 MIMO transceiver RFIC with fully integrated 20.5-dBm p_{1dB} power amplifiers in 90-nm CMOS," *IEEE JSSC*, vol. 41, no. 12, pp. 2746–2756, Dec. 2006.

[18] P. Petrus, Q. Sun, S. Ng, J. Cho, N. Zhang, D. Breslin, M. Smith, B. McFarland, S. Sankaran, J. Thomson, R. Mosko, A. Chen, T. Lu, Y.-H. Wang, X. Zhang, D. Nakahira, Y. Li, R. Subramanian, A. Venkataraman, P. Kumar, S. Swaminathan, J. Gilbert, W. J. Choi, and H. Ye, "An integrated draft 802.11n compliant MIMO baseband and MAC processor," in *Proc. of IEEE ISSCC*, San Francisco, CA, USA, Feb. 2007, pp. 266–602.

[19] B. M. Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Trans. on Comm.*, vol. 51, no. 3, pp. 389–399, Mar. 2003.

[20] X. Wang and H. V. Poor, "Iterative (turbo) soft interference cancellation and decoding for coded CDMA," *IEEE Trans. on Comm.*, vol. 47, no. 7, pp. 1046–1061, July 1999.

[21] C. Studer, "Sphere decoding with resource constraints," Master's thesis, ETH Zurich, Communication Technology Laboratory, Aug. 2005.

[22] A. Burg, M. Borgmann, M. Wenk, C. Studer, and H. Bölcskei, "Advanced receiver algorithms for MIMO wireless communications," in *Proc. of DATE*, vol. 1, Munich, Germany, Mar. 2006.

[23] C. Studer, A. Burg, and H. Bölcskei, "Soft-output sphere decoding: Algorithms and VLSI implementation," *IEEE Journal on Sel. Areas in Comm.*, vol. 26, no. 2, pp. 290–300, Feb. 2008.

[24] C. Studer, M. Wenk, A. Burg, and H. Bölcskei, "Soft-output sphere decoding: Performance and implementation aspects," in *Proc. of 40th Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, CA, USA, Oct. 2006, pp. 2071–2076.

[25] C. Studer, D. Seethaler, and H. Bölcskei, "Finite lattice-size effects in MIMO detection," in *Proc. of 42th Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, CA, USA, Oct. 2008.

[26] C. Studer and H. Bölcskei, "Soft-input soft-output sphere decoding," in *Proc. of IEEE ISIT*, Toronto, ON, Canada, July 2008, pp. 2007–2001.

[27] ——, "Soft-input soft-output single tree-search sphere decoding," *submitted to IEEE Trans. on Inf. Th.*, July 2008.

[28] H. Vikalo and B. Hassibi, "Modified Fincke-Pohst algorithm for low-complexity iterative decoding over multiple antenna channels," in *Proc. of IEEE ISIT*, Lausanne, Switzerland, 2002, p. 390.

[29] ——, "Towards closing the capacity gap on multiple antenna channels," in *Proc. of IEEE ICASSP*, Orlando, FL, USA, 2002, pp. 2385–2388.

[30] S. Bäro, J. Hagenauer, and M. Witzke, "Iterative detection of MIMO transmission using a list-sequential (LISS) detector," in *Proc. of IEEE ICC*, vol. 4, Anchorage, AK, USA, May 2003, pp. 2653–2657.

[31] J. Boutros, N. Gresset, L. Brunel, and M. Fossorier, "Soft-input soft-output lattice sphere decoder for linear channels," in *Proc. of IEEE GLOBECOM*, vol. 3, San Francisco, CA, USA, Dec. 2003, pp. 1583–1587.

[32] H. Vikalo, B. Hassibi, and T. Kailath, "Iterative decoding for MIMO channels via modified sphere decoder," *IEEE Trans. on Wireless Comm.*, vol. 3, no. 6, pp. 2299–2311, Nov. 2004.

[33] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bölcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE JSSC*, vol. 40, no. 7, pp. 1566–1577, July 2005.

[34] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. on Inf. Th.*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[35] C. Studer, N. Preyss, C. Roth, and A. Burg, "Configurable high-throughput decoder architecture for quasi-cyclic LDPC codes," in *Proc. of 42th Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, CA, USA, Oct. 2008.

[36] C. Benkeser, A. Burg, T. Cupaiuolo, and Q. Huang, "Design and optimization of an HSDPA turbo decoder ASIC," *IEEE JSSC*, vol. 44, no. 1, pp. 98–106, Jan. 2008.

[37] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless," in *Proc. of IEEE ISSCC*, vol. 1, San Francisco, CA, USA, 2003, pp. 150–484.

[38] H. Vikalo and B. Hassibi, "On joint detection and decoding of linear block codes on Gaussian vector channels," *IEEE Trans. on Sig. Proc.*, vol. 54, no. 9, pp. 3330–3342, Sept. 2006.

[39] R. G. Gallager, "Low density parity-check codes," Ph.D. dissertation, MIT Press, Cambridge, MA, 1963.

[40] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding," in *Proc. of IEEE ICC*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[41] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. on Inf. Th.*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

[42] M. Tüchler, A. C. Singer, and R. Kötter, "Minimum mean squared error equalization using a priori information," *IEEE Trans. on Sig. Proc.*, vol. 50, no. 3, pp. 673–683, Mar. 2002.

[43] M. Witzke, S. Bäro, F. Schreckenbach, and J. Hagenauer, "Iterative detection of MIMO signals with linear detectors," in *Proc. Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, USA, Nov. 2002, pp. 289–293.

[44] G. Caire, G. Taricco, and E. Biglieri, "Bit-interleaved coded modulation," *IEEE Trans. on Inf. Th.*, vol. 44, no. 3, pp. 927–946, May 1998.

[45] S. H. Müller-Weinfurtner, "Coding approaches for multiple antenna transmission in fast fading and OFDM," *IEEE Trans. on Sig. Proc.*, vol. 50, no. 10, pp. 2442– 2450, Oct. 2002.

[46] E. Biglieri, Yi Hong, and E. Viterbo, "On fast-decodable space-time block codes," *IEEE Trans. on Inf. Th.*, vol. 55, no. 2, pp. 524–530, Feb. 2009.

[47] S. Häne, "VLSI circuits for MIMO-OFDM physical layer," Ph.D. dissertation, ETH Zürich, Switzerland, 2008.

[48] J. Jaldén and B. Ottersten, "Parallel implementation of a soft output sphere decoder," in *Proc. of Asilomar Conf. on Signals, Systems, and Computers*, Monterey, CA, USA, Nov. 2005, pp. 581–585.

[49] P. van Emde Boas, "Another NP-complete partition problem and the complexity of computing short vectors in a lattice." *Mathematisch Instituut, Amsterdam, Rep. 81-04*, Apr. 1981.

[50] J. M. Cioffi, G. P. Dudevoir, M. Vedat Eyuboglu, and G. Forney Jr., "MMSE decision-feedback equalizers and coding. I. equalization results," *IEEE Trans. on Comm.*, vol. 43, no. 10, pp. 2582–2594, Oct. 1995.

[51] M. K. Varanasi and T. Guess, "Optimum decision feedback multiuser equalization with successive decoding achieves the total capacity of the Gaussian multiple-access channel," in *Proc. of 31st Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, CA, USA, Nov. 1997, pp. 1405–1409.

[52] P. W. Wolniansky, G. J. Foschini, G. D. Golden, and R. A. Valenzuela, "V-BLAST: an architecture for realizing very high data rates over the rich-scattering wireless channel," in *Proc. of URSI ISSSE*, Pisa, Italy, Sept. 1998.

[53] B. Hassibi, "An efficient square-root algorithm for BLAST," in *Proc. of IEEE ICASSP*, vol. 2, Istanbul, Turkey, June 2000, pp. 737–740.

[54] D. Wübben, R. Böhnke, J. Rinas, V. Kühn, and K.-D. Kammeyer, "Efficient algorithm for decoding layered space-time codes," *IEE Electronics Letters*, vol. 37, no. 22, pp. 1348–1350, Oct. 2001.

[55] D. Wübben, R. Böhnke, V. Kühn, and K.-D. Kammeyer, "MMSE extension of V-BLAST based on sorted QR decomposition," in *Proc. of IEEE VTC-Fall*, vol. 1, Orlando, FL, USA, Oct. 2003, pp. 508–512.

[56] P. Luethi, "VLSI circuits for MIMO preprocessing," Ph.D. dissertation, ETH Zürich, Switzerland, 2009.

[57] C. Windpassinger and R. F. H. Fischer, "Low-complexity near-maximum-likelihood detection and precoding for MIMO systems using lattice reduction," in *Proc. of IEEE Inf. Th. Workshop*, Paris, France, Mar. 2003, pp. 345–348.

[58] D. Wübben, R. Böhnke, V. Kühn, and K.-D. Kammeyer, "MMSE-based lattice-reduction for near-ML detection of MIMO systems," in *Proc. ITG/IEEE Workshop on Smart Antennas*, Munich, Germany, Mar. 2004, pp. 106–113.

[59] H. Yao and G. Wornell, "Lattice-reduction-aided detectors for MIMO communication systems," in *Proc. of IEEE GLOBECOM*, vol. 1, Taipei, Taiwan, Nov. 2002, pp. 424–428.

[60] V. Tarokh, N. Seshadri, and A. R. Calderbank, "Space-time codes for high data rate wireless communication: performance criterion and code construction," *IEEE Trans. on Inf. Th.*, vol. 44, no. 2, pp. 744–765, Mar. 1998.

[61] E. Biglieri, G. Taricco, and A. Tulino, "Performance of space-time codes for a large number of antennas," *IEEE Trans. on Inf. Th.*, vol. 48, no. 7, pp. 1794–1803, July 2002.

[62] Y. Jiang, X. Zheng, and J. Li, "Asymptotic performance analysis of V-BLAST," in *Proc. of IEEE GLOBECOM*, vol. 6, St. Louis, MO, USA, Dec. 2005, pp. 3882–3886.

[63] M. Taherzadeh, A. Mobasher, and A. K. Khandani, "LLL reduction achieves the receive diversity in MIMO decoding," *IEEE Trans. on Inf. Th.*, vol. 53, no. 12, pp. 4801–4805, Dec. 2007.

[64] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, "Factoring polynomials with rational coefficients," *Math. Annalen*, vol. 261, no. 4, pp. 515–534, Dec. 1982.

[65] M. Taherzadeh and A. K. Mahmoud, "On the limitations of the naive lattice decoding," in *Proc. of IEEE ISIT*, Nice, France, June 2007, pp. 201–204.

[66] Y. H. Gan and W. H. Mow, "Complex lattice reduction algorithms for low-complexity MIMO detection," in *Proc. IEEE GLOBECOM*, St. Louis, MO, USA, Nov. 2005, pp. 2953–2957.

[67] B. Steingrimsson, T. Luo, and K. M. Wong, "Soft quasi-maximum-likelihood detection for multiple-antenna wireless channels," *IEEE Trans. on Sig. Proc.*, vol. 51, no. 11, pp. 2710–2719, Nov. 2003.

[68] M. R. G. Butler and I. B. Collings, "A zero-forcing approximate log-likelihood receiver for MIMO bit-interleaved coded modulation," *IEEE Comm. Letters*, vol. 8, no. 2, pp. 1089–7798, Feb. 2004.

[69] D. Seethaler, G. Matz, and F. Hlawatsch, "An efficient MMSE-based demodulator for MIMO bit-interleaved coded modulation," in *Proc. of IEEE GLOBECOM*, vol. 4, Dallas, TX, USA, Nov. 2004, pp. 2455–2459.

[70] E. Biglieri, J. Proakis, and S. Shamai, "Fading channels: Information-theoretic and communications aspects," *IEEE Trans. on Inf. Th.*, vol. 44, no. 6, pp. 2619–2692, Oct. 1998.

[71] H. Bölcskei, D. Gesbert, and A. J. Paulraj, "On the capacity of OFDM-based spatial multiplexing systems," *IEEE Trans. on Comm.*, vol. 50, no. 2, pp. 225–234, Feb. 2002.

[72] L. Zheng and D. N. C. Tse, "Diversity and multiplexing: A fundamental tradeoff in multiple-antenna channels," *IEEE Trans. on Inf. Th.*, vol. 5, no. 49, pp. 1073–1096, May 2003.

[73] A. Dejonghe and L. Vandendorpe, "Turbo-equalization for multilevel modulation: an efficient low-complexity scheme," in *Proc. of IEEE ICC*, vol. 3, New York City, NY, USA, Apr. 2002, pp. 1863–1867.

[74] A. Tomasoni, M. Ferrari, D. Gatti, F. Osnato, and S. Bellini, "A low complexity turbo MMSE receiver for W-LAN MIMO systems," in *Proc. of IEEE ICC*, vol. 9, Istanbul, Turkey, June 2006, pp. 4119–4124.

[75] L. Boher, R. Rabineau, and M. Helard, "FPGA implementation of an iterative receiver for MIMO-OFDM systems," *IEEE Journal on Sel. Areas in Comm.*, vol. 26, no. 6, pp. 857–866, Aug. 2008.

[76] I. B. Collings, M. R. G. Buttler, and M. McKay, "Low complexity receiver design for MIMO bit-interleaved coded modulation," in *Proc. of IEEE 8th ISSSTA*, Sydney, Australia, Aug. 2004, pp. 12–16.

[77] J. Erfanian, S. Pasupathy, and G. Gulak, "Reduced complexity symbol detectors with parallel structures for ISI channels," *IEEE Trans. on Comm.*, vol. 42, pp. 1661–1671, Feb. 1994.

[78] S. ten Brink, "Convergence of iterative decoding," *IEEE Electronic Letters*, vol. 35, pp. 806–808, May 1999.

[79] S. Fateh, "VLSI implementation of soft-input soft-output MMSE parallel interference cancellation," Master's thesis, ETH Zurich, Integrated Systems Laboratory, Mar. 2009.

[80] A. Burg, S. Häne, D. Perels, P. Luethi, N. Felber, and W. Fichtner, "Algorithm and VLSI architecture for linear MMSE detection in MIMO-OFDM systems," in *Proc. of IEEE ISCAS*, Kos, Greece, May 2006, pp. 4102–4105.

[81] H. S. Kim, W. Zhu, J. Bhatia, K. Mohammed, A. Shah, and B. Daneshrad, "A practical, hardware friendly MMSE detector for MIMO-OFDM-based systems," *EURASIP Journal on Advanced Sig. Proc.*, vol. 2008, no. 2, pp. 1–14, Jan. 2008.

[82] S. Yoshizawa, Y. Yamauchi, and Y. Miyanaga, "A complete pipelined MMSE detection architecture in a 4x4 MIMO-OFDM receiver," in *Proc. of IEEE ISCAS*, Seattle, QA, USA, May 2008, pp. 2486–2489.

[83] S. Eberli, D. Cescato, and W. Fichtner, "Divide-and-conquer matrix inversion for linear MMSE detection in SDR MIMO receivers," in *Proc. of IEEE NORCHIP*, Trondheim, Norway, Nov. 2008, pp. 162–167.

[84] P. Luethi, C. Studer, S. Duetsch, E. Zgraggen, H. Kaeslin, N. Felber, and W. Fichtner, "Gram-schmidt-based QR decomposition for MIMO detection: Vlsi implementation and comparison," in *Proc. of IEEE APCCAS*, Macao, China, Nov. 2008, pp. 830–833.

[85] S. Eberli, "Application-specific processor for MIMO-OFDM software-defined radio," Ph.D. dissertation, ETH Zürich, Switzerland, 2009.

[86] C. Studer, P. Blösch, P. Friedli, and A. Burg, "Matrix decomposition architecture for MIMO systems: Design and implementation trade-offs," in *Proc. of 41th Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, CA, USA, Nov. 2007, pp. 1986–1990.

[87] G. H. Golub and C. F. van Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins Univ. Press, 1996.

[88] H. R. Schwarz, *Numerische Mathematik*. B. G. Teubner Stuttgart, 1997.

[89] R. Zimmermann, "Computer arithmetic: Principles, architectures, and VLSI design," *Lecture notes, Integrated Systems Laboratory, ETH Zurich*, Mar. 1999.

[90] U. Küçükkabak and A. Akkaş, "Design and implementation of reciprocal unit using table look-up and newton-raphson iteration," in *Proc. of Euromicro Symp. on Digital System Design*, Rennes, France, Aug. 2004, pp. 249–253.

[91] D. Chen, B. Zhou, Z. Guo, and P. Nilsson, "Design and implementation of reciprocal unit," in *Proc. of IEEE 48th MWSCAS*, Cincinnati, OH, USA, Aug. 2005, pp. 1318–1321.

[92] B. Gestner and D. V. Anderson, "Single Newton-Raphson iteration for integer-rounded divider for lattice reduction algorithms," in *Proc. of IEEE 51th MWSCAS*, Knoxville, TN, USA, Aug. 2008, pp. 966–969.

[93] M. Shabany and P. G. Gulak, "A 0.13 µm CMOS, 655 Mb/s 4×4 64-QAM k-best MIMO detector," in *Proc. of IEEE ISSCC*, San Francisco, CA, USA, 2009.

[94] M. Pohst, "On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications," *ACM SIGSAM Bulletin*, vol. 15, no. 1, pp. 37–44, Feb. 1981.

[95] U. Fincke and M. Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis," *Math. of Computation*, vol. 44, no. 170, pp. 463–471, Apr. 1985.

[96] W. H. Mow, "Maximum likelihood sequence estimation from the lattice viewpoint," Master's thesis, Chinese University of Hong Kong, Department. of Information Engineering, June 1991.

[97] ——, "Maximum likelihood sequence estimation from the lattice viewpoint," in *Proc. of ISITA*, vol. 1, Nov. 1992, pp. 127–131.

[98] E. Viterbo and E. Biglieri, "A universal decoding algorithm for lattice codes," *Colloque GRETSI*, vol. 14, pp. 611–614, Sept. 1993.

[99] C. P. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," *Math. Programming: Series A and B*, vol. 66, no. 2, pp. 181–191, Sept. 1994.

[100] E. Viterbo and J. Boutros, "A universal lattice code decoder for fading channels," *IEEE Trans. on Inf. Th.*, vol. 45, no. 5, pp. 1639–1642, July 1999.

[101] M. O. Damen, A. Chkeif, and J.-C. Belfiore, "Lattice code decoder for space-time codes," *IEEE Comm. Letters*, vol. 4, no. 5, pp. 161–163, May 2000.

[102] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, "Closest point search in lattices," *IEEE Trans. on Inf. Th.*, vol. 48, no. 8, pp. 2201–2214, Aug. 2002.

[103] A. Burg, M. Wenk, M. Zellweger, M. Wegmueller, N. Felber, and W. Fichtner, "VLSI implementation of the sphere decoding algorithm," in *Proc. of 30th ESSCIRC*, Leuven, Belgium, Sept. 2004, pp. 303–306.

[104] M. O. Damen, H. El Gamal, and G. Caire, "On maximum-likelihood detection and the search for the closest lattice point," *IEEE Trans. on Inf. Th.*, vol. 49, no. 10, pp. 2389–2402, Oct. 2003.

[105] B. Cerato, G. Masera, and E. Viterbo, "Decoding the Golden Code: A VLSI design," *IEEE Trans. on VLSI*, vol. 17, no. 1, pp. 156–160, Jan. 2009.

[106] K. Wong, C. Tsui, R. S. Cheng, and W. Mow, "A VLSI architecture of a K-Best lattice decoding algorithm for MIMO channels," in *Proc. of IEEE ISCAS*, vol. 3, Scottsdale, AZ, USA, May 2002, pp. 273–276.

[107] Z. Guo and P. Nilsson, "A 53.3 Mb/s 4x4 16-QAM MIMO decoder in 0.35-um CMOS," in *Proc. of IEEE ISCAS*, vol. 5, Kobe, Japan, May 2005, pp. 4947–4950.

[108] M. Wenk, M. Zellweger, A. Burg, N. Felber, and W. Fichtner, "K-best MIMO detection VLSI architectures achieving up to 424 Mbps," in *Proc. of IEEE ISCAS*, Kos, Greece, May 2006.

[109] S. Haykin, M. Sellathurai, Y. de Jong, and T. Willink, "Turbo-MIMO for wireless communications," *IEEE Comm. Magazine*, vol. 42, no. 10, pp. 48–53, Oct. 2004.

[110] P. Marsch, E. Zimmermann, and G. Fettweis, "Smart candidate adding: A new low-complexity approach towards near-capacity MIMO detection," in *Proc. of EUSIPCO*, Antalya, Turkey, Sept. 2005.

[111] L. G. Barbero and J. S. Thompson, "A fixed-complexity MIMO detector based on the complex sphere decoder," in *Proc. of IEEE 7th SPAWC*, Cannes, France, July 2006.

[112] J. Jaldén, L. G. Barbero, B. Ottersten, and J. S. Thompson, "Full diversity detection in MIMO systems with a fixed-complexity sphere decoder," in *Proc. of IEEE ICASSP*, vol. 3, Toulouse, France, Apr. 2006, pp. 49–52.

[113] L. G. Barbero and J. S. Thompson, "Rapid prototyping of a fixed-throughput sphere decoder for MIMO systems," in *Proc. of IEEE ICC*, vol. 7, Istanbul, Turkey, June 2006, pp. 3082–3087.

[114] C. Hess, M. Wenk, A. Burg, P. Luethi, C. Studer, N. Felber, and W. Fichtner, "Reduced-complexity MIMO detector with close-to ML error rate performance," in *Proc. of GLSVLSI*, Stresa-Lago Maggiore, Italy, Mar. 2007, pp. 200–203.

[115] L. Barbero, T. Ratnarajah, and C. Cowan, "A low-complexity soft-MIMO detector based on the fixed-complexity sphere decoder," in *Proc. of IEEE ICASSP*, Apr. 2008, pp. 2669–2672.

[116] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Num. Mathematik*. Mathematisch Centrum, Amsterdam, The Netherlands, 1959, vol. 1, pp. 269–271.

[117] T. Fukatani, R. Matsumoto, and T. Uyematsu, "Two methods for decreasing the computational complexity of the MIMO ML decoder," *IEICE Trans. Fundamentals*, vol. E87-A, no. 10, pp. 2571–2576, Oct. 2004.

[118] M. Stojnic, H. Vikalo, and B. Hassibi, "Further results on speeding up the sphere decoder," in *Proc. of IEEE ICASSP*, vol. 4, Toulouse, France, 2006, pp. 549–552.

[119] K. Su, "Efficient maximum likelihood detection for communication over multiple input multiple output channels," Ph.D. dissertation, University of Cambridge, United Kingdom, Feb. 2005.

[120] C. Studer, A. Burg, and W. Fichtner, "A unification of ML-optimal tree-search decoders," in *Proc. of 40th Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, CA, USA, Oct. 2006, pp. 2185–2189.

[121] F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM Journal Research and Development*, vol. 13, pp. 675–685, Nov. 1969.

[122] J. Hagenauer and C. Kuhn, "The list-sequential (LISS) algorithm and its application," *IEEE Trans. on Comm.*, vol. 55, no. 5, pp. 918–928, May 2007.

[123] A. D. Murugan, H. El Gamal, M. O. Damen, and G. Caire, "A unified framework for tree search decoding: Rediscovering the sequential decoder," *IEEE Trans. on Inf. Th.*, vol. 52, pp. 933–953, Mar. 2006.

[124] O. Damen, A. Chkeif, and J.-C. Belfiore, "Lattice code decoder for space-time codes," *IEEE Comm. Letters*, vol. 4, no. 5, pp. 161–163, May 2000.

[125] R. Wang and G. B. Giannakis, "Approaching MIMO channel capacity with soft detection based on hard sphere decoding," *IEEE Trans. on Comm.*, vol. 54, no. 4, pp. 587–590, Apr. 2006.

[126] J. Jaldén and B. Ottersten, "On the complexity of sphere decoding in digital communications," *IEEE Trans. on Sig. Proc.*, vol. 53, no. 4, pp. 1474–1484, Apr. 2005.

[127] M. Wenk, A. Burg, M. Zellweger, C. Studer, and W. Fichtner, "VLSI implementation of the list sphere algorithm," in *Proc. of 24th NORCHIP Conf.*, Linköping, Sweden, Nov. 2006, pp. 107–110.

[128] M. Stojnic, H. Vikalo, and B. Hassibi, "Speeding up the sphere decoder with H^∞ and SDP inspired lower bounds," *IEEE Trans. on Sig. Proc.*, vol. 56, no. 2, pp. 712–726, Feb. 2008.

[129] M. S. Yee, "Max-Log-Map sphere decoder," in *Proc. of IEEE ICASSP*, vol. 3, Philadelphia, PA, USA, Mar. 2005, pp. 1013–1016.

[130] D. Seethaler, H. Artés, and F. Hlawatsch, "Dynamic nulling-and-canceling for efficient near-ML decoding of MIMO systems," *IEEE Trans. on Sig. Proc.*, vol. 54, no. 12, pp. 4741–4752, Dec. 2006.

[131] S. W. Kim and K. P. Kim, "Log-likelihood-ratio-based detection ordering in V-BLAST," *IEEE Trans. on Comm.*, vol. 54, no. 2, pp. 302–307, Feb. 2006.

[132] M. Damen, H. Gamal, and G. Caire, "On maximum likelihood detection and the search for the closest lattice point," *IEEE Trans. on Inf. Th.*, vol. 49, no. 10, pp. 2389–2402, Oct. 2003.

[133] P. Luethi, A. Burg, S. Haene, D. Perels, N. Felber, and W. Fichtner, "VLSI implementation of a high-speed iterative sorted MMSE QR decomposition," in *Proc. of IEEE ISCAS*, New Orleans, LA, USA, May 2007, pp. 1421–1424.

[134] E. Zimmermann and G. Fettweis, "Unbiased MMSE tree search detection for multiple antenna systems," in *Proc. of WPMC*, San Diego, USA, September, 2006.

[135] D. Seethaler and H. Bölcskei, "Infinity-norm sphere-decoding," *to appear in IEEE Trans. on Inf. Th.*

[136] D. Seethaler, J. Jaldén, C. Studer, and H. Bölcskei, "Tail behavior of sphere-decoding complexity in random lattices," in *Proc. of ISIT*, Seoul, South Korea, June 2009.

[137] M. van Dijk, A. J. E. M. Janssen, and A. G. C. Koppelaar, "Correcting systematic mismatches in computed log-likelihood ratios," *European Trans. on Telecomm.*, vol. 14, no. 3, pp. 227–244, July 2003.

[138] A. Burg, M. Wenk, and W. Fichtner, "VLSI implementation of pipelined sphere decoding with early termination," in *Proc. of EUSIPCO*, Florence, Italy, September 2006.

[139] *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Channel Coding and Multiplexing Examples*, 3GPP Organizational Partners TS 25.944, Rev. 4.1.0, June 2001.

[140] J. J. Boutros, F. Boixadera, and C. Lamy, "Bit-interleaved coded modulations for multiple-input multiple-output channels," in *Proc. of IEEE 6th ISSSTA*, vol. 1, Newark, NJ, USA, Sept. 2000, pp. 123–126.

[141] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. on Comm.*, vol. 49, pp. 1727–1737, Oct. 2001.

[142] R. Shariat-Yazdi and T. Kwasniewski, "A multi-mode sphere detector architecture for WLAN applications," in *Proc. of IEEE Int. SOC Conf.*, Newport Beach, CA, USA, Sept. 2008, pp. 155–158.

[143] M. Wenk *et al.*, "VLSI implementation of high-throughput sphere decoding with low-complexity enumeration," in preparation.

[144] T. Richardson and R. Urbanke, *Modern Coding Theory*, 1st ed. Cambridge, United Kingdom: Cambdirge Univ. Press, 2008.

[145] P. Elias, "Coding for noisy channels," in *IRE Convention Record*. Part IV, 1955, pp. 37–46.

[146] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. on Inf. Th.*, vol. 13, no. 2, pp. 260–269, Apr. 1967.

[147] J. G. Proakis, *Digital Communications*, 4th ed. New York, USA: McGraw–Hill, 2001.

[148] R. E. Blahut, *Algebraic Codes for Data Transmission*. Cambridge, United Kingdom: Cambridge Univ. Press, 2003.

[149] R. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. on Inf. Th.*, vol. 9, no. 2, pp. 64–74, Apr. 1963.

[150] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. of IEEE GLOBECOM*, Nov. 1989, pp. 1680–1686.

[151] J. Hagenauer, "A soft-in/soft-out list sequential(LISS) decoder for turbo schemes," in *Proc. of IEEE ISIT*, Yokohama, Japan, June 2003, p. 382.

[152] N. Champaneria, T. K. Moon, and J. H. Gunther, "A soft-output stack algorithm," in *Proc. of 40th Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, USA, Oct. 2006, pp. 2195–2199.

[153] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. on Inf. Th.*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[154] J. Vogt and A. Finger, "Improving the max-log-MAP turbo decoder," *Electronics Letters*, vol. 36, no. 23, pp. 1937–1939, Nov. 2000.

[155] V. Franz and J. B. Anderson, "Concatenated decoding with a reduced-search BCJR algorithm," *IEEE Journal on Sel. Areas in Comm.*, vol. 16, no. 2, pp. 186–195, Feb. 1998.

[156] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, "VLSI architectures for metric normalization in the Viterbi algorithm," in *Proc. of IEEE ICC*, vol. 4, Atlanta, GA, USA, Apr. 1990, pp. 1723–1728.

[157] A. G. Dempster and M. D. Macleod, "Constant integer multiplication using minimum adders," in *IEE Proc. Circuits, Devices and Systems*, vol. 141, no. 5, Oct. 1994, pp. 407–413.

[158] V. Lefévre, "Multiplication by an integer constant," *INRIA Technical Report*, no. 4192, pp. 1–20, May 2001.

[159] R. G. Gallager, "Low density parity check codes," *Trans. of the IRE Professional Group on Inf. Th.*, vol. IT-8, pp. 21–28, Jan. 1962.

[160] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. on Inf. Th.*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[161] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. on Inf. Th.*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[162] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. on Inf. Th.*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.

[163] H. Zhong and T. Zhang, "Block-LDPC: a practical LDPC coding system design approach," *IEEE Trans. on Circuits and Systems I*, vol. 52, no. 4, pp. 766–775, Apr. 2005.

[164] Y. Sun, M. Karkooti, and J. R. Cavallaro, "High throughput, parallel, scalable LDPC encoder/decoder architecture for OFDM systems," in *Proc. of IEEE DCAS*, Dallas, TX, USA, Oct. 2006, pp. 39–42.

[165] K. K. Gunnam, G. S. Choi, W. Wang, E. Kim, and M. B. Yeary, "Decoding of quasi-cyclic LDPC codes using an on-the-fly computation," in *Proc. of 40th Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, USA, Oct. 2006, pp. 1192–1199.

[166] K. K. Gunnam, G. S. Choi, W. Wang, and M. B. Yeary, "Multi-rate layered decoder architecture for block LDPc codes of the IEEE 802.11n wireless standard," in *Proc. of IEEE ISCAS*, New Orleans, LA, USA, May 2007, pp. 1645–1648.

[167] *Digital Video Broadcasting (DVB) User guidelines for the second generation system for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)*, ETSI TR 102 376, Feb. 2005.

[168] S. Sharon, J. Litsyn, and J. Goldberger, "An efficient message-passing schedule for LDPC decoding," in *Proc. of 23rd IEEE*

Convention of Electrical and Electronics Engineers in Israel, Tel-Aviv, Israel, Sept. 2004, pp. 223–226.

[169] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE JSSC*, vol. 37, no. 3, pp. 404–412, Mar. 2002.

[170] X.-Y. Shih, C.-Z. Zhan, C.-H. Lin, and A.-Y. Wu, "An 8.29mm2 52mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13um CMOS process," *IEEE JSSC*, vol. 43, no. 3, pp. 672–683, Mar. 2008.

[171] S. Sharon, J. Litsyn, and J. Goldberger, "Efficient serial message-passing schedulers for LDPC decoding," *IEEE Trans. on Inf. Th.*, vol. 53, no. 11, pp. 4076–4091, Nov. 2007.

[172] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. on Comm.*, vol. 53, no. 7, pp. 1288–1299, Aug. 2005.

[173] C.-H. Liu, C.-C. Lin, H.-C. Chang, C.-Y. Lee, and Y. Hsua, "Multi-mode message passing switch networks applied for QC-LDPC decoder," in *Proc. of IEEE ISCAS*, Seattle, WA, USA, May 2008, pp. 752–755.

[174] M. Rovini, G. Gentile, F. Rossi, and L. Fanucci, "A scalable decoder architecture for IEEE 802.11n LDPC codes," in *Proc. of IEEE GLOBECOM*, Washington, DC, USA, Nov. 2007, pp. 3270–3274.

[175] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. of IEEE ICC*, vol. 2, Geneva, Switzerland, May 1993, pp. 1064–1070.

[176] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. on Comm.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.

[177] J. J. Boutros and G. Zemor, "On quasi-cyclic interleavers for parallel turbo codes," *IEEE Trans. on Inf. Th.*, pp. 1732–1739, Apr. 2006.

[178] A. Nimbalker, Y. Blankenship, B. Classon, and T. K. Blankenship, "ARP and QPP interleavers for LTE turbo coding," in *Proc. of IEEE WCNC*, Las Vegas, NV, USA, Mar. 2008, pp. 1032–1037.

[179] J. Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Trans. on Inf. Th.*, vol. 51, no. 1, pp. 101–119, Jan. 2005.

[180] B. Moision and J. Hamkins, "Coded modulation for the deep-space optical channel: Serially concatenated pulse-position modulation," *IPN Progess Report 41-161*, pp. 1–25, May 2005.

[181] G. Fettweiss and H. Meyr, "Parallel Viterbi algorithm implementation: breaking the ACS-bottleneck," *IEEE Trans. on Comm.*, vol. 37, no. 8, pp. 785–790, Aug. 1989.

[182] K. E. Batcher, "Sorting networks and their applications," in *Proc. of 32th AFIPS Spring Joint Computer Conf.*, Atlantic City, NJ, USA, 1968, pp. 307–314.

[183] S. Belfanti and S. Schläpfer, "Turbo decoder for 3GPP LTE," Semester Thesis, ETH Zurich, Integrated Systems Laboratory, Dec. 2008.

[184] D. Frank, R. Dennard, E. Nowak, P. Solomon, Y. Taur, and H.-S. Wong, "Device scaling limits of Si MOSFETs and their application dependencies," *Proc. of the IEEE*, vol. 89, no. 3, pp. 259–288, 2001.

[185] R. H. Dennard, J. Cai, and A. Kumar, "A perspective on today's scaling challenges and possible future directions," *Solid-State Electronics*, vol. 51, no. 4, pp. 518–525, Apr. 2007.

[186] M. Bohr, "A 30 year retrospective on Dennard's MOSFET scaling paper," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 11–23, Winter 2007.

[187] A. Burg, P. Belanovic, N. Felber, P. Luethi, C. Studer, and M. Wenk, "Algorithm assessment criteria for hardware implementation," MASCOT IST-026905 project deliverable D2.1.1, Tech. Rep., Dec. 2006.

[188] C. Studer, M. Wenk, and A. Burg, "MIMO transmission with residual transmit-RF impairments," *in preparation.*

[189] R. A. Horn and C. R. Johnson, *Matrix Analysis.* Cambridge Univ. Press, 1985.

Curriculum Vitae

Christoph Studer was born in Solothurn, Switzerland on December 25, 1979. He received the MSc degree in electrical engineering from the Eidgenössische Technische Hochschule (ETH) Zurich, Switzerland, in 2005, where he is currently working toward the Dr. sc. techn. degree. In 2005, he was a Visiting Researcher with the Smart Antennas Research Group of Prof. Dr. A. Paulraj, Stanford University, Stanford, CA, USA. Since 2006, he has been a Research Assistant with the Integrated Systems Laboratory (IIS) and the Communication Technology Laboratory (CTL) of ETH Zurich under the supervision of Prof. Dr. W. Fichtner and Prof. Dr. H. Bölcskei, respectively. His research interests include signal processing algorithms for wireless communications and the design of VLSI circuits and systems.

Mr. Studer was the recipient of an ETH Medal in 2005 for his Master's Thesis on "Sphere Decoding with Resource Constraints" and has won the Student Paper Contest of the 41th Asilomar Conference on Signals, Systems, and Computers in 2007 with the contribution "Matrix Decomposition Architecture for MIMO Systems: Design and Implementation Trade-Offs." In 2008, he was a winner of the Student Paper Contest of the IEEE International Symposium on Circuits and Systems with the paper entitled "VLSI Architecture for Data-Reduced Steering Matrix Feedback in MIMO Systems".