UNIVERSITY OF SOUTHAMPTON

Energy-Efficient Architectures for Multi-Gigabit MIMO Detection

by

Ibrahim Ahmad Bello

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Physical Sciences and Engineering

School of Electronics and Computer Science

July 2017

Academic Thesis: Declaration of Authorship

I, **Ibrahim Bello**, declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

Energy-Efficient Architectures for Multi-Gigabit MIMO Detection

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Either none of this work has been published before submission, or parts of this work have been published as listed in Section 1.5 of this thesis.

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Ibrahim Ahmad Bello

The use of multiple antennas in wireless transmission, otherwise known as multiple-input multiple-output (MIMO), is an important technique for achieving the high data-rates required by future communication systems. Already, MIMO technology has been adopted by the 3rd Generation Partnership Project Long Term Evolution, WiMAX and by recent Wireless Local Area Network standards, such as the IEEE 802.11ac. It is envisaged that multi-antenna systems will play an even more prominent role in future as more diverse platforms become interconnected and user data rates requirements increase. Although MIMO offers numerous advantages to communication systems, it also presents a number of challenges, in particular to the receiver, where the complexity of the signal detection is exacerbated by the interferences from the multiple transmit antennas. In the worst-case scenario, the signal detection in MIMO systems is an NP-hard problem, which makes its application to real-time systems impractical. As a result, low-complexity detection algorithms, with near-optimal performance, have been extensively studied in the literature in the past decade.

In this thesis, a number of detection techniques for MIMO systems will be investigated, with particular focus on achieving high throughput and low power consumption. We begin by presenting the VLSI implementation of the sphere decoder (SD), which achieves the optimal maximum likelihood bit error rate (BER) performance. Although the SD has the potential of achieving a high throughput - specifically in high signal-to-noise ratios (SNR) - it also suffers a severe throughput degradation at low SNR, which is undesirable in a real-time system. This problem motivates us to investigate the K-best algorithm, which delivers a constant throughput irrespective of the channel condition. Two architectures for the K-best detector are considered: single and multi-stage architectures. The latter case is particularly interesting as the multiple stages can be utilised to achieve deeply pipelined detectors, which is attractive for high-throughput applications. The proposed multi-stage K-best detector is implemented in a 65 nm CMOS process, and achieves a throughput of 3.29 Gbps and a power consumption of 580 mW for a 64-QAM 4×4 MIMO configuration, which compares favourably with recent implementations in the literature.

Contents

List of Figures

List of Tables

Abbreviations

ASM	Algorithmic state machine
ASMD	Algorithmic state machine with datapath
AWGN	Additive white Gaussian noise
BER	Bit error rate
CMOS	Complementary metal-oxide semiconductor
CORDIC	Coordinate rotation digital computer
FSD	Fixed-complexity sphere decoder
HDL	Hardware description language
HPED	Horizontal partial Euclidean distance
ICU	Interference cancellation unit
kGE	Kilo gate equivalent
MIMO	Multiple input multiple output
MIN	Minimum
ML	Maximum likelihood
MMSE	Minimum mean square error
MUX	Multiplexer
OFDM	Orthogonal frequency division multiplexing
ORVD	Orthogonal real-valued decomposition
PED	Partial Euclidean distance
QAM	Quadrature amplitude modulation
QRD	QR decomposition
RSV	Received signal vector
RVD	Real-valued decomposition
SE	Schnorr Euchner
SD	Sphere decoder
SQRD	Sorted QR decomposition
SINR	Signal to interference and noise ratio
SNR	Signal to noise ratio
TAR	Throughput to area ratio
V-BLAST	Vertical Bell Laboratories layered space time
VPED	Vertical partial Euclidean distance
ZF	Zero forcing

Nomenclature

$\mathbf{A}$	Matrix
$a_{i,j}$	Element at i_{th} row and j_{th} column of matrix $\mathbf{A}$
$\mathbf{H}$	Channel matrix
$\mathcal{S}$	Complex constellation set
$\mathcal{D}$	Real constellation set
$\langle . \rangle$	Expected Value
$\|.\|$	Euclidean norm
$\mathbf{I}_N$	N $\times$ N identity matrix
T_i	Partial Euclidean distance at the ith level
$T_{i,k}$	Partial Euclidean distance at the ith level for the kth path
$\Re\{.\}$	Real part of a complex number
$\Im\{.\}$	Imaginary part of a complex number
N_T	Number of transmit antennas
N_R	Number of receive antennas
N_0	Noise spectral density
σ_n^2	Noise variance
$\mathbf{A}^T$	Transpose of matrix $\mathbf{A}$
$\mathbf{A}^H$	Hermitian transpose of matrix $\mathbf{A}$
$\|x\|$	Absolute value of x
$\overline{x}$	Complement of x
$\|\mathbf{v}\|$	Number of elements in the set $\mathbf{v}$
λ	Wavelength

Acknowledgements

Firstly, I will like to thank my supervisors, Prof Mark Zwolinski and Dr Basel Halak, for the regular discussions, feedbacks and guidance, which I found invaluable during my research. I will also like to acknowledge Dr Mohammed El-Hajjar for his advice on numerous aspects of my research. Thanks also to my examiners, Professors Koushik Maharatna and Tughrul Arslan (University of Edinburgh) for their comments and recommendations, which helped to improve the quality of this thesis.

I will also like to acknowledge my colleagues in Zepler Lab 4 including Haider Abbas, Illani Mohd, Syafiq Mispan, Yue Lu and others for the discussions and exchange of ideas, which made the PhD journey much more interesting. I had many fruitful discussions with Dr Dwaipayan Biswas and I will like to thank him for his helpful suggestions.

I will like to use this opportunity to thank my parents for supporting me throughout my education and for motivating me to always aim for the best. Thanks also to my brothers and sisters who provided plentiful support and encouragement throughout my studies.

Finally, I will like to thank my wife, Aishah, for her numerous sacrifices and support which made the PhD so much easier. Thanks also to my son, Abdurrahman, who provided welcome distraction while I worked on my thesis.

The PhD was sponsored by the Petroleum Technology Development Fund, and their support is gratefully acknowledged. I will also like to acknowledge the use of the IRIDIS University of Southampton computer cluster, which was used in certain aspects of this work.

To my parents...

Chapter 1

Introduction

The field of wireless communications has experienced quite spectacular advancements in recent years. From humble beginnings in radio telegraphy and Morse codes, wireless communications is now an indispensable part of daily life, present in applications as diverse as mobile phones to domestic electrical installations. In the next few years, it is estimated that tens to hundreds of billions of devices will become interconnected [1]. This development has instigated a significant amount of research work in developing new standards and techniques to power the next generation communication devices.

One of the most noticeable trends in wireless communications is the exponential increase in the data rate as illustrated in Fig. 1.1. Over time, this has been achieved by switching from analogue to digital signal processing, increasing the channel bandwidth, adopting more sophisticated modulation schemes and more recently, by employing multiple antennas at the transmitter and receiver. The latter case, which is the focus of this thesis, is also known as multiple-input multiple-output (MIMO), and it allows higher transmission rates to be achieved without increasing the bandwidth [2]. A MIMO communication system is illustrated in Fig. 1.2 and compared with a conventional single-antenna system.

MIMO can be employed in three major ways: spatial-multiplexing, diversity and beamforming. In the *spatial-multiplexing* mode, the MIMO system aims to maximise the transmission rate by transmitting independent streams of data in parallel over multiple transmit antennas. In the absence of correlation between the parallel streams, the spatial-multiplexing MIMO system can increase the channel capacity in direct proportion to the number of transmit antennas [3]. Meanwhile, the MIMO system operating in the *diversity* mode aims to take advantage of the several antennas at the transmitter and receiver to improve the error rate of the transmission. The information bits are duplicated over the multiple antennas using one of a number of space-time block codes (STBC), such that multiple copies of the same data are available to the receiver, thereby, improving the performance. Diversity can be implemented both at the transmitter and receiver sides of the communication link; however, it is typically implemented at the transmitter since

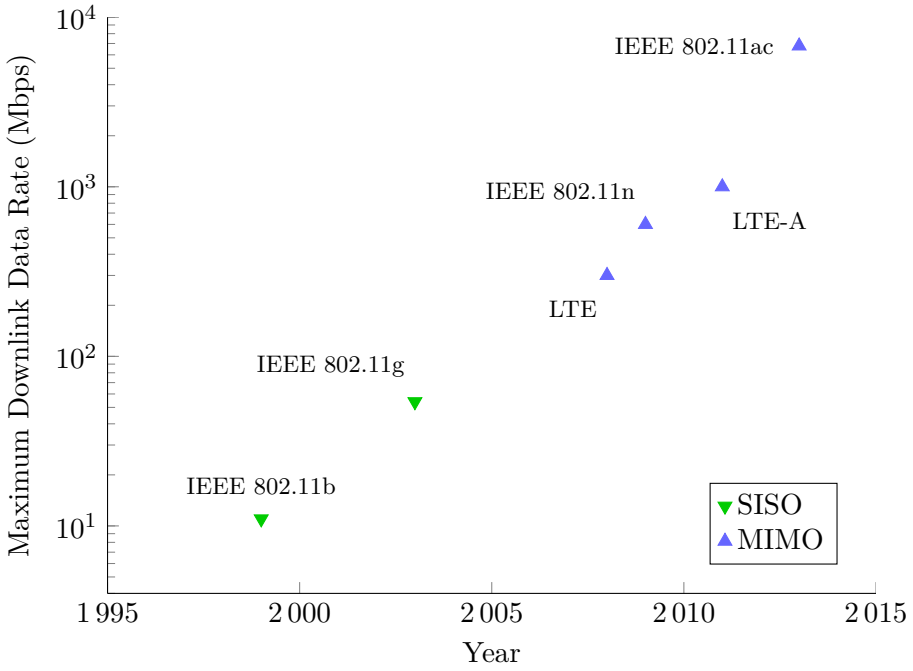


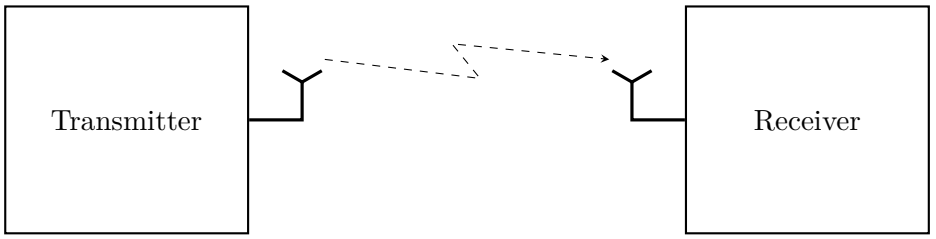
Figure 1.1: Wireless trends in the past 20 years

implementing diversity techniques at the receiver-side could result in highly complex remote units due to their physical and computational limitations [4]. MIMO can also be utilised to provide a beamforming gain in a multi-user scenario, where the gain of a transmitter or receiver is increased in the direction of the desired user while suppressing the interferences from other users [5]. MIMO can also be used in conjunction with orthogonal frequency division multiplexing (OFDM), to combat inter-symbol interference at the receiver by transmitting independent streams on a tone-by-tone basis [6]. In spite of these benefits, MIMO also complicates the wireless transceiver, which is explained in more detail in the next section.

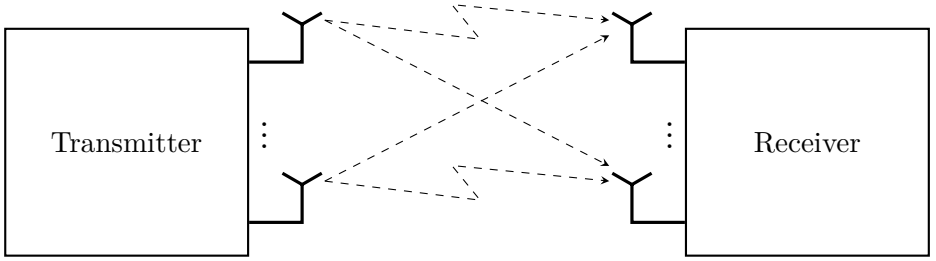
1.1 Challenges of Multi-Antenna Communication Systems

The use of multiple antennas at the transmitter and receiver presents a number of challenges as follows:

- **At the transmitter:** In spatial-multiplexing and diversity modes, the transmitter needs to transmit multiple data simultaneously over multiple antennas. In a multi-user scenario, the transmitter will also be equipped with user-selection capabilities based on certain criteria. All these impose additional complexity at the multi-antenna transmitter compared to a single-antenna transmitter.
- **At the receiver:** The signal detection at the receiver in a MIMO system is complicated as a result of interferences between the transmitted substreams on each



(a) Single-input single-output



(b) Multiple-input multiple-output

Figure 1.2: Illustration of single and multiple antenna systems

receive antenna. A straightforward detection using the maximum likelihood detector [7], results in an NP-hard problem, which is undesirable in a real-time communication system. This is particularly problematic when the receiver is a small hand-held device with limited power supply. This problem has inspired a significant amount of research into low-complexity alternatives to the maximum likelihood detector.

1.2 Energy-Efficient Communications

Energy efficiency refers to the ability of a digital circuit to meet its throughput target at a relatively low power consumption. Due to the increasing need for higher data rates, energy efficiency is expected to play a crucial role in future communication systems [8]. In this thesis, we will consider various strategies for achieving energy-efficient signal detection for MIMO communication systems. These design trade-offs are summarised in Fig. 1.3, which we elaborate as follows:

1. High throughput: Digital circuits that achieve a high throughput tend to be highly energy efficient, since they are operational over a short period, compared to slower digital circuits. However, a high throughput may also come at the expense of a higher power consumption.

2. Low power design techniques: These refer to circuit-level design techniques for reducing power consumption. Examples include well-established low power techniques such as power gating, clock gating and operand isolation. Optimising a

design for low power is a critical step in achieving highly energy-efficient digital circuits.

3. Low complexity: Complexity refers to different things depending on the context. In hardware design, this refers to the area consumption. A circuit with high complexity tends to incur a high power consumption due to the corresponding increase in the number of combinational and sequential elements. Thus, reducing the area of a design may lead to a more energy efficient system. Alternatively, complexity may refer to the number of operations required by an algorithm. This is usually measured in terms of floating-point operations per second (FLOPS). A circuit with a large number of FLOPS will tend to incur a high power consumption.

4. Error rate: This is the mismatch between the expected result of a computation and the *actual* output of a digital circuit. This is typically measured as the ratio of the number of correct information bits to the total number of bits transmitted. A relationship between the error rate and energy efficiency is less obvious than the previous three metrics. However, if a circuit does not meet its target error rate, this may increase the transmission power of the communication link [9]. Conversely, a system may deliberately aim for a lower error performance to reduce the power consumption.

1.3 Research Motivation

As already highlighted in Fig. 1.1, the data rates required by wireless communication systems have experienced an exponential growth in the past few decades. This phenomenon is expected to continue in the coming years with the introduction of 5G wireless systems. MIMO is a driving force of future wireless communication systems, and it is vital that the VLSI circuits for MIMO systems are designed as efficiently as possible to deliver the advantages promised by MIMO with high throughput and low power consumption. To achieve this, application specific integrated circuits (ASICs) are attractive as they have better potential for achieving low-power implementations compared to other platforms, such as field programmable gate arrays (FPGAs) [10] and digital signal processors (DSPs) [11].

In Section 1.1, the detector at the receiver-side was identified as a potential bottleneck in the MIMO communication system due to the high complexity required for detecting the transmitted data. Unfortunately, most of the existing VLSI implementations of MIMO detection in the literature do not achieve the high data-rates required by the next-generation communication systems, which now exceed the 1 Gbps mark [1]. Thus, more advanced circuit-level techniques are required to achieve the theoretic benefits promised by MIMO. The aim of this thesis is thus to fill this gap by presenting the

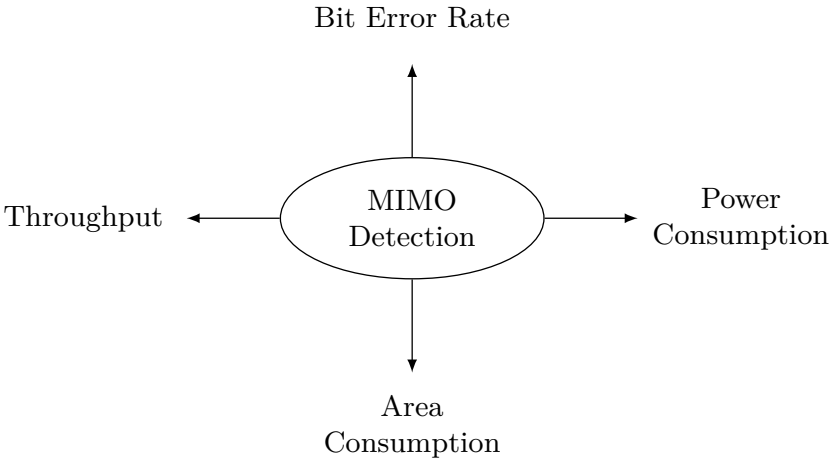


Figure 1.3: Design trade-offs for MIMO detection

VLSI implementations of different MIMO detector architectures and investigating numerous circuit-level strategies for achieving low complexity and high throughput MIMO detection.

1.4 Research Objectives

As highlighted in the previous sections, the signal detection in a MIMO receiver presents a major bottleneck in achieving the gains promised by MIMO technology. Several competing objectives come into play when implementing MIMO detection in hardware as illustrated in Fig. 1.3. Therefore, an efficient MIMO detector implementation should achieve a suitable trade-off among these objectives. This leads us to formulate a number of research questions as follows:

1. What is the impact of algorithm choice on the hardware implementation of a MIMO detector?
2. How do different architectural design choices of a particular algorithm impact the hardware implementation of the MIMO detector?
3. How do throughput optimisation strategies, such as pipelining and parallelism, compare with regards to achieving high data-rate MIMO detection?

To address these questions, we present a number of objectives for this thesis as follows:

1. Undertake a comprehensive survey of VLSI implementations of MIMO detection algorithms, which to the best of our knowledge is missing in the literature.

2. Investigate techniques for reducing the complexity of the sphere decoding algorithm [12] through critical path optimisations of the Euclidean distance computation. We will also implement a "look-ahead" strategy for reducing the number of clock cycles required by the sphere decoder for detecting the transmitted symbols.

3. Implement a high-speed sorting architecture, which will be applied to the K-best algorithm [13]. We will also study the impact of a reduced-complexity K-best algorithm on the BER performance compared with the original K-best algorithm.

4. Implement a fully-pipelined K-best architecture with a processing rate of 1 MIMO symbol per clock cycle. A novel pipeline scheduling technique will be implemented with the view of reducing the area consumption required by successive input signals. The fully-pipelined implementation will be compared with interleaved implementations (i.e. parallelism) of unpipelined K-best detectors.

5. Present a number of design guidelines for the hardware implementation of energy-efficient MIMO detectors, which will hopefully excite more research activity and innovations within this area.

1.5 Publications

The following papers have been published/submitted for publication as part of this research:

1. "A Survey of VLSI Implementations of Tree Search Algorithms for MIMO Detection," Circuits, Systems, and Signal Processing, 2015 [14].

2. "VLSI Implementation of a Scalable K-best MIMO Detector," in 2015 *15th International Symposium on Communications and Information Technologies (ISCIT)*, Oct. 2015 [15].

3. "VLSI Implementation of a Low-Complexity Look-Ahead Sphere Decoder," *submitted* to Transactions on Circuits and Systems II: Express Briefs.

1.6 Organisation of the Thesis

The thesis is organised as follows:

- **Chapter 2:** In this chapter, we tackle the first objective of the thesis as provided in Section 1.4. Firstly, the mathematical foundation for the MIMO communication

channel is presented. The complexity problem of the signal detection in a multi-antenna system is highlighted, and several low-complexity detection algorithms are presented. We pay particular attention to the class of detection algorithms that carry out the MIMO detection as a tree search, where each antenna corresponds with a level in the search-tree, and each branch of the tree corresponds to a possible solution. The chapter concludes by presenting a number of notable VLSI implementation results of tree-search detection algorithms from the literature.

- **Chapter 3:** This chapter addresses the second objective of the thesis. The sphere decoder is adopted for VLSI implementation due to its ML performance, which makes it useful in benchmarking other detection algorithms implemented in subsequent chapters of the thesis. We consider a number of strategies for improving the throughput of the sphere decoder, especially at low signal to noise ratios. We present and compare two implementations of the sphere decoder based on the conventional and proposed techniques for computing the partial Euclidean distance.

- **Chapter 4:** In this chapter, the implementation of a hybrid merge architecture is presented, which will be used in the K-best detector implementation, to meet the third objective of the thesis. Furthermore, a classification of K-best architectures is provided according to different criteria. The single-stage K-best architecture is selected due to its small area, and the VLSI implementation results are compared with that of the sphere decoder.

- **Chapter 5:** In this chapter, we tackle the fourth objective of the thesis by investigating pipelining as a technique for improving the throughput of the K-best detector. The pipeline schedules of a partial and fully-pipelined K-best detector are presented and analysed. A VLSI implementation of the fully-pipelined K-best detector is presented, and the results are compared with the interleaved single-stage K-best detector, as well as state-of-the-art implementations of K-best implementations from the literature. Finally, the potential application of the implemented fully-pipelined K-best architecture to recent high-throughput wireless communication standards is discussed.

- **Chapter 6:** The thesis is concluded in this chapter. A number of design guidelines are outlined, and possible topics are presented for future research.

Chapter 2

Background

2.1 Introduction

In Chapter 1, we provided an overview of MIMO technology and the impact it has had on wireless communications in the past few decades. In this chapter, we will provide a more in-depth discussion of MIMO technology, with particular focus on the signal detection at the receiver. The chapter is organised as follows. In Section 2.2, we will present the MIMO system model. In Section 2.3, we will provide an overview of the maximum likelihood MIMO detector. In Section 2.4, we will discuss linear detection algorithms. In Sections 2.5 and 2.6, we will present a number of non-linear techniques for improving the performance of linear detectors. In Section 2.7 a number of tree-search detection techniques will be presented. In Section 2.8, we will present a number of performance metrics for comparing hardware implementations of MIMO detectors. Furthermore, a number of notable results of tree-search VLSI implementations from the literature are provided. The chapter is concluded in Section 2.9.

2.2 System Model

Figure 2.1 illustrates a MIMO system employing N_T and N_R antennas at the transmitter and receiver respectively. A stream of bits, denoted by $\mathbf{x}$ is sent to a quadrature amplitude modulator (QAM), which maps Q bits to a QAM symbol, denoted by s, which is drawn from a constellation set of complex-valued symbols denoted by $\mathcal{S}$. The symbols for a 16-QAM scheme and their corresponding bit patterns are illustrated in Fig. 2.2. The QAM symbols are then demultiplexed over N_T transmit antennas to get an $N_T \times 1$ MIMO symbol, $\mathbf{s}$, using one of several MIMO encoding schemes [16]. The wireless channel can be represented by the $N_R \times N_T$ matrix, $\mathbf{H}$, whose elements represent the channel

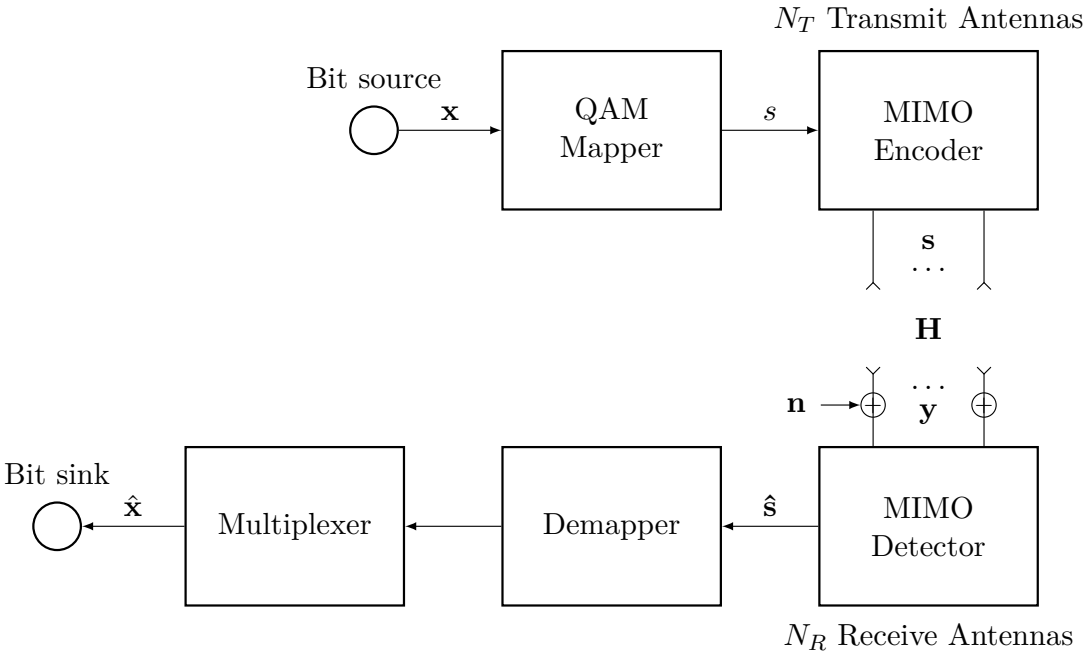


Figure 2.1: Block diagram of MIMO transmission and detection

fading coefficient for each path between the transmit and receive antennas. That is,

$$\mathbf{H} = [\mathbf{h}_1\, \mathbf{h}_2\, \ldots\, \mathbf{h}_{\mathrm{N_T}}] = \begin{bmatrix} h_{1,1} & h_{1,2} & \ldots & h_{1,N_T} \\ h_{2,1} & h_{2,2} & \ldots & h_{2,N_T} \\ \vdots & \vdots & \ddots & \vdots \\ h_{N_R,1} & h_{N_R,2} & \ldots & h_{N_R,N_T} \end{bmatrix}$$

where h_{ij} are complex-valued with normal distribution [3]. The ith column of the channel matrix, $\mathbf{h}_i$, represents the $N_R \times 1$ vector of channel coefficients from the ith transmit antenna to all the N_R receive antennas. If the transmitted substreams are sufficiently decorrelated (which is achieved by separating the transmit antennas by a distance of at least $\lambda/2$), then the entries of $\mathbf{H}$ can be considered to be independent and identically distributed, which allows the maximum channel capacity[1] to be achieved [17].

The $N_R \times 1$ received signal vector (RSV) at the receiver can be represented using the following linear equation:

$$\mathbf{y} = \mathbf{Hs} + \mathbf{n}, \tag{2.1}$$

where $\mathbf{n}$ is the $N_R \times 1$ additive white Gaussian noise (AWGN) experienced by the signal[2] at the receive antennas. At the receiver, a MIMO detector estimates the transmitted symbols by attempting to find the symbol, $\hat{\mathbf{s}}$, that minimises the error probability $P(\hat{\mathbf{s}} \neq \mathbf{s})$.

[1] Capacity, here, refers to the maximum data rate that the communication system can achieve at an arbitrarily small BER.

[2] The terms "signal" and "symbol" will be used frequently throughout the thesis. A symbol is scalar and is drawn from the complex constellation set, while a signal is a vector of complex numbers, obtained as a result of the attenuation and interferences experienced by the transmitted symbol vector in the channel.

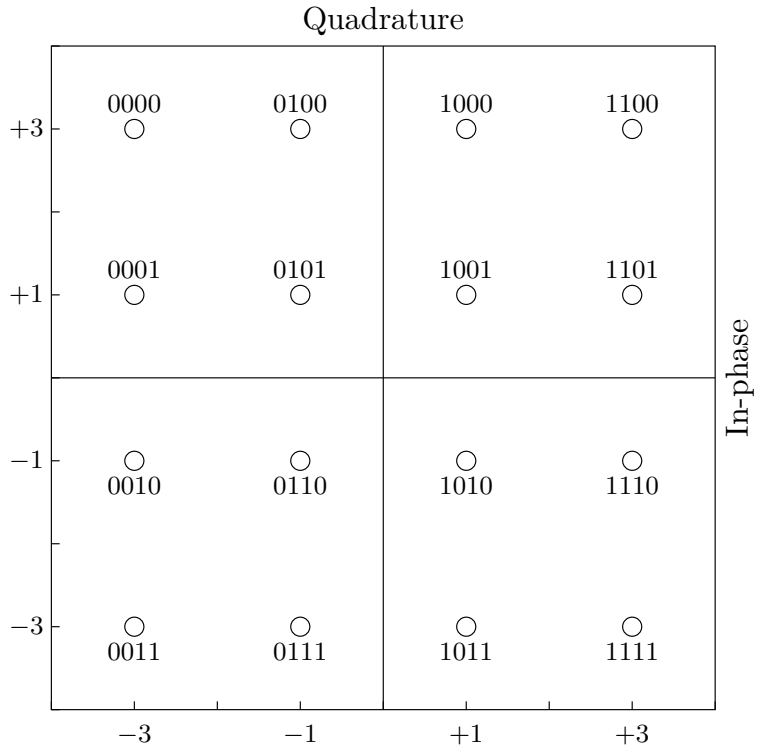


Figure 2.2: 16-QAM constellation points and corresponding binary representation

This procedure can be achieved in several ways as summarised in Fig. 2.3. After the signal has been detected, a demapper converts the symbols to their binary equivalent, according to the chosen modulation scheme, and a multiplexer converts the detected parallel bit streams into a single bit stream to recover the transmitted bits. A channel decoder may also be concatenated with the MIMO detector in a technique known as iterative decoding [18], in order to improve the BER. For the remainder of the thesis, we will assume an equal number of antennas at the transmitter and the receiver such that $N_T = N_R$.

2.3 MIMO Detection

In Chapter 1, we highlighted the problem of the signal detection in a multi-antenna system, which is complicated by the mutual interferences between the transmitted sub-streams at each receive antenna. Essentially, the received substream at each antenna will need to be detected in the presence of $N_T - 1$ interfering substreams. One approach of carrying out the signal detection is to consider all the possible transmitted symbol vectors as lattice points centred on the received signal, $\mathbf{y}$, as shown in Fig. 2.4. The detector exhaustively examines all the lattice points and determines the solution that minimises the Euclidean distance to the received signal. The maximum likelihood (ML) solution is obtained as follows:

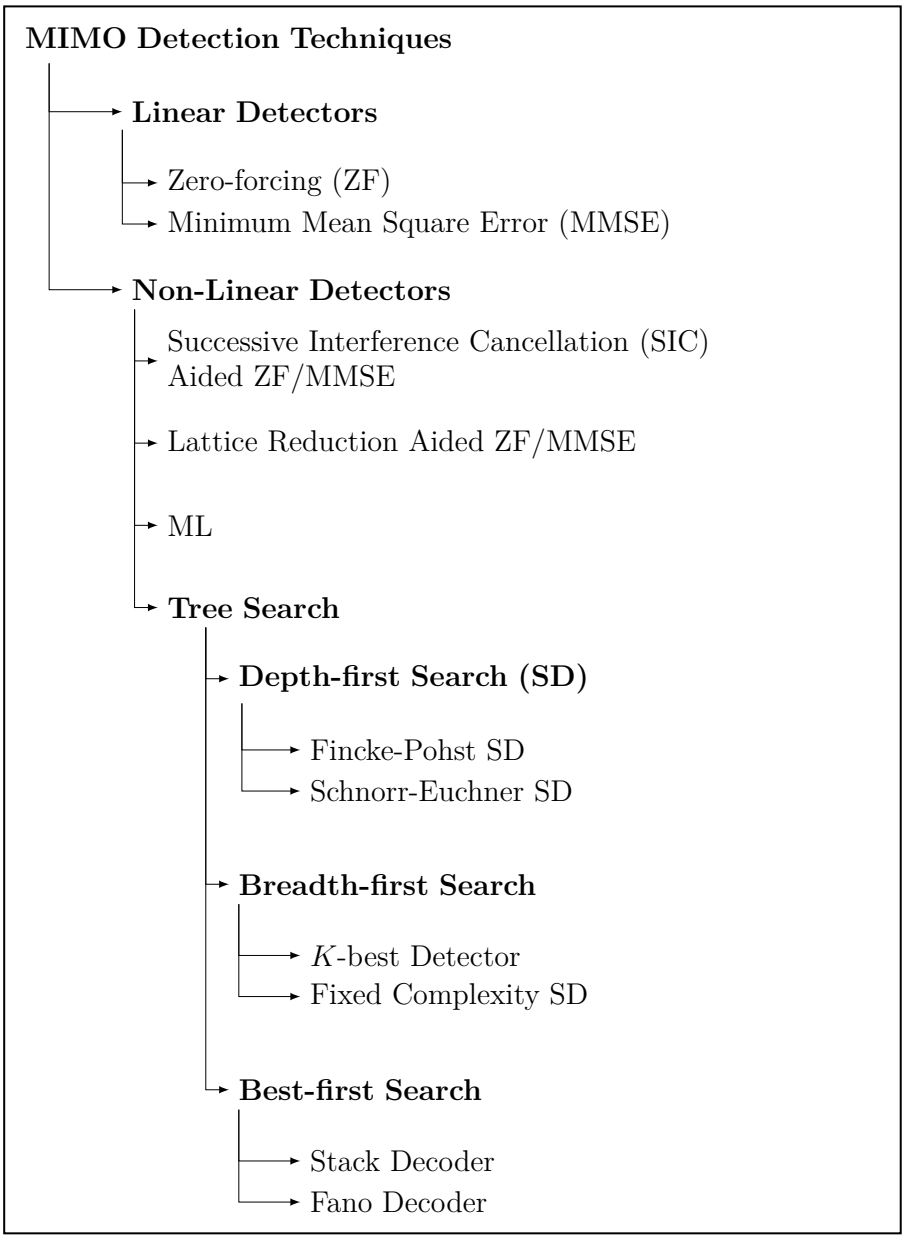


Figure 2.3: Classification of MIMO detection algorithms

$$\mathbf{s}_{\mathrm{ML}} = \underset{\mathbf{s}\in\mathcal{S}^{N_T}}{\arg\min} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2, \tag{2.2}$$

where $\mathcal{S}$ represents the complex QAM constellation set and $\mathcal{S}^{N_T}$ is an N_T dimensional lattice with complex entries formed from all possible combinations of the $N_T \times 1$ transmitted symbols. The total number of lattice points in the ML search is $|\mathcal{S}|^{N_T}$ which requires $N_T^2 + N_T$ complex multiplications per lattice point. Thus, the ML detector for a MIMO system employing binary phase shift keying (BPSK) modulation and 2 antennas will need to examine 4 lattice points and perform 12 complex multiplications, while $65,536$ lattice points and $393,216$ complex multiplications will be required in the case of 16-QAM with 4 antennas. This exponential complexity makes the ML detector

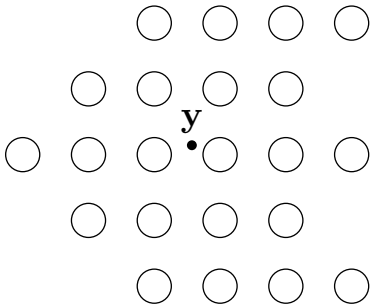


Figure 2.4: Lattice search for the maximum likelihood detector. Each of the circles represents a lattice point, **Hs**.

unsuitable for implementation in real-time systems, especially at larger MIMO dimensions. As such, a number of low-complexity alternatives to the ML detector have been investigated in recent years, which we will discuss in the subsequent sections.

2.4 Linear Detection Algorithms

Linear detectors apply a linear filter, $\mathbf{W}$, to the received signal, which suppresses the mutual interference on each of the receive antennas. The received signal is multiplied by the filter matrix, and the symbol is detected by carrying out a parallel decision on all the layers [19]. Linear detection can be performed by applying either the zero-forcing (ZF) or the minimum mean square error (MMSE) criterion and is discussed in the following sections.

2.4.1 Zero-forcing

The ZF detector suppresses the mutual interferences between all the layers (i.e. the N_T transmitted data substreams [19]). In this detector, the filter matrix is computed as the pseudoinverse of the channel matrix and the ZF estimate is obtained as follows:

$$\begin{aligned} \mathbf{W}_{\mathrm{ZF}} &= \mathbf{H}^{\dagger} = (\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H, \\ \tilde{\mathbf{s}}_{\mathrm{ZF}} &= \mathbf{W}_{\mathrm{ZF}}\mathbf{y}. \end{aligned} \tag{2.3}$$

The estimation errors of the different layers correspond to the main diagonal of the error covariance matrix given as follows [19]:

$$\Phi_{\mathrm{ZF}} = \mathrm{E}\{(\tilde{\mathbf{s}}_{\mathrm{ZF}} - \mathbf{s})\,(\tilde{\mathbf{s}}_{\mathrm{ZF}} - \mathbf{s})^H\} = \sigma_n^2 \left(\mathbf{H}^H\mathbf{H}\right)^{-1}. \tag{2.4}$$

The result of the multiplication in (2.3) is a vector of floating-point numbers, therefore, a quantiser is required in a final step to round off the ZF estimate to the nearest constellation point as follows:

$$\hat{\mathbf{s}}_{\mathrm{ZF}} = \mathcal{Q}\left(\tilde{\mathbf{s}}_{\mathrm{ZF}}\right),$$

where $\mathcal{Q}$ is the *slicing* function corresponding to the modulation scheme adopted for the data transmission. Where $N_T = N_R$ (i.e. when $\mathbf{H}$ is square), $\mathbf{W}_{\text{ZF}}$ can simply be computed as the ordinary inverse of $\mathbf{H}$, i.e.

$$\mathbf{W}_{\text{ZF}} = \mathbf{H}^{-1}, \tag{2.5}$$

which eliminates $2N_T^3$ complex multiplications and $2N_T^2(N_T - 1)$ complex additions,[3] compared to the pseudoinverse based computation in (2.3).

2.4.2 Minimum Mean Square Error

In the computation of the error covariance matrix in (2.4) for the ZF detector, it can be observed that small eigenvalues of the matrix $\mathbf{H}^H\mathbf{H}$ will result in large errors due to noise amplification, especially for large values of N_T [19]. The performance can be improved by incorporating the noise variance in the filter matrix, which results in a minimum mean square error criterion. The MMSE detection is carried out as follows:

$$\begin{aligned} \mathbf{W}_{\text{MMSE}} &= (\mathbf{H}^H\mathbf{H} + \sigma_n^2\mathbf{I}_{N_T})^{-1}\mathbf{H}^H \\ \tilde{s}_{\text{MMSE}} &= \mathbf{W}_{\text{MMSE}}\mathbf{y}, \end{aligned} \tag{2.6}$$

which results in a better estimate of the transmitted symbols. The estimation errors of the layers using the MMSE detector is given as follows:

$$\Phi_{\text{MMSE}} = \text{E}\{(\tilde{\mathbf{s}}_{\text{MMSE}} - \mathbf{s})\,(\tilde{\mathbf{s}}_{\text{MMSE}} - \mathbf{s})^H\} = \sigma_n^2\left(\mathbf{H}^H\mathbf{H} + \sigma^2\mathbf{I}_{N_T}\right)^{-1}.$$

Alternatively, the MMSE equalisation can be performed by substituting $\mathbf{H}$ in (2.3) with an $(N_T + N_R) \times N_T$ extended channel matrix $\underline{\mathbf{H}}$ as follows [19]:

$$\underline{\mathbf{H}} = \begin{bmatrix} \mathbf{H} \\ \sigma_n\mathbf{I}_{N_T} \end{bmatrix}.$$

The received signal vector is extended as follows:

$$\underline{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0}_{N_T,1} \end{bmatrix},$$

where $\mathbf{0}_{N_T,1}$ is a zero vector with N_T rows. Thus, the MMSE detection can be computed using (2.3), by simply replacing $\mathbf{H}$ and $\mathbf{y}$ with $\underline{\mathbf{H}}$ and $\underline{\mathbf{y}}$ respectively. The advantage of this approach is that the addition in (2.6) is eliminated and it is also convenient for the sorted QR decomposition MMSE detection described in [19].

[3] Two extra matrix multiplications are required if (2.3) is used to compute the ZF estimate compared to (2.5). Each matrix multiplication requires N_T^3 complex multiplications and $N_T^2(N_T - 1)$ complex additions.

Table 2.1: Numerical complexity of linear detectors versus the ML detector

Detector	**ZF**	**MMSE**	**ML**
$\mathbf{W}$	N_T^3	$3N_T^3$	-
$\mathbf{W} \cdot \mathbf{y}$	N_T^2	N_T^2	-
$\mathbf{H} \cdot \mathbf{s}$	-	-	N_T^2
$\|\mathbf{y} - \mathbf{H} \cdot \mathbf{s}\|^2$	-	-	$N_T^2 + N_T$
Total CMs	$N_T^3 + N_T^2$	$3N_T^3 + N_T^2$	$(N_T^2 + N_T)\,\|\mathcal{S}\|^{N_T}$

2.4.3 Complexity Analysis

The complexities of the ZF and MMSE detectors, in terms of the number of complex multiplications (CMs), are compared to the ML detector in Table 2.1. The number of CMs for the MMSE detector is determined according to (2.6). The Gauss-Jordan algorithm was considered for computing the matrix inverse, which has a polynomial complexity to the order of N_T^3 [20]. The slicing operation and minimum Euclidean search required by the linear detectors and MLD respectively have not been considered. It is clear from the table that linear detectors have a much lower complexity compared to the ML detector. Furthermore, their complexities are independent of the constellation size employed. For example, the number of CMs required at 4-QAM for 4×4 MIMO is 5120 for the ML detector and approximately 1.3×10^6 at 16-QAM. Meanwhile, the number of CMs required for the ZF and MMSE for 4×4 MIMO is 80 and 208 respectively, and remains the same irrespective of the modulation scheme employed.

The complexities of the linear detectors can also be reduced if the channel is relatively stationary. In this case, the computationally expensive matrix inverse operations need only be performed for $1/\tau$ of the time on average [21]. The ZF and MMSE detectors require N_T^3 and $3N_T^3$ CMs respectively for computing the filter matrix (channel-rate processing). N_T^2 CMs are required for the multiplication with $\mathbf{y}$ (i.e. symbol rate processing). If the channel is sampled at the rate of $1/\tau$, it implies that the total number of CMs required for the ZF and MMSE detectors are $N_T^3/\tau + N_T^2$ and $3N_T^3/\tau + N_T^2$ respectively. On the other hand, there is no possibility to separate the ML detector into channel and symbol-rate processing. Unfortunately, this complexity advantage enjoyed by the ZF and MMSE detectors comes at the cost of a reduced diversity order compared to the ML detector, which limits their usefulness in practice. Non-linear techniques achieve an improved diversity order compared to linear detection techniques and will be discussed in the subsequent sections.

Algorithm 1 Ordered SIC-aided ZF detection for V-BLAST spatial multiplexing

$\mathbf{r}^{(1)} \leftarrow \hat{\mathbf{y}}$
$\mathbf{H}^{(1)} \leftarrow \mathbf{H}$

for $k = 1$ **to** N_T **do**
  $\mathbf{W} \leftarrow \mathbf{H}^{(k)^\dagger}$
  $i \leftarrow \arg\min_i \|\mathbf{w}_i\|, 1 \leq i \leq N_T$, where substream i is yet to be detected
  $\hat{\mathbf{s}}_i \leftarrow \mathcal{Q}(\mathbf{w}_i^T \mathbf{r}_k)$
  $\mathbf{r}^{(k+1)} \leftarrow \mathbf{r}^{(k)} - \mathbf{h}_i \hat{\mathbf{s}}_i$
  $\mathbf{h}_i \leftarrow \mathbf{0}_{N_T \times 1}$
end for

return $\hat{\mathbf{s}}$

2.5 Successive Interference Cancellation

Linear detectors estimate the transmitted symbols by suppressing the interference on each of the layers. The performance can be improved by successively carrying out the detection layer by layer and cancelling out the interference of the already determined substreams on the currently detected symbol at every iteration. The successive interference cancellation algorithm (SIC) begins with the following initialisations:

$$\begin{aligned} \mathbf{r}^{(1)} &= \mathbf{y} \\ \mathbf{W} &= \mathbf{H}^\dagger, \end{aligned} \tag{2.7}$$

where the superscript indicates the current iteration of the algorithm. With this initialisation, the SIC-aided detection iteratively computes the signal at the kth iteration as follows:

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \mathbf{h}_k \hat{\mathbf{s}}_k, \tag{2.8}$$

where $\hat{\mathbf{s}}_k$ is obtained by quantising the product of the kth column of $\mathbf{W}$ and the received signal vector in the current iteration as follows:

$$\hat{\mathbf{s}}_k = \mathcal{Q}(\mathbf{w}_k^T \mathbf{r}_k), \tag{2.9}$$

where $\mathbf{w}_k$ is the kth column of $\mathbf{W}$ and $\mathcal{Q}(.)$ is the quantisation function, which rounds the soft estimate, $\hat{\mathbf{s}}_k$, to the nearest constellation point in $\mathcal{S}$. It should be noted that $\mathbf{w}_k$ is derived by taking the pseudoinverse of a deflated version of the channel matrix, $\mathbf{H}$, obtained by setting the kth column of $\mathbf{H}$ to zero. The MMSE criterion can also be used to compute the weighting vectors in (2.7) for improved performance.

Although SIC improves the performance of linear detectors, the error propagation from previously detected symbols affects the overall performance of the detection. Essentially,

if a symbol is erroneously detected in previous iterations, then this will affect the reliability of the subsequent detections. The SIC detection proposed by Wolniansky, Foschini, Golden, *et al.* [22] for the vertical Bell Laboratories layered space-time (V-BLAST) communication system mitigates this problem by carrying out the detection starting with the layer with the maximum post-detection SNR, which is defined as:

$$\rho_k = \frac{\left\langle |s_k|^2 \right\rangle}{\sigma_n^2 \|\mathbf{w}_k\|^2}. \tag{2.10}$$

Since the values for $\left\langle |s_k|^2 \right\rangle$ are not known prior, the optimal detection ordering can simply be estimated by detecting the symbols according to the ascending order of $\|\mathbf{w}_k\|^2$. An ordered SIC-aided ZF detection is illustrated in Algorithm 1.

Figure 2.5 provide a comparison of the performances of linear detectors with the MLD. The BER plot is simulated for 16-QAM with $N_T = 4$ based on 100, 000 randomly generated symbol vectors. The transmitter is assumed to employ spatial multiplexing in transmitting the symbol vectors over a Rayleigh fading channel. From the figure, it can be seen that the MMSE detector provides an improved performance compared to the ZF detector. Meanwhile, the use of V-BLAST ordering improves the BER in both cases. However, the performance still falls short of the diversity order delivered by the MLD. Another technique for improving the performance of linear detectors (as well as other sub-optimal algorithms) is lattice reduction, which is discussed in the next section.

2.6 Lattice Reduction

Although lattice reduction (LR) is originally applied in the field of linear algebra, it has also been successfully applied to MIMO detection [23]. If the matrix $\mathbf{H}$ is considered to be a *basis* for the lattice $\mathbf{Hs}$, then lattice reduction techniques can be applied to make $\mathbf{H}$ more orthogonal, which leads to better decision boundaries for the detection [24]. Linear detection techniques, as well as SIC-aided linear detectors, can then be applied to the reduced channel matrix to complete the detection.

The main operation of the LR-aided detector is to find a unimodular matrix, $\mathbf{T}$ (i.e. $\det(\mathbf{T}) = \pm 1$), such that $\widetilde{\mathbf{H}} = \mathbf{HT}$ where $\widetilde{\mathbf{H}}$ spans the same lattice as $\mathbf{H}$. This transforms the channel equation as follows [24]:

$$\begin{aligned} \mathbf{y} &= (\mathbf{HT})\mathbf{T}^{-1}\mathbf{s} + \mathbf{n} \\ \mathbf{y} &= (\widetilde{\mathbf{H}}\mathbf{T}^{-1})\mathbf{s} + \mathbf{n}. \end{aligned}$$

The ZF (or MMSE) linear detection is then computed with respect to $\widetilde{\mathbf{H}}$, rather than $\mathbf{H}$. After the output of the detector is quantised to the nearest constellation point, the detected symbol is computed as $\hat{\mathbf{s}} = \mathbf{T}\hat{\mathbf{z}}$, where $\hat{\mathbf{z}} = \mathcal{Q}(\widetilde{\mathbf{H}}^{-1}\mathbf{y})$.

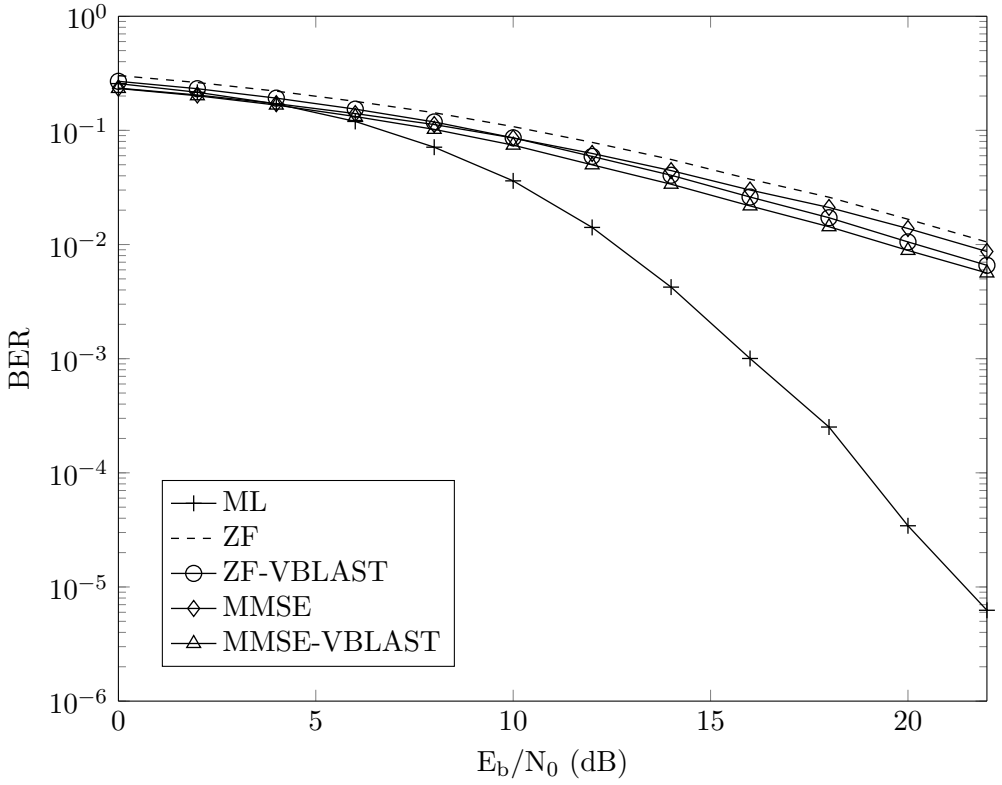


Figure 2.5: BER performance of ZF and MMSE detectors for 16-QAM and $N_T = 4$

Unfortunately, the computation of **T** for large matrix dimensions is NP hard, which results in a less than favourable detection complexity. Furthermore, the procedure is non-deterministic, which implies that different computation times will be required depending on the condition of the matrix. The most common method of finding the orthogonal matrix is using the Lenstra, Lenstra, Lovász (LLL) algorithm [25], which was originally formulated on real-valued lattices. This implies that a real-valued decomposition needs to be performed on the channel matrix [26]. A complex version of the LLL algorithm (CLLL) was proposed by Gan, Ling, and Mow [27], which implements the LLL algorithm on complex-valued lattices, which eliminates the need for an extra RVD step.

Like SIC, LR-aided detection has the attractive feature of having a complexity that is independent of the number of constellation points. Additionally, LR-aided detection is able to restore the full ML diversity order to the linear detectors [28]. However, LR has a variable complexity as the procedure is affected by the characteristics of the channel matrix which is random [29]. A particularly interesting result was by Shabany, Youssef, and Gulak [30], where a *fixed* complexity "hardware-optimized" CLLL (HOLLL) algorithm was realised. The implementation was able to achieve a fixed complexity by using a fixed number of iterations for its lattice reduction.

LR techniques can be used to improve the BER performance of the linear detectors and achieve ML diversity. However, the complexity is non-trivial, especially on larger matrix sizes [31]. The problem of a variable throughput has been resolved in [29] and [30] with

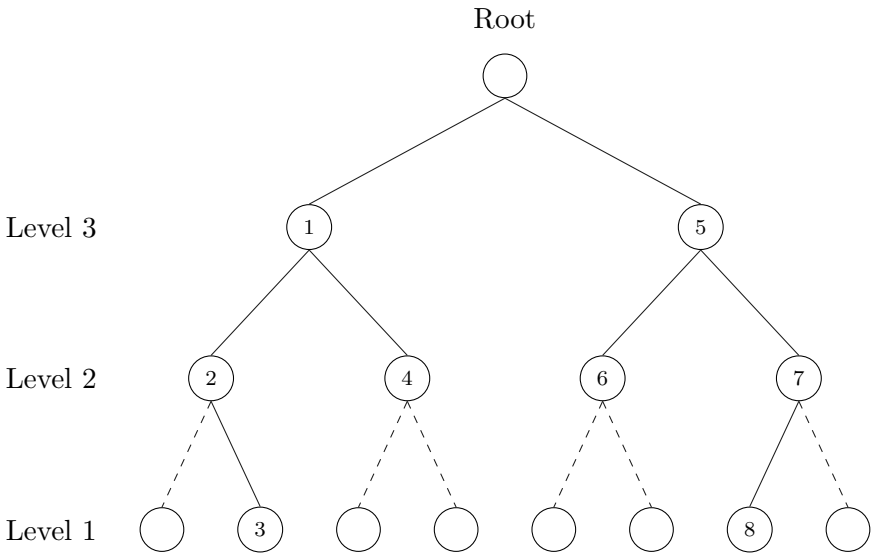


Figure 2.6: Depth first tree search for a MIMO system with $N_T = 3$

some penalty to the BER performance. The sphere decoder is only slightly more complex than the LR-aided linear detector and achieves a better BER performance at a given SNR especially at lower order modulations [32]. A background on the sphere decoder and similar algorithms is provided in the next section.

2.7 Tree-Search Detection Algorithms

Tree search (TS) algorithms refer to a wide range of MIMO detection techniques that carry out the signal detection successively from one layer to another in a procedure somewhat similar to SIC-aided linear detection. Like the ML detector, TS algorithms are search-based methods that apply a branch-and-bound operation [33] to every candidate solution by keeping or discarding it depending on certain criteria. Unlike the ML detector, however, only a subset of the candidates is considered which allows performance-complexity trade-offs to be achieved. In the next sections, some well-known TS algorithms and notable VLSI implementations from the literature are discussed.

2.7.1 Sphere Decoding

The channel equation presented in (2.1) can be simplified by carrying out a QR decomposition[4] on the channel matrix to transform the channel equation (2.1) as follows:

$$\begin{aligned} \mathbf{y} &= \mathbf{QRs} + \mathbf{n}, \\ \mathbf{Q}^H\mathbf{y} &= \mathbf{Rs} + \mathbf{Q}^H\mathbf{n}, \end{aligned} \tag{2.11}$$

where $\mathbf{Q}$ is an $N_T \times N_T$ unitary matrix and $\mathbf{R}$ is an $N_R \times N_T$ triangular matrix. Due to the unitary nature of $\mathbf{Q}$ (i.e. $\mathbf{Q}^H\mathbf{Q} = \mathbf{I}$), the norm, and consequently the statistics of $\mathbf{n}$, is unchanged by multiplication with $\mathbf{Q}^H$ and thus (2.11) can be rewritten as:

$$\mathbf{Q}^H\mathbf{y} = \mathbf{Rs} + \mathbf{n}. \tag{2.12}$$

For simplicity, $\mathbf{Q}^H\mathbf{y}$ shall be denoted by $\hat{\mathbf{y}}$, and will refer to the "received signal vector" in subsequent discussions.

The triangular property of $\mathbf{R}$ has interesting implications for the signal detection as only one symbol needs to be detected in each layer, while removing the interferences from previously detected symbols, unlike the ML detector where an $N_T \times 1$ symbol vector is considered at a time. This results in an M-ary tree with N_T levels as shown in Fig. 2.6. The sphere decoder (SD) invented by Viterbo and Boutros [12] carries out a depth-first search (DFS), where the detection proceeds from level N_T, corresponding with the channel entry r_{N_T,N_T}, down to level 1 corresponding with $r_{1,1}$. At the top level, the detector selects a symbol from the QAM constellation points and proceeds iteratively down the tree until the cumulative metric of the selected path violates a pre-determined constraint. With respect to the ML lattice search, the SD only considers those lattice points that lie within a given radius from the filtered received signal, $\hat{\mathbf{y}}$, as shown in shown in Fig. 2.7. This can be expressed mathematically as:

$$d(\mathbf{s}) = \|\hat{\mathbf{y}} - \mathbf{Rs}\|^2 < r^2, \tag{2.13}$$

where $d(\mathbf{s})$ represents the Euclidean distance between the symbol vector, $\mathbf{s}$, and $\hat{\mathbf{y}}$. The radius constraint can be computed using a simple linear detector such as zero-forcing or as a measure of the noise variance [35]. In the next sections, we will present some important concepts and equations related to the sphere decoder.

[4] A Cholesky decomposition which decomposes the channel matrix into the product of an upper triangular matrix and its complex conjugate (i.e. $\mathbf{H}^*\mathbf{H} = \mathbf{U}^*\mathbf{U}$) has also been applied to the SD [18]. However, the QR decomposition is attractive for hardware implementation as it can easily be realised using CORDIC rotations, which replace many complex arithmetic functions with shifts and additions [34]. A number of QR decomposition methods have been provided in Appendix A.

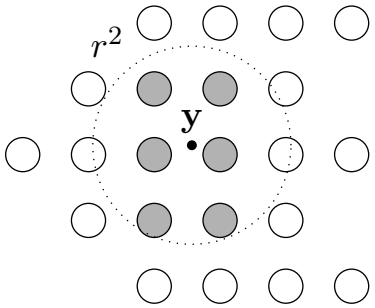


Figure 2.7: Modified lattice search using the sphere decoding algorithm.
Only the lattice points that fall within the radius are considered in the search.

2.7.1.1 Complexity of the Sphere Decoder

The complexity of the SD is directly related to the number of nodes that are visited to arrive at a solution, which determines the number of times the Euclidean distance operation is invoked. Unlike the ML and ZF/MMSE detectors, it is difficult to derive a closed-form expression for the number of visited nodes, since the SD algorithm is highly recursive and has different termination times depending on the channel condition. Since the SD is essentially a reduced-complexity ML algorithm, we can assume there exists a number $\gamma \in (0, 1]$, which reduces the problem size of the SD compared to the ML detector, such that the complexity of the SD can be expressed as follows [36]:

$$C(N_T) = |\mathcal{S}|^{\gamma N_T},$$

where the value of γ is a function of the statistics of the channel matrix and the noise variance.

In the worst case, the SD can be expected to approach the exponential complexity of the MLD, while in high SNR, and for a small number of antennas, the SD exhibits a low complexity, which competes favourably with polynomial-time algorithms [36]. This result implies that at low SNR, the lattice points are more closely clustered around the received signal, than at high SNR, necessitating a much larger radius to ensure that a solution is found. Fig. 2.8 shows the number of visited nodes for the SD for different antenna sizes.

2.7.1.2 Partial Euclidean Distance Computation

The sequential nature of the depth-first search ensures that the Euclidean distance (2.13) cannot be calculated at once, instead, it must be computed incrementally as the detector progresses deeper into the tree. The Euclidean distance up to any level in the tree is thus known as the partial Euclidean distance (PED). By traversing the tree from level

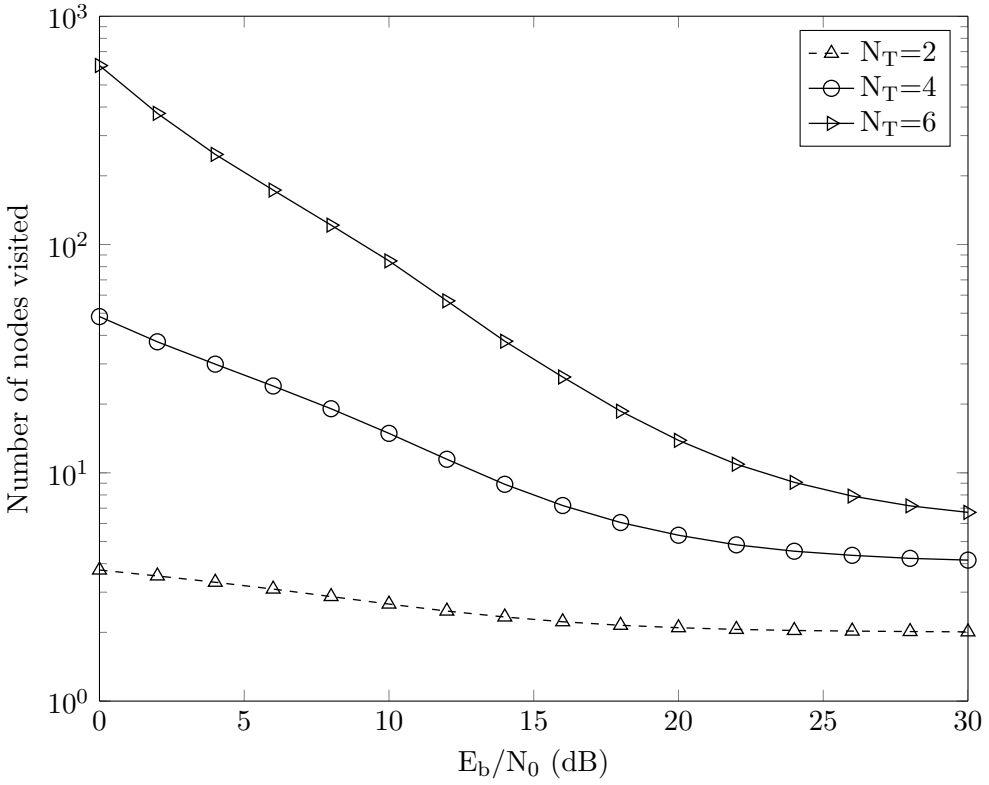


Figure 2.8: Number of visited nodes for the SD versus SNR using 16-QAM

$i = N_T$ to $i = 1$, the PED T_i, up to the i_{th} level is given as

$$T_i = T_{i+1} + |e_i|^2 , \tag{2.14}$$

where $|e_i|^2$ is the PED increment at the i_{th} level and is given by

$$|e_i|^2 = |b_i - r_{i,i}s_i|^2 , \tag{2.15}$$

where s_i represents a symbol at the i_{th} level and b_i is defined as

$$b_i = \hat{y}_i - \sum_{j=i+1}^{N_T} r_{i,j}s_j. \tag{2.16}$$

The summation term in (2.16) represents the interference of previously detected symbols which needs to be cancelled out from the currently detected symbol.

2.7.1.3 Real-valued Channel Decomposition

The original sphere decoder works on the basis of a complex constellation set and complex channel matrix. The channel equation described in (2.1) can also be decomposed

into the real and imaginary parts of the respective variables as follows [37]:

$$\begin{bmatrix} \Re\{\mathbf{y}\} \\ \Im\{\mathbf{y}\} \end{bmatrix} = \begin{bmatrix} \Re\{\mathbf{H}\} & -\Im\{\mathbf{H}\} \\ \Im\{\mathbf{H}\} & \Re\{\mathbf{H}\} \end{bmatrix} \begin{bmatrix} \Re\{\mathbf{s}\} \\ \Im\{\mathbf{s}\} \end{bmatrix} + \begin{bmatrix} \Re\{\mathbf{n}\} \\ \Im\{\mathbf{n}\} \end{bmatrix}, \tag{2.17}$$

where $\Re\{.\}$ and $\Im\{.\}$ denote the real and imaginary parts of a complex number respectively. This results in an equivalent tree with twice as many levels as the tree search based on the original complex-valued channel equation. Furthermore, the number of children per parent is reduced from M to $\sqrt{M}$ resulting in a new constellation set of odd-valued integer symbols, $\mathcal{D}$, which is defined as: $\{-\sqrt{M}+1, \ldots, \sqrt{M}-1\}$. It should be noted that as a result of the doubling of the tree depth, the summation in (2.16) now needs to be carried out up to $j = 2N_T$.

The conversion of the channel equation to a real model results in an attractive hardware implementation as it is generally easier to compute on real numbers than on complex numbers. The critical path of the multiplication in (2.16) is reduced from one real multiplier and adder in series to one real multiplier only, which results in a faster processing per node. However, the overall throughput of the real-valued detector suffers as a result of the longer time required to reach the leaf nodes.

2.7.1.4 Orthogonal Real-valued Decomposition

In the previous section, a real-valued decomposition of the channel equation was presented, which generates an equivalent tree detection to the complex channel model, but which allows for a simpler hardware realisation. A different real-valued channel decomposition was proposed by Azzam and Ayanoglu [38], where the complex channel matrix is decomposed as follows:

$$\bar{\mathbf{H}} = \begin{bmatrix} \Re\{h_{1,1}\} & -\Im\{h_{1,1}\} & \ldots & \Re\{h_{1,N_T}\} & -\Im\{h_{1,N_T}\} \\ \Im\{h_{1,1}\} & \Re\{h_{1,1}\} & \ldots & \Im\{h_{1,N_T}\} & \Re\{h_{1,N_T}\} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \Re\{h_{N_T,1}\} & -\Im\{h_{N_T,1}\} & \ldots & \Re\{h_{N_T,N_T}\} & -\Im\{h_{N_T,N_T}\} \\ \Im\{h_{N_T,1}\} & \Re\{h_{N_T,1}\} & \ldots & \Im\{h_{N_T,N_T}\} & \Re\{h_{N_T,N_T}\} \end{bmatrix}. \tag{2.18}$$

This new channel representation has the property that adjacent columns (i.e. $\widetilde{\mathbf{h}}_n$ and $\widetilde{\mathbf{h}}_{n+1}$) are orthogonal to each other, that is, $\widetilde{\mathbf{h}}_n \cdot \widetilde{\mathbf{h}}_{n+1}^T = 0$ for odd values of n (i.e. $1, 3, \ldots, 2N_T - 1$). It can also be shown that the QR decomposition of the modified channel matrix results in $\widetilde{r}_{i,i+1} = 0$ for all odd values of i. Thus, setting $\widetilde{r}_{i,i+1} s_{i+1} = 0$,

Table 2.2: Simplification of $r_{i,i} \times s_i$ for 16-QAM [40]

Computation	Equivalent
$r_{i,i} \times (+3)$	$r_{i,i} + (r_{i,i} << 1)$
$r_{i,i} \times (+1)$	$r_{i,i}$
$r_{i,i} \times (-3)$	$-r_{i,i} - (r_{i,i} << 1)$
$r_{i,i} \times (-1)$	$-r_{i,i}$

the computation of (2.16) at the i_{th} level is modified to

$$\begin{aligned}\widetilde{b}_i &= \hat{y}_i - \sum_{j=i+1}^{2N_T} \widetilde{r}_{i,j}\hat{s}_j \\ &= \hat{y}_i - \sum_{j=i+2}^{2N_T} \widetilde{r}_{i,j}\hat{s}_j.\end{aligned}$$

Meanwhile, $\widetilde{b}_{i+1}$ for the computation of the PED in the $(i+1)_{th}$ level is computed normally as

$$\widetilde{b}_{i+1} = \hat{y}_{i+1} - \sum_{j=i+2}^{2N_T} \widetilde{r}_{i+1,j}\hat{s}_j.$$

Thus, $|\widetilde{e}_i|^2$ for odd-valued levels can be computed concurrently with $|\widetilde{e}_{i+1}|^2$ since $\widetilde{b}_i$ no longer depends on the previously detected symbol, s_{i+1}. The PED up to the i_{th} level is then computed as

$$T_i = T_{i+2} + \left|\widetilde{b}_{i+1} - \widetilde{r}_{i+1,i+1}s_{i+1}\right|^2 + \left|\widetilde{b}_i - \widetilde{r}_{i,i}s_i\right|^2.$$

This result allows two adjacent levels to be processed concurrently, which can allow higher throughputs to be achieved.

Another consequence of the orthogonal real-valued decomposition (ORVD) is that the detection ordering is altered from detecting the symbols from the imaginary part, then the real part, to detecting the symbols alternately from the imaginary to the real part of each symbol. More precisely, the symbol ordering is altered from $\Im\{s_{N_T}\}\Im\{s_{N_T-1}\}\ldots\Im\{s_1\}\Re\{s_{N_T}\}\Re\{s_{N_T-1}\}\ldots\Re\{s_1\}$ to $\Im\{s_{N_T}\}\Re\{s_{N_T}\}\Im\{s_{N_T-1}\}\Re\{s_{N_T-1}\}\ldots\Im\{s_1\}\Re\{s_1\}$. Thus, the ORVD tree search generates a different set of intermediary results and PEDs from the conventional RVD. As a result, the ORVD incurs a performance degradation compared with the conventional RVD channel model [39].

2.7.1.5 Schnorr-Euchner Lattice Search

The SD is originally based on the Fincke-Pohst algorithm [41], which does not specify a particular order when visiting the lattice points. This can be inefficient, as the SD might

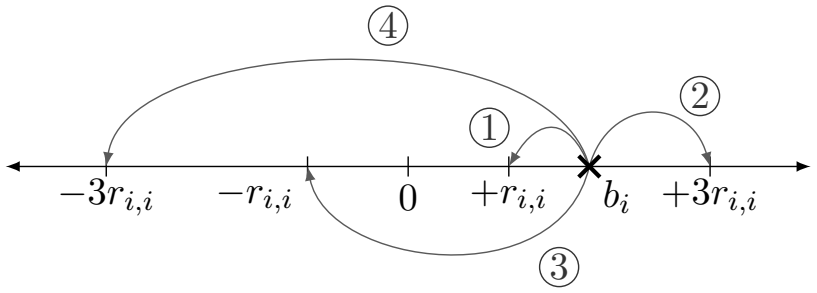


Figure 2.9: SE enumeration based on a real axis for 16-QAM. The encircled numbers indicate the child ordering.

spend too much time traversing non-promising paths. Furthermore, the Fincke-Pohst SD requires an initial starting point, called the Babai point [42], which results in added complexity. If the Babai point is not well selected, the Fincke-Pohst SD may fail to find a solution.

Schnorr and Euchner [43] proposed a modification to the Fincke-Pohst lattice search by visiting the nodes according to their path metrics, which enables the solution to be reached more quickly. Additionally, the Schnorr-Euchner (SE) solves the problem of detection failure by starting with an infinity radius, which is reduced any time a solution with a smaller metric is found.

The SE search visits the node according to their distance from a so-called SE centre, which is computed as follows:

$$c_i = b_i / r_{i,i}. \tag{2.19}$$

Thus, the PED increment for a given symbol level can be expressed alternatively as:

$$|e_i|^2 = |r_{i,i}\,(c_i - s_i)|^2\,, \tag{2.20}$$

The magnitude of the PED increment is directly proportional to the distance of the symbol from the SE centre. It should be noted that the PED does not have to be computed explicitly in order to determine the SE enumeration, instead, a zigzag search is carried out by iteratively determining the closest constellation points to the SE centre [44], which can be done by exploiting the geometrical relationship between the constellation points and the SE centre.

In hardware implementation however, the SE enumeration is typically determined based on (2.15) rather than (2.20) in order to avoid the costly division required in computing the SE centre. Considering a complex channel model, the starting point of the zigzag search, $s_i^{(0)}$, is determined as the symbol that minimises the phase difference to b_i as follows [45]:

$$s_i^{(0)} = \underset{s_i \in \mathcal{S}}{\arg\min}\, |arc(b_i) - arc(s_i)|\,,$$

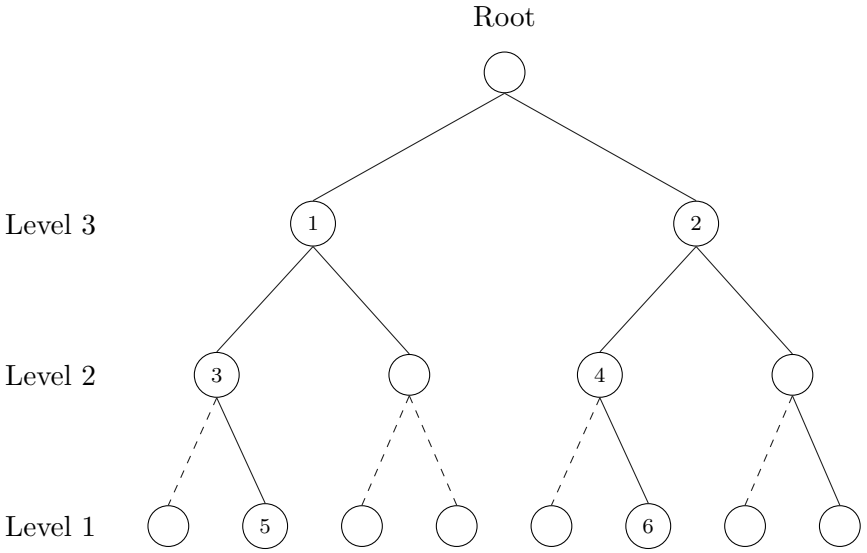


Figure 2.10: Breadth first tree search for a MIMO system using BPSK and $N_T = 3$

where $arc\,(.)$ computes the phase of a complex number. $arc\,(.)$ requires the computation of trigonometric functions, which is cumbersome in a hardware implementation. A real channel model results in a much simpler implementation as it dispenses with the computation of phase values. Furthermore, since s_i is drawn from an integer set in the case of the real channel model, the realization of $r_{i,i} \times s_i$ is quite simple as illustrated in Table 2.2. Figure 2.9 shows the SE enumeration for 16-QAM based on a real constellation set.

2.7.2 K-Best Algorithm

The sphere decoder presented in the previous section is highly recursive making it impossible to predict when a solution will be found. This makes it unsuitable for hardware implementation, especially in applications where a high throughput is desired. The tree search can also be carried out in a breadth-first manner where all sibling nodes are visited before descending to the subsequent levels. This is illustrated in Fig 2.10, where the numbers within the nodes indicate the order in which the nodes are visited. The breadth-first search is "forward-only", which results in a highly parallel and pipelineable architecture that is attractive for hardware implementation.

The most well-known breadth-first search is the K-best algorithm [13], which retains a fixed number of nodes, K, at each level, after a sorting operation. The K-best nodes are selected from a total of KM candidates or $K\sqrt{M}$ candidates if the real channel model is employed. Typically, no sorting is carried out at the top level, since K is typically greater than the number of candidates available. The K value provides a performance-complexity trade-off as larger K values approach the ML performance but incur a huge area cost, while smaller K values achieve a smaller area but with a penalty to the performance. In view of this, it is also possible to employ a non-constant K value for

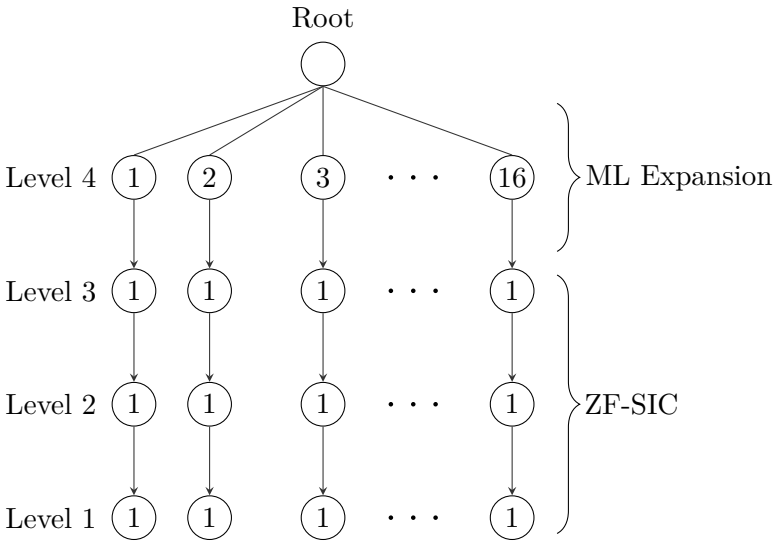


Figure 2.11: FSD tree search using 16-QAM and $N_T = 4$

the tree search, where large K values are employed at the upper levels, and smaller K values are employed towards the leaf levels, in order to achieve a smaller area [46], [47]. Since the probability of making an erroneous detection reduces with each level,[5] this does not have a significant effect on the performance.

2.7.3 Fixed-Complexity Sphere Decoder

The *fixed-complexity* sphere decoder (FSD) [48] is similar to the non-constant K-best detector as it expands a variable number of nodes from each level in its breadth-first detection. The FSD assigns a "node distribution" to the tree search, which determines the number of children that are extended from each parent node at each level. Typically, the FSD carries out an ML search at the topmost layer, that is $n_{N_T} = M$, where n_i is the number of nodes expanded in the i_{th} layer. The ML search at the topmost level ensures that the FSD does not miss the ML solution right at the beginning of the search, which is a pitfall of the K-best detector. The number of extended nodes is decreased in subsequent layers to satisfy the relation $n_{N_T} \geq n_{N_T-1} \geq \ldots \geq n_1$ [49]. For example, a node distribution of $(1, 1, 1, 16)^T$ implies that 16 nodes are expanded at the topmost layer while only a single node is expanded from each parent in the remaining layers as illustrated in Fig. 2.11. In subsequent layers, a simple decision feedback equalisation using a linear detector (such as zero-forcing) is carried out to extend a single node from each parent.

Since all the nodes are expanded in the topmost layer and only a simple linear detection is carried out in subsequent layers, the FSD is able to eliminate the sorting operation that

[5] Tree-search algorithms are quite similar to SIC-aided detection where the detection error is propagated from the upper to the lower levels. As such, the upper levels need to be detected stringently while this requirement can be relaxed at lower levels since there are fewer levels where the error could be propagated to.

is required in the conventional K-best detector [50]. Although the FSD was originally formulated for a complex constellation, it can also be used on real-valued constellations.

The FSD also introduces a novel channel ordering at the preprocessing stage, where the ML expansion at the topmost level is executed on the weakest substream, that is, the substream with the smallest post-detection SNR [22]. In subsequent layers, however, the linear detection is carried out on the substream with the largest post-detection SNR among the yet-to-be detected substreams, which is quite similar to ZF-SIC detection in V-BLAST systems [22]. In this respect, the FSD can also be considered to be a hybrid scheme combining ML and linear detection.

2.7.4 Best-First Search Algorithm

Unlike the DFS and BFS algorithms, the best-first search (BeFS) always extends along the path of the least-metric node irrespective of its level on the tree. The most popular implementation of the BeFS is the "stack" decoder [51], which maintains a sorted list for storing all the expanded nodes, and always extends the tree along the path of the node at the top of the list (i.e. the least-metric node). The search is terminated whenever a leaf node emerges on top of the list, and its path is presented as the ML estimate [52].

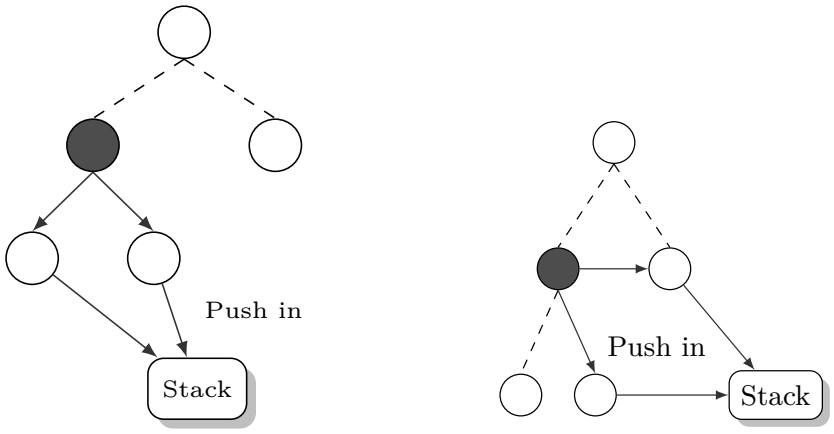
The need for a sorted list to store all the expanded nodes obviously makes the BeFS a memory-hungry algorithm, and a constraint is usually applied to the memory to reduce its complexity in hardware [52]. The detector may also spend too much time in the upper layers without reaching a leaf node under a given time constraint [53].

Fig. 2.12 illustrates the BeFS for a 2-ary tree. The current best node is fetched off the top of the stack and is replaced by all of its children. However, this may be expensive if M is large. Alternatively, a node may be replaced by its best child and best sibling as illustrated in Fig. 2.12(b) [54]. The modified BeFS reduces the complexity of the BeFS as only the PED of two nodes needs to be computed at a time to extend the tree.

Like the DFS, the BeFS has a variable complexity; however, it achieves a better worst-case and average complexity than the DFS [55]. A more detailed discussion of the BeFS is provided in [33], where it is indicated that the BeFS achieves the best performance-complexity trade-off among all the TS algorithms as it expands the fewest number of nodes on average.

2.7.5 Soft-Output Detection

Our discussion thus far has been focused on hard-output detection, where only a single solution is presented by the MIMO detector. This limits the performance of the detector



(a) Original best-first tree search (b) Modified best-first tree search

Figure 2.12: Best-first tree search [53]

as the performance depends on the "correctness" of the single hard-detection output and there is no possibility to explore other hypotheses.

Soft-output detection generates several other possible solutions (referred to as counter-hypotheses) apart from the hard-output solution. These soft outputs can then be used to generate the *a posteriori* reliability information for each bit position, expressed as a log-likelihood ratio (LLR), which is then passed to a channel decoder. For a given bit position i, the reliability information is expressed as [18]:

$$L_{\mathrm{D}}(x_i \,|\, \mathbf{y}) = \frac{P(x_i = 1 \,|\, \mathbf{y})}{P(x_i = 0 \,|\, \mathbf{y})}.$$

In each iteration, the soft-input soft-output (SISO) detector computes the probability, $L^D_{i,b}$, that the bth bit of the ith symbol in the output is a 1 or 0, given a channel observation, $\mathbf{y}$. The *a priori* reliability information computed by the channel decoder, $L^A_{i,b}$, in the previous iteration is fed back to the SISO detector to derive the new extrinsic information, $L^E_{i,b}$, which in turn is fed to the channel decoder. $L^D_{i,b}$ is expressed as a log likelihood ratio (LLR) and can be computed as [56]:

$$\begin{aligned} L^D_{i,b} \triangleq \min_{\mathbf{s}\in\mathcal{X}^0_{i,b}} \left\{ \frac{1}{N_0} \|\hat{\mathbf{y}} - \mathbf{R}\mathbf{s}\|^2 - \log P[\mathbf{s}] \right\} - \\ \min_{\mathbf{s}\in\mathcal{X}^1_{i,b}} \left\{ \frac{1}{N_0} \|\hat{\mathbf{y}} - \mathbf{R}\mathbf{s}\|^2 - \log P[\mathbf{s}] \right\}, \end{aligned} \tag{2.21}$$

where $\mathcal{X}^0_{i,b}$ and $\mathcal{X}^1_{i,b}$ are the sets of symbol vectors with the bth bit equal to 0 and 1 respectively, and $P[\mathbf{s}]$ is the *a priori* reliability information computed by the channel decoder. Computing (2.21) for every bit is computationally expensive, and the SD can be applied to reduce the complexity by considering only those $\mathbf{s}$ for which the PED is small [18]. These solutions are stored in a candidate list and the decoder computes the

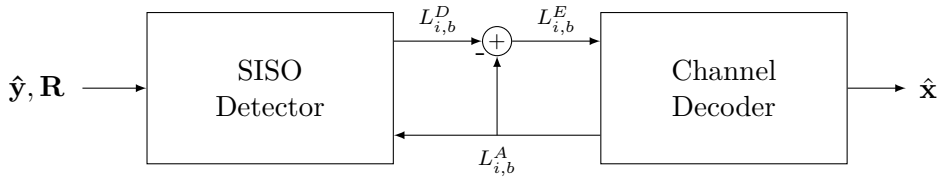


Figure 2.13: Block diagram of iterative MIMO detection

extrinsic information only for the solutions within that list. As such, the algorithm is referred to as the "list" sphere decoder (LSD).

A different strategy from the LSD is the single tree search (STS) proposed by C. Studer and H. Bolcskei [56], which does not require a list for storing the possible solutions. The algorithm computes the maximum *a posteriori* probability (MAP) solution, $\mathbf{s}^{\text{MAP}}$, and its bit-wise counter-hypotheses concurrently, in a single tree search. The MAP solution is given as

$$\mathbf{s}^{\text{MAP}} = \underset{\mathbf{s}\in\mathcal{S}^{N_T}}{\arg\min} \left\{ \frac{1}{N_0}\|\hat{\mathbf{y}} - \mathbf{R}\mathbf{s}\|^2 - \log P[\mathbf{s}] \right\}$$

and its corresponding reliability λ^{MAP} is computed as

$$\lambda^{\text{MAP}} = \frac{1}{N_0}\|\hat{\mathbf{y}} - \mathbf{R}\mathbf{s}^{\text{MAP}}\|^2 - \log P[\mathbf{s}^{\text{MAP}}].$$

One of the two minima in (2.21) corresponds to the MAP solution, as such, $L^D_{i,b}$ can be computed by determining $\mathbf{s}^{\text{MAP}}$, λ^{MAP} and its bit-wise counter-hypotheses $\lambda^{\overline{\text{MAP}}}$, which is computed as

$$\lambda^{\overline{\text{MAP}}} = \min_{\mathbf{s}\in\mathcal{X}^{\overline{MAP}}_{i,b}} \left\{ \frac{1}{N_0}\|\hat{\mathbf{y}} - \mathbf{R}\mathbf{s}\|^2 - \log P[\mathbf{s}] \right\},$$

where $\mathcal{X}^{\overline{MAP}}_{i,b} = \mathcal{X}^{\overline{x^{\text{MAP}}_{i,b}}}_{i,b}$. The STS-SD employs an efficient tree search strategy, where a node is traversed only once, which is achieved by descending into a sub-tree only if it would lead to an update to either λ^{MAP} or $\lambda^{\overline{\text{MAP}}}$.

The inclusion of the *a priori* information in the STS also modifies the SE enumeration such that the geometric properties of $\mathcal{S}$ can no longer be directly applied to determine the node with the smallest metric as described in Section 2.7.1.5. In this case, the metric of a node, $\mathcal{M}_P(s_i)$, comprises of two separate components: the channel-based PED denoted by $\mathcal{M}_C(s_i)$, and the *a priori* based metric, $\mathcal{M}_A(s_i)$, which is computed as

$$\mathcal{M}_A(s_i) = -\log P[s_i] \approx \sum_{b=1}^{Q} \frac{1}{2}(|L^A_{i,b}| - x_{i,b}L^A_{i,b}),$$

for $\left|L^A_{i,b}\right| > 2$ [57]. In [58], a hybrid enumeration is proposed, where two candidates (based on $\mathcal{M}_C$ and $\mathcal{M}_A$ respectively) are selected in each iteration, and the node with the smaller metric is selected for the next visit.

Due to the inclusion of the *a priori* information in the tree search, the STS achieves a better performance than the LSD, which only considers candidates around the ML solution for computing the extrinsic information. It also requires less area than the LSD as it does not require a candidate list. A more in-depth discussion on the STS is provided in [59] and [56].

2.8 Previous Work

Tree-search techniques have received plenty of research interest in recent years due to their excellent performance-complexity trade-off compared with other sub-optimal methods. In the following sections, we will present some notable VLSI implementation results of TS algorithms. We will begin with a brief discussion on technology scaling and the effect this has on the comparison of results from different implementations. We will then provide a number of metrics which will be used in comparing the implementations presented in this section.

2.8.1 Performance Metrics

The performance of digital circuits is typically assessed using the throughput, area and power consumption. Other performance metrics such as power-delay product and hardware efficiency can be derived from these three metrics. In many cases, the throughput is the most important consideration, since a device is only useful if it meets the data rate required by the target application. However, low power consumption is becoming increasingly important recently due to the proliferation of devices that need to run on a limited power supply for a long period of time. It must be stated that performance metrics, when considered in isolation, might not always present a complete picture of the performance of a circuit: for example, a serial architecture might have a much better energy efficiency than a pipelined architecture, but will only be useful for applications requiring a low data rate. Similarly, a circuit could achieve a high throughput at a power consumption that overwhelms the target device. In this section, we will present different performance metrics that will be used for comparing previous works in the literature, as well as the designs presented in the rest of the thesis.

2.8.1.1 Impact of Technology Scaling

Since different implementations employ different CMOS processes for their designs, a direct comparison of results will result in unfair conclusions. For example, an architecture implemented on 90 nm would be expected to be faster and smaller compared with the 180 nm implementation of the same architecture. The generalised scaling equations assume

that as transistor feature size reduces, the supply voltage, delay and power dissipation will change similarly [60]. However, as we approach submicron levels ($\leq$ 45 nm), leakage power becomes quite significant, which makes comparison with larger CMOS processes inaccurate. In [61], a set of scaling equations based on a lookup table, derived from SPICE simulations of an inverter chain, is presented. However, in this thesis, we generally compare designs using the same bulk type CMOS processes of 65 nm and above and therefore use the simpler generalised scaling equations. All the designs are scaled to the STMicroelectronics 65 nm process with a 1.05 V supply voltage. S is taken as the ratio of a given CMOS technology to the reference technology (i.e. 65 nm), and U denotes the ratio between a given supply voltage and the reference supply voltage (i.e. 1.05 V).

2.8.1.2 Area Consumption

The area consumption of a circuit is typically provided in metric units by synthesis tools; however, this does not take into account the technology in which the design is implemented. A more accurate comparison of different implementations is performed by expressing the area in terms of gate equivalent (GE), where one gate is taken as a 2-input drive-1 NAND gate. The gate equivalent of an implementation is expressed as the ratio of the area in metric units to the area occupied by one gate as follows:

$$\text{Area} = \frac{\text{Area of implementation}}{\text{Area of one 2-input NAND gate}} \text{ GE}$$

$$= \frac{\text{Area of implementation}}{1000 \times \text{Area of one 2-input NAND gate}} \text{ kGE}.$$

For the STMicroelectronics 65 nm "CORE65LPLVT" technology, 1 GE is equivalent to 2.08 μm^2.

2.8.1.3 Throughput

The throughput, Φ, of the MIMO detector is the rate at which one $N_T \times 1$ symbol vector is detected and is measured in megabits-per-second (Mbps). The throughput is directly proportional to the clock frequency, f_{clk}, which in turn is affected by the process technology. The throughput of a given implementation can be scaled to the reference technology as follows:

$$\Phi^{'} = S \times \Phi.$$

As an example, if a design achieves a throughput of 100 Mbps in 130 nm technology, its throughput if implemented on 65 nm will be approximately 200 Mbps according to the generalised scaling theory, given $S = 130/65 = 2$. Based on the throughput, we can define the hardware efficiency of a detector as the ratio of the throughput to the area

consumption (TAR). This metric provides a measure of how efficiently the hardware resources are utilised to meet a target throughput. A high TAR figure suggests that the implementation utilises the available hardware resources more efficiently than an implementation with a low TAR value.

2.8.1.4 Power Consumption

The power consumption of a digital circuit is the rate at which it utilises energy per unit of time and comprises the dynamic power, which relates to the switching activity of the circuit and static power, which refers to the power consumption due to short-circuit and leakage current. Thus, the average power consumption is determined as follows [62]:

$$\begin{aligned} \text{Power} &= \text{Power}_{\text{dynamic}} + \underbrace{\text{Power}_{\text{short-circuit}} + \text{Power}_{\text{leakage}}}_{\text{static}} \\ \text{Power} &= \alpha_{0\to1} C_L V_{\text{dd}}^2 f_{\text{clk}} + I_{\text{short-circuit}} V_{\text{dd}} + I_{\text{leakage}} V_{\text{dd}}, \end{aligned} \tag{2.22}$$

where $\alpha_{0\to1}$ captures the switching component of the power consumption, C_L represents the load capacitance, and $I_{\text{short-circuit}}$ and I_{leakage} represent the short-circuit and leakage currents respectively.

It is evident from the previous equations that power has a quadratic relationship with the supply voltage. We can therefore scale the power consumption to the reference technology as follows:

$$P' = \text{Power} \times \left(\frac{1}{U}\right)^2 \times \left(\frac{1}{S}\right). \tag{2.23}$$

This scaling is only approximate as it does not incorporate all the components of the power equation presented in (2.22). However, we can assume it provides a fair comparison of the power consumption since the dynamic power typically dominates the static components [63]. Based on the power consumption, we can also define the energy-efficiency, which is computed as $1/E_{\text{bit}}$, where E_{bit} is the energy required to detect one bit of data, which is computed as the ratio of the power consumption to the throughput (i.e. power-delay product). As such, the energy-efficiency provides us with the number of bits that can be detected per unit joule of energy. The normalised energy consumption scaled to the reference technology is given as follows [53]:

$$E'_{\text{bit}} = \left(\frac{\text{Power}}{\text{Throughput}}\right) \times \left(\frac{1}{U}\right)^2 \times \left(\frac{1}{S}\right). \tag{2.24}$$

The energy-efficiency provides a more accurate assessment of the performance of a design than the power consumption alone, since it also captures the speed of the design. Thus, if a design has a high power consumption, but a low E_{bit}, it implies the design is more energy-efficient compared to one with a low power consumption but relatively high E_{bit}.

2.8.2 VLSI Implementations of the Sphere Decoder

In this section, we will present notable implementation results of the sphere decoder from the literature. The first VLSI implementation of the sphere decoder in the open literature was by Burg, Borgmann, Wenk, *et al.* [45]. In this work, two detectors based on different methods for computing the PED were presented. In the first implementation (ASIC I), the PED was computed using the ℓ^2 norm i.e. according to (2.14), while in the second implementation (ASIC II), the PED was computed according to an ℓ^∞ approximation as follows:

$$T_i \approx \max\left(T_{i+1}, |e_i|\right), \tag{2.25}$$

which resulted in an area reduction of about 50% compared with the original ℓ^2-norm SD. Interestingly, this approximation only resulted in a 1.4 dB loss to the BER. Unfortunately, the implementation suffered a throughput degradation at low SNR due to the SNR-dependent complexity of the SD. An *early termination* strategy was proposed by the authors in another work [64], which places a maximum number of visited nodes on the SD with some penalty to the performance. Both ASIC I and ASIC II are based on a serial *one-node-per-cycle* (ONPC) architecture based on a single stage consisting of a metric computation unit, which computes the PED, and an enumeration unit, which determines the next node to visit according to the SE ordering. For a 4×4, 16-QAM system, ASIC II achieves a scaled throughput of 650 Mbps at a reference SNR of 20 dB, while ASIC I achieves a scaled throughput of 281 Mbps.

Borlenghi, Witte, Ascheid, *et al.* [65] implemented the first soft-input soft-output STS detector based on the ONPC architecture described by Burg, Borgmann, Wenk, *et al.* [45]. Using a convolutional channel code with 1/2 code rate, the STS detector achieves an SNR gain of about 5 dB compared to the hard-output SD for a target BER of 10^{-2} using two iterations. The implementation has three cores for 4-QAM, 16-QAM and 64-QAM. The 64-QAM implementation achieves a throughput of 132.9 Mbps using two iterations at an SNR of 24 dB, while it is capable of achieving a maximum throughput of more than 1 Gbps.

Yang, Tsai, Chang, *et al.* [66] adopt the use of a "table enumeration" [67], which stores precomputed SE orderings in a lookup table. In this method, the 16-QAM complex plane is divided into 64 sub-regions, and the ordering is determined based on the location of the SE centre as illustrated in Fig. 2.14. Due to the symmetry of the complex plane, only the orderings based on the first quadrant need to be stored in memory. The implementation achieves a throughput of up to 231 Mbps at high SNR.

Jenkal and Davis [68] implemented a deeply pipelined detector that is capable of processing multiple received signals in order to achieve a higher throughput. Each independent received signal is assigned a separate memory unit for storing the surviving nodes; however, the received signals share the same computation resources in a time-multiplexed

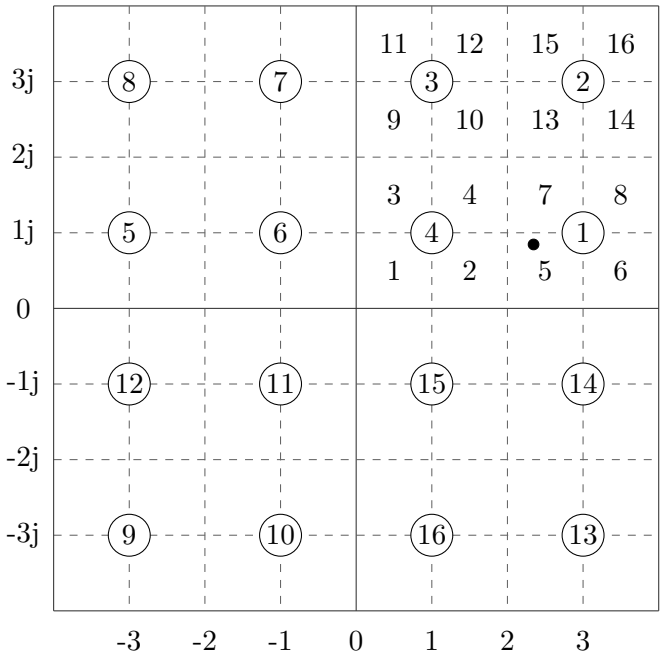


Figure 2.14: Illustration of tabular enumeration using 16-QAM

arrangement. The implementation achieves a throughput of 443 Mbps at 24 dB with an area consumption of 175 kGE.

2.8.3 VLSI Implementations of the K-Best Algorithm

The K-best algorithm, which provides an SNR-independent alternative to the sphere decoder, has featured quite prominently in the literature. The first VLSI implementation of a tree-search based algorithm in the open literature was by Wong, Tsui, Cheng, *et al.* [13], where a K-best detector for a 16-QAM 4×4 MIMO system was implemented. The implementation was based on a bubble-sort algorithm, which resulted in a relatively modest scaled throughput of 54 Mbps. Guo and Nilsson [69] implemented a similar detector to that of Wong, Tsui, Cheng, *et al.* [13], and achieve an improved throughput of 287 Mbps by using a smaller K value of 5 and other hardware-level optimisations. Certain PED operations are relegated to the preprocessing stage, which reduces the complexity of the detector unit compared to that of Wong, Tsui, Cheng, *et al.* [13].

Both Wong, Tsui, Cheng, *et al.* [13] and Guo and Nilsson [69] employ a bubble-sort unit, which requires several cycles to select the best K nodes. Wenk, Zellweger, Burg, *et al.* [37] implemented a single-cycle list merge, which merges the partially sorted children of each parent node in the search-tree, into one sorted list in a single step. The SE enumeration of the child nodes, for each parent, is determined in a zigzag manner based on the position of b_i on the real axis. The implementation achieves a throughput of 1.63 Gbps.

Shabany and Gulak [70] proposed a "distributed" sorting scheme that provides a compromise between the high-latency bubble-sort unit of Wong, Tsui, Cheng, *et al.* [13], and the low-latency, large-area single-cycle merge of Wenk, Zellweger, Burg, *et al.* [37]. This implementation is able to deliver K best nodes out of $K\sqrt{M}$ candidates in only K cycles. In the conventional K-best algorithm [13], all the children of the parent nodes from the previous level are expanded in parallel and sent to a bubble-sort unit. In this implementation, only the minimum-metric child of each parent node is expanded in a cycle, and the minimum amongst them is declared as the first node in the K-best list. Once a candidate is selected as part of the list, it is replaced by its next best sibling in the next cycle, and the process is repeated until all the K-best candidates are determined.

A hardware implementation of the distributed K-best detector is presented in [71], which achieves a scaled throughput of more than 1 Gbps. The implementation is extended in [72] to support soft-output generation by using the generated K best paths at the end of the detection to compute the LLR values. To improve the BER, selected discarded paths are also included in generating the soft outputs using ZF augmentation, which extends partial paths to full length by rounding them to the nearest constellation point [57], [69]. By using a convolutional turbo encoder, with 1/2 code rate, the soft-output implementation achieves an SNR gain of 2.9 dB at a BER of 10^{-3} compared to the hard-output detector.

Kim and Park [73] implemented a K-best detector based on the ORVD channel model [38], which allows adjacent levels to be processed simultaneously in a pipeline stage. The total number of pipeline stages in their implementation is reduced from 8 to 3, which leads to a small area consumption. Another contribution of this work is the use of an approximate sorting scheme, where only a subset of the children of the parent nodes is considered for the sorting, which is carried out in a distributed fashion.

Two VLSI implementations of the K-best are presented. The first implementation (KB-I) consists of a single K-best detector core, while the second implementation (KB-II) consists of 4 detector cores that are interleaved in order to increase the throughput. The single-core detector achieves a throughput of 404 Mbps, while the multi-core implementation improves the throughput by a factor of 4, with a corresponding increase in the area.

Liu, Ye, Ma, *et al.* [74] implemented a configurable K-best architecture that supports different number of antennas (2×2 up to 4×4) and modulation schemes (quadrature phase-shift keying (QPSK) up to 64-QAM). In this implementation, an "extension number" is formulated to determine the number of nodes that are to be extended from each parent node at a given level. More nodes are expanded from the more "reliable" parent nodes (i.e. nodes with smaller metrics) than from the less reliable nodes. A candidate generation unit calculates all the possible values of $r_{i,j}s_j$ in (2.16) and makes them available to all the candidates at a given level. In a block fading channel, the computation of

the $r_{i,j}s_j$ values can also be pushed back to the preprocessing unit and performed once per frame, which leads to further energy savings.

All the previously discussed implementations are based on a multi-stage architecture, where a PE is assigned to each level of the tree. Moezzi-Madani, Thorolfsson, and Davis [75] implemented a single-stage architecture where a single PE is used for all levels in a folded arrangement similar to the SD. A single-stage architecture is attractive for an application requiring moderate throughputs and where area consumption is premium. Similar to the SD, a higher throughput can be achieved by employing more than one detector core to operate in parallel on independent received symbol vectors. Their implementation supports antenna configurations of 2×2 up to 4×4, and a single core is able to achieve a throughput of 480 Mbps.

The K value contributes significantly to the complexity of the K-best detector. However, a lower complexity can be achieved by using smaller K values at lower levels, without significantly affecting the BER performance. Moezzi-Madani and Davis [46] implemented a modified K-best algorithm, which uses a non-constant K value that is decreased gradually at the lower levels. The implementation is able to save on area by up to 20%, while incurring a loss of 0.03 dB at an SNR of 20 dB compared to the conventional implementation. Another contribution of this work is a novel parallel merge algorithm (PMA) that is able to merge two sorted lists in one step, which makes it suited to high throughput applications. However, the area cost of the PMA is relatively high with a complexity of $O(N^2)$ [76]. The throughput of the PMA-based detector is 540 Mbps with an area consumption of 131 kGE.

Tsai et al. [47] implemented a non-constant K-best detector utilising the distributed sorting proposed by Shabany and Gulak [70]. Like Yang, Tsai, Chang, *et al.* [66], this is another work that implements the SE enumeration using a table lookup; however, a real constellation is considered in this case. Instead of deciding the SE ordering by finding the nearest constellation points to the SE centre (which requires a division step), this implementation decides the SE ordering through (2.16) by finding the closest $r_{i,i}s_i$ to b_i. The enumeration module in this implementation consists only of 3 adders/subtractors and one small table that is pre-calculated. A fully-pipelined version of this detector, which generates one MIMO symbol vector per clock cycle, was presented in [77].

2.8.4 VLSI Implementations of the Fixed-Complexity Sphere Decoder

The first hardware implementation of the FSD was by Barbero and Thompson [49], which was realised on an FPGA device for a 4×4 16-QAM system employing a node distribution of $(1, 1, 1, 16)^T$. The performance degradation of the implementation with respect to the ML at a BER of 10^{-3} is only 0.06 dB. The implementation achieves a throughput of 400 Mbps.

Liu, Lofgren, and Nilsson [78] implemented a modified FSD algorithm that uses extension numbers [74] to replace the ML search in the top layer of the tree with a reliability-based search, where more children are extended from the more reliable parent nodes. This results in an "imbalanced" tree expansion, where an unequal number of nodes are extended from each parent node. Similar to [74], this work employs the use of a candidate generation unit for precomputing the $r_{i,j}s_j$ values required to detect a symbol vector. In this case, an ORVD channel model (2.18) is employed, which allows the precomputed results to be shared by adjacent layers. The implementation achieves a throughput of 1.98 Gbps with an area consumption of 88.2 kGE.

Chen, He, and Ma [79] extended the imbalanced FSD architecture, proposed in [78], to support iterative decoding. Unlike the STS-SD enumeration [58], which uses 2 symbols (channel-based and *a priori*-based) in deciding the node for the next visit, this implementation derives an extra symbol that is derived from the *a priori*-based node, and is closest to the channel-based node, in order to get a better estimate of the node with the minimum $\mathcal{M}_P$. The implementation achieves a throughput of approximately 3 Gbps per iteration with an area consumption of 555 kGE.

2.8.5 VLSI Implementations of the Best-First Search Detector

Unlike the SD and breadth-first search algorithms, the best-first search has not received much attention in the literature. This might be due to the large memory required by the BeFS to store the visited nodes. Liao, Wang, and Chiueh [53] implemented a soft-output BeFS detector, which is capable of supporting QPSK up to 64-QAM modulations and 2×2 up to 8×8 antenna configurations. The stack is managed using a quad-dual-heap data structure [80], which reduces the complexity of identifying the best and worst nodes. This work also computes the PED incrementally, by distributing its computation over previous levels, which significantly reduces the critical path. The implementation achieves a maximum throughput of 863.6 Mbps in the 4×4, 64-QAM configuration.

Shen, Eltawil, Salama, *et al.* [81] implemented a soft/hard-output BeFS detector, which adopts features of depth-first and breadth-first search proposed by the authors in [54]. For any selected node, the detector enumerates to its best child and next best sibling and then extends along the path with the lower metric. Additionally, a Fano-like bias [82] is used to enable the detector to generate full solutions more quickly. The implementation achieves an average throughput of 199.8 Mbps for the hard-detection case, and an average throughput of 83.3 Mbps for the soft-detection case over the entire SNR range.

2.9 Summary and Conclusion

In this chapter, we have carried out an in-depth survey of MIMO detection algorithms. With respect to complexity, detection algorithms range from linear detectors, which carry out the detection by applying a linear filter to the received signal, to the maximum likelihood detector, which carries out an exhaustive search of all possible hypotheses. The high complexity of the ML detector limits its usefulness to applications requiring only small spectral efficiencies (e.g. 2×2, BPSK). Meanwhile, linear detectors suffer from a severe performance degradation at high SNR. This problem motivates us to explore alternative detection algorithms that achieve a good balance between performance and complexity. Tree-search algorithms, such as the sphere decoder, are very suitable candidates to fill this gap as they are able to achieve ML diversity with a significant reduction in complexity. In the next chapter, we will present VLSI implementation aspects of the sphere decoder, with focus on achieving both low complexity and high throughput.

Chapter 3

VLSI Implementation of the Sphere Decoder

3.1 Introduction

In the previous chapter, a number of low-complexity alternatives to the ML detector were presented. We paid particular attention to tree-search detection algorithms, which are attractive for implementing MIMO detection in hardware due to their excellent performance-complexity trade-offs. In particular, the sphere decoder has received a lot of research interest as it achieves the ML performance while incurring a similar complexity, in terms of number of numerical operations, to linear detection schemes at high SNR. As a result, the sphere decoder provides a more practical means of assessing the performances of other detection algorithms rather than the exponential-complexity ML detector.

In this chapter, the VLSI implementation of the sphere decoding algorithm is presented. The main objectives of the chapter are as follows:

1. Investigate techniques for reducing the computational complexity of the sphere decoder. The use of a novel look-ahead tree search and adaptive runtime constraints to improve the worst-case complexity of the SD will be investigated.

2. Present the ASIC implementation of the proposed sphere decoder. Several circuit-level techniques for reducing the complexity of the computational units of the SD will be presented.

The chapter is organised as follows. In Chapter 3.2, the methodology employed for the ASIC implementations proposed in this chapter, and the rest of the thesis, is presented. In Chapter 3.3, we evaluate the impact of runtime constraints on the complexity of the

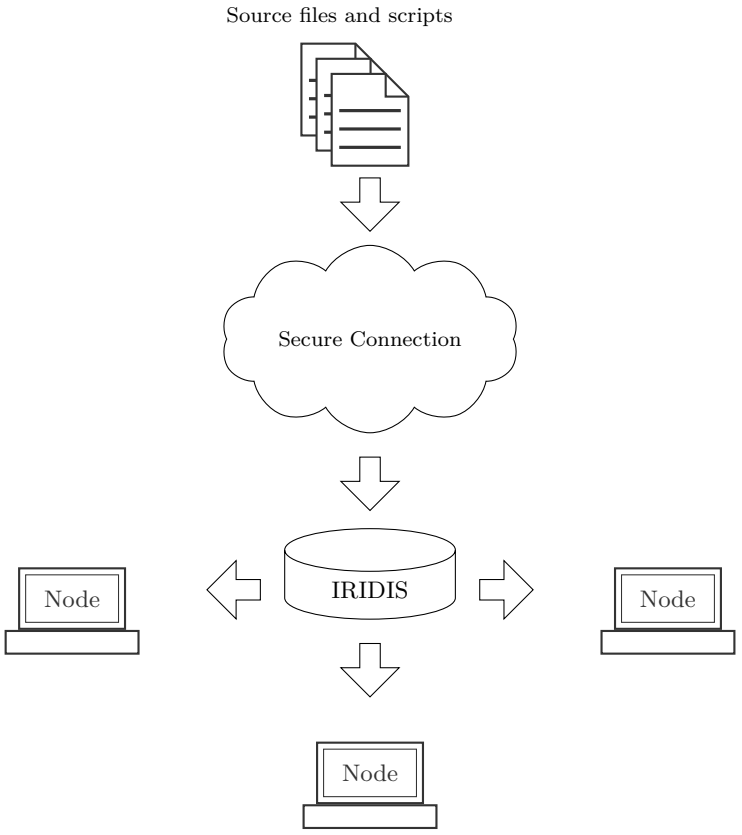


Figure 3.1: High performance computing setup using the IRIDIS computer cluster

sphere decoder. In Section 3.4, we present the architecture of the proposed sphere decoder. We also present a novel "look-ahead" strategy, which reduces the average number of clock cycles required to process a received signal. In Section 3.5, we present the hardware implementation results of the sphere decoder and compare with notable results of the SD from the literature. The chapter is concluded in Section 3.6.

3.2 Methodology

In this section, the steps taken to implement the sphere decoder in hardware from a high-level description in MATLAB to a placed-and-routed layout will be described. We will also provide an overview of how relevant implementation results, such as power consumption and bit error rate, are obtained. The same methodology will be used for the designs in Chapters 4 and 5.

3.2.1 High Level Analysis

The first step in the hardware implementation of the sphere decoder, and other MIMO detection algorithms, is a functional verification using a high-level programming language such as MATLAB. In order to achieve accurate results, several channel matrices and symbol vectors need to be generated. For this thesis, a Rayleigh channel is considered, with the assumption that the transmit antennas are sufficiently set apart to prevent the deleterious effects of correlation on the channel capacity at the receiver. At the receiver, additive white Gaussian noise (AWGN) is imposed on the incoming signal vectors. Unless otherwise specified, a total of $100,000$ symbol vectors are generated for each SNR point, with the assumption that the channel remains stationary for the duration of 4 symbol vectors. The received signal vectors are assumed to have been filtered by multiplication with $\mathbf{Q}^H$ according to (2.11). For a 64-QAM, 4×4 system this is equivalent to 2.4 Mbits (6 bits per symbol $\times$ 4 antennas $\times$ $100,000$ symbol vectors) considered for each SNR point. A sample wireless communication testbed, implemented in MATLAB and using the ZF algorithm as the MIMO detector, is provided in Appendix B.

Unfortunately, the large number of symbol vectors required for the high-level simulations makes meaningful analysis using a single workstation employing a serial processing approach almost impossible. For this thesis, high-performance computing using the University of Southampton IRIDIS computer cluster is employed to speed up the analysis by parallelising the high-level source files over several compute nodes as illustrated in Fig. 3.1. Each node has 16 processors and runs at 2.4 GHz. This distributed processing approach results in several orders of magnitude reduction in the simulation times compared with a single workstation. After the functionality of the algorithm is verified, the next step is to convert the variables into a suitable digital number format, which is discussed in the next section.

3.2.2 Number Representation

The received signal and the entries of the channel matrix in (2.17) are real numbers and must be converted to a suitable format for hardware implementation. Numbers can be represented using either floating or fixed-point format. Floating-point number representation converts a number into a format similar to the scientific notation as a product of a significand and an exponent. With a relatively small word-length, a wide range of numbers can be represented using floating-point, however, for many applications, the area and power consumption of the floating-point computation units outweighs the extra accuracy obtained from this number representation [83]. The fixed-point format is much simpler than the floating-point format and provides an acceptable performance for many DSP applications. The fixed-point format is described in more detail in the next section. Then in Section 3.2.2.2, the appropriate variable word-lengths for the hardware implementation of the SD for a 64-QAM 4×4 MIMO system will be determined.

Table 3.1: Fixed-point values of SD parameters

Variable	$\hat{y}_i$	$r_{i,j}$	s_i	b_i	T_i
Format	$Q(8,6)$	$Q(4,10)$	$Q(3,0)$	$Q(8,6)$	$Q(7,6)$
Signed	Yes	Yes	Yes	Yes	No

3.2.2.1 Fixed-Point Number Representation

The fixed-point number representation assigns a fixed number of bits for the integer and fractional parts of a number respectively. The most common notation for fixed-point numbers is the Q-notation, where a number is represented as $Q(i, f)$, where i denotes the number of bits for the integer part, and f denotes the number of bits for the fractional part of the number. For a fixed word-length, i and f can be varied to increase or decrease the range and the precision of the fixed-point representation respectively. The maximum number a number can take using the Q-format is $\sum_{f}^{-i+1} 2^{-k}$ for unsigned numbers and $\sum_{f}^{-i/2+1} 2^{-k}$ for signed numbers.

Another variant of the fixed-point representation sets $i = 0$, which pushes the decimal point to the left of the MSB [84]. In this case, every number takes a value between 0 and $1-2^{-f}$. If the maximum value (MAX) a number can take is known, then the number can be normalised to $[0, 1)$ by dividing it by the maximum value. This notation is particularly attractive for multiplication since the multiplication of 2 fractional numbers is always fractional. However, it also assumes that the target application has the facility to scale the result back to its real-world representation (i.e. by multiplying it with MAX).

Unlike the floating-point representation, where the location of the decimal point varies depending on the exponent, the location of the decimal point for the fixed-point notation is fixed. The flowchart for adding two fixed-point numbers is shown in Fig. 3.2. If the numbers have different Q formats, then the decimal points have to be aligned manually. Furthermore, if the numbers have different signs, a sign augmentation will be necessary to ensure correct results. Despite these extra bookkeeping procedures, the fixed-point format is widely used in digital circuits where area and power consumption are critical.

3.2.2.2 Fixed-Point Simulation

In this thesis, the fixed-point number format is adopted due to its suitability for low-complexity digital systems. In order to derive the word-lengths for the various SD variables, extensive simulations are carried out using MATLAB's inbuilt fixed-point conversion tool, which determines the typical values that the variables may assume based on simulation data.

A BER versus SNR simulation for the SD for a 64-QAM 4×4 MIMO system using various word-lengths for the entries of $\hat{\mathbf{y}}$ and $\mathbf{R}$ is shown in Fig. 3.3. The simulation is carried out

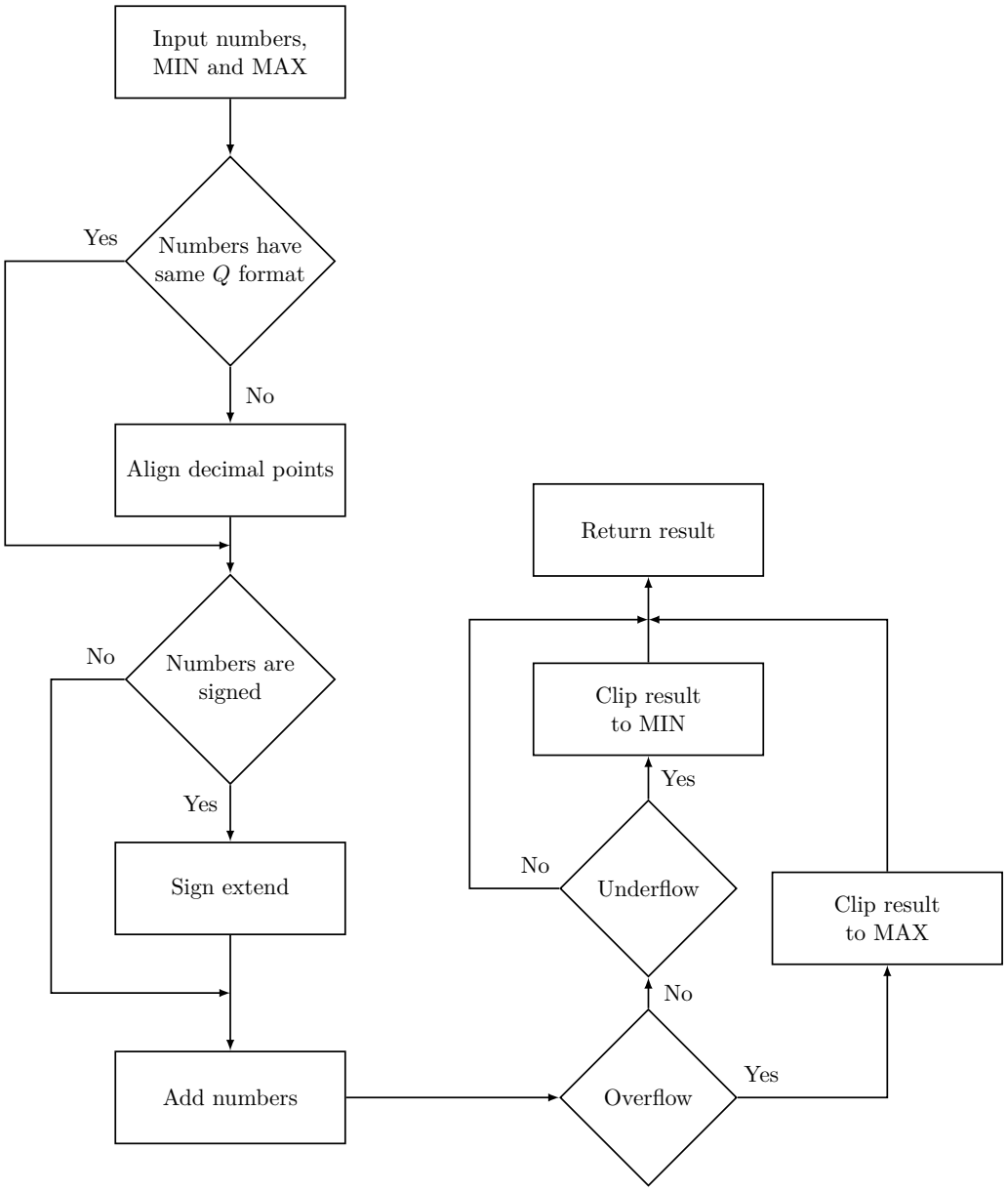


Figure 3.2: Flowchart for adding two fixed-point numbers

using 100, 000 transmitted symbol vectors with the channel randomly regenerated for every 4 consecutive symbol vectors. The simulations show that a word-length of 14 for $\hat{y}_i$ and $r_{i,j}$ practically achieves the same performance as a word-length of 32 and incurs an SNR loss of less than 1 dB relative to the floating-point ℓ^2-norm simulation at a BER of 10^{-3}. The fixed-point simulation of the SD is based on the ℓ^1-norm approximation of the partial Euclidean distance presented in [45]. The fixed-point values of the various SD variables are shown in Table 3.1. The constellation points, s_i, for 64-QAM can be represented using a word-length of 4. However, since all the constellation points are odd-valued, the LSB, which is always a "1" can be discarded during the normal SD processing in order to save area and appended at the output of the detector.

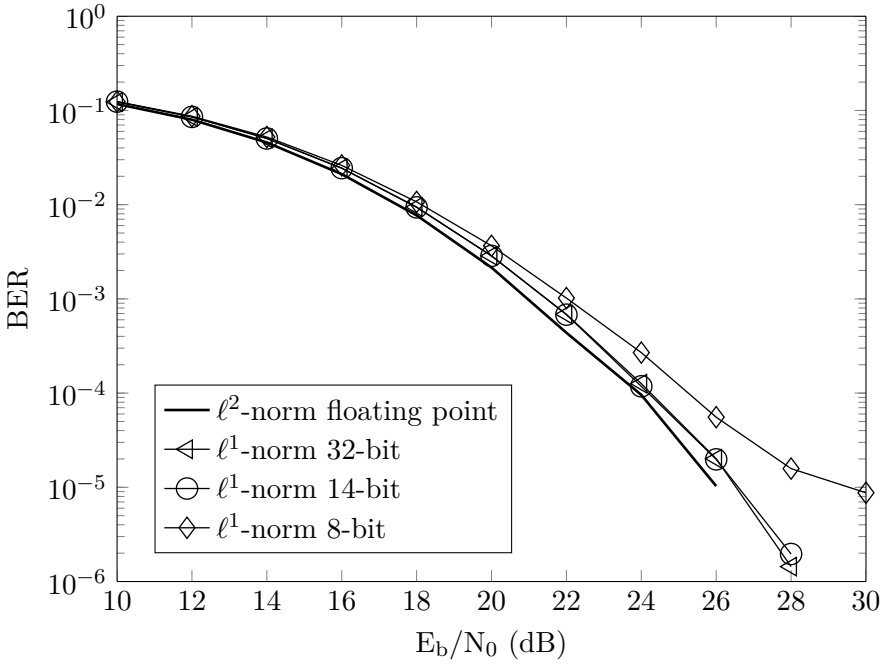


Figure 3.3: BER versus SNR simulation for different word-lengths

3.2.3 ASIC Design Flow

In this section, we will describe the design flow for taking an algorithm, implemented in the high-level analysis step, and converting it into an ASIC ready for fabrication. The algorithm can alternatively be implemented on an FPGA, or as a software routine running on a general-purpose microprocessor, however, an ASIC allows a relatively small design and lower power consumption to be achieved compared to the alternatives. Figure 3.4 shows a simplified ASIC design flow. The flow is highly iterative, and several passes of the flowchart are usually necessary to achieve a design that meets the desired specification. In the next sections, the various stages of the ASIC design flow are discussed in more detail.

3.2.3.1 RTL Implementation

After the algorithm has been functionally verified at high-level and the fixed-point word-lengths for the variables have been determined, the next step is to implement the algorithm using a hardware description language, such as SystemVerilog, to create a register transfer level (RTL) design. The RTL implementation can be generated automatically from a high-level programming language, such as SystemC and MATLAB, using a high-level synthesis tool [85], or it can be implemented through manual coding. For this thesis, the manual approach is adopted as it allows for more control over the architecture selection. The RTL is then simulated in ModelSim to verify its functionality with the aid of stimulus files generated from MATLAB. The stimulus files contain randomly generated

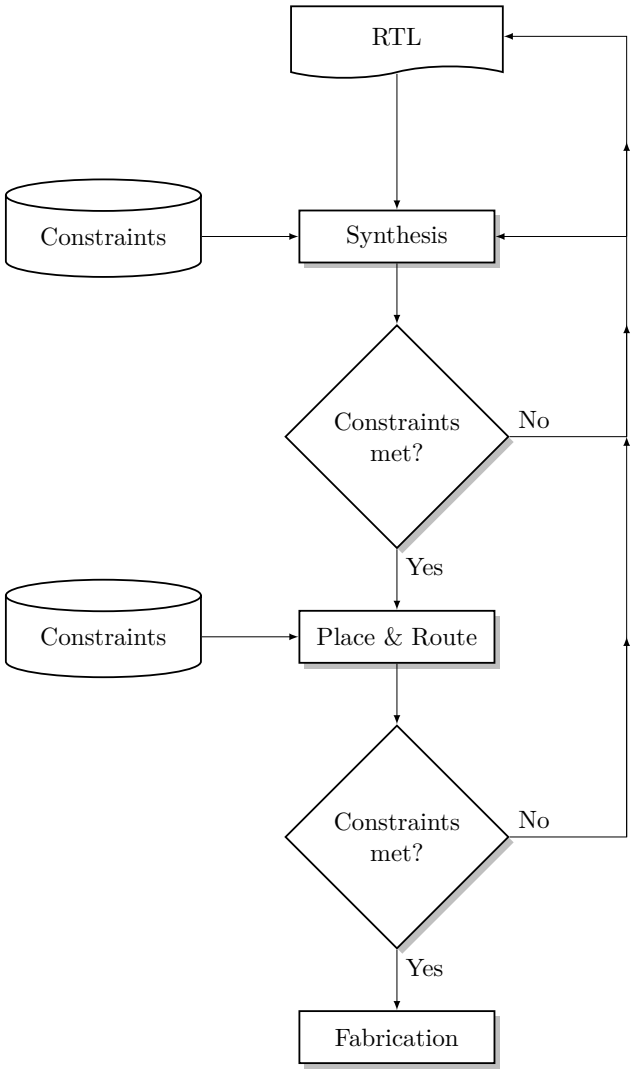


Figure 3.4: Simplified ASIC Design Flow

values of $\hat{y}_i$ and $r_{i,j}$ for the SNR range considered (0 to 30 dB). The BER of the hardware implementation is then obtained as follows:

$$\text{BER} = \frac{\sum_{i=1}^{|\mathbf{x}|} x_i \neq \hat{x}_i}{|\mathbf{x}|},$$

where $|\mathbf{x}|$ is the number of bits in the transmitted vector, $\mathbf{x}$, and x_i and $\hat{x}_i$ are the ith transmitted (from MATLAB) and detected bits (from the SystemVerilog implementation) respectively. During the entire ASIC flow, it is often necessary to return to the functional verification step several times to verify that the circuit still meets the design intent.

3.2.3.2 Logic Synthesis

In this stage, the RTL netlists are converted into gate-level netlists using gates from a proprietary cell library, using a synthesis tool such as Design Compiler from Synopsys. For the proposed implementation, the 65 nm low threshold voltage process, `CORE65LPLVT`, from STMicroelectronics is used for the synthesis. The first step of the logic synthesis is to determine a number of realistic constraints that the circuit should meet. These constraints include the timing and power consumption budget for the design. The synthesis begins with an initial design exploration using a low-effort compilation. Typically, if the timing of the design fails to meet the timing budget by more than 15%, further modifications to the RTL might be necessary [86]. The low-effort synthesis also allows various architectures to be investigated quite quickly. For larger designs, a bottom-up synthesis strategy is employed, where sub-modules are compiled separately and then linked at the topmost module. The choice of synthesis strategy can have quite a dramatic impact on the quality of results. The bottom-up synthesis procedure is carried out using the following pseudocode:

```
foreach submodule in submodules
    analyse submodule
    apply constraints
    compile submodule
end

analyse top level module
apply top level and I/O port constraints
compile top level module
generate reports
```

Small-sized submodules are synthesised in a top-down manner along with their parent modules. Although the bottom-up strategy does not always yield the best synthesis results, it reduces the memory usage, and enables "snapshots" of the synthesis output at various stages to be saved for further exploration.

3.2.3.3 Place and Route

Place and route refers to a series of steps where the cells in the gate-level netlist, generated by the synthesis tool, are interconnected. Other important steps include specifying the floorplan for the chip, and efficiently routeing the clock signal (also referred to as "clock tree synthesis") to reduce the clock skew and latency. For this thesis, Cadence Encounter is used for the place and route stage. In order to make the timing result from

the logic synthesis closely match the output of the place and route, a clock latency of 1 ns is added to the clock period during the synthesis step. This reduces the number of potential iterations between the synthesis and place and route stages. After the design is routed, it is typically sent for further functional and layout verification steps before being sent to the foundry for fabrication. The area consumption and maximum clock frequency of the design are determined at this stage.

3.2.4 Power Estimation Flow

Power estimation at different stages of the ASIC design flow is a necessary step for achieving a power-efficient implementation. Power estimation during early design stages can quickly inform the designer on whether a modification to the RTL is necessary, or if low power techniques, such as clock gating and operand isolation, need to be applied during the *high-effort* synthesis stage, to meet the power target set for the design.

After the logic synthesis step, the gate-level netlist is passed through a second *post-synthesis* step to obtain a more accurate power consumption estimate. Synopsys Power Compiler [87] is used for estimating the power consumption of the design. Power Compiler estimates the power consumption by using the static probability and toggle rate for all the nets in the design, which are collected during a gate-level post-synthesis timing simulation using the Synopsys Delay Format (SDF) file, generated after the logic synthesis. (The SDF will also be later used during the post-route timing optimisation step in Cadence Encounter). The switching activity of the design is saved as a value change dump (VCD) file, which is subsequently converted into a switching activity information format (SAIF) file by the synthesis tool. This is then used to *annotate* all the nets and signals in the design during the post-synthesis gate-level optimisation.

For the proposed implementation, operand isolation is applied automatically to the designs during the synthesis stage in order to reduce power consumption. Flip-flops are also equipped with enable signals wherever possible during *idle* states. For the post-synthesis power optimisation, only an incremental compilation is performed, where the synthesis tool incrementally tries different logic selections in order to improve the quality of results. No logic mapping is carried out in this stage. Due to the large size of the switching activity dump files, the power analysis simulations are restricted to a total of 100 symbol vectors per SNR point. Eleven SNR points are considered, which results in a total of 1100 symbol vectors. For the post-synthesis gate-level simulation, the analysis is carried out using 220 symbol vectors (20 symbol vectors per SNR). To summarise, the power consumption estimate is obtained as follows:

1. Simulate RTL/post-synthesis netlist in Modelsim and generate a value change dump (VCD) file containing switching activities of the design and its sub-modules
2. Convert VCD file into a SAIF file for use by Power Compiler

3. Read SAIF file from Design Compiler and annotate the design and its sub-modules
4. Run Design/Power Compiler to synthesise design and perform power optimisation and estimation
5. Generate power consumption reports
6. Modify RTL/synthesis strategy/synthesis constraints until a satisfactory power estimate is obtained

3.3 Sphere Decoder with Runtime Constraints

In the previous chapter, the sphere decoder was introduced as a low-complexity alternative to the maximum likelihood detector. It was noted that the SD is a data-dependent algorithm with a complexity that varies according to the SNR and the characteristics of the channel matrix. As such, the number of clock cycles required to process a single RSV and by extension, the throughput of the detector, cannot be known beforehand. The throughput of the SD is computed based on the expected number of clock cycles, $E\{N_{\text{clk}}\}$, as follows:

$$\Phi = \frac{Q \times N_T \times f_{\text{clk}}}{E\{N_{\text{clk}}\}}. \tag{3.1}$$

Unfortunately, N_{clk} at lower SNRs may be too large as a result of the higher number of nodes visited (see Fig. 2.8), which results in a significant throughput degradation. In practice, runtime constraints are applied to the SD to achieve a higher average throughput. Some of these techniques are discussed subsequently.

3.3.1 Sphere Decoder with Early Termination

The complexity of the SD can be bounded by specifying a maximum number of nodes, D_{max}, which can be visited during the tree traversal [64]. The tree search is terminated if D_{max} nodes are visited or if no remaining solution is better than the best solution found so far. D_{max} must be selected such that at least one valid solution could be found (i.e. $D_{\text{max}} \geq N_T$ for the complex model SD and $D_{\text{max}} \geq 2N_T$ for the real channel model). Unfortunately, this simplistic approach may result in a performance degradation, since all symbols are detected with the same computational effort irrespective of the channel condition. A more efficient early termination strategy is described in the next section.

3.3.2 Sphere Decoder with Block Processing

Consider a block of N received signal vectors. Instead of the per-symbol early termination described in the previous section, an average of runtime constraint $D_{\text{avg}} \approx D_{\text{max}}$ can

be assigned to each symbol vector in the block, such that $\sum_{n=1}^{N} D^{(n)} = ND_{\max}$, where $D^{(n)}$ is the number of nodes actually visited for the nth symbol vector in the block.

One strategy for assigning the runtime constraint among the RSVs is the recursive "maximum-first" approach proposed in [88] for the one-node-per-cycle SD, where earlier RSVs are given preference to later ones. For the nth symbol vector, the maximum number of nodes that may be visited is determined as:

$$D_{\max}^{(n)} = ND_{\text{avg}} - \sum_{j=1}^{n-1} D^{(j)} - (N-n)N_T. \tag{3.2}$$

The first term on the right-hand side of (3.2) represents the total number of nodes that can be visited per block; the second term refers to the total number of nodes visited for previous symbol vectors within the block prior to the current symbol vector, while the third term guards against detection failure by guaranteeing at least one solution is found for the yet-to-be-detected symbol vectors. A solution needs at least N_T nodes to be visited using a complex channel model and the one-node-per-cycle (ONPC) tree search [45]. For a detector that requires more than one cycle per node, and employing a real channel model, then the third term will need to be adjusted to *number of clock cycles per node* $\times(N-n) \times 2N_T$.

In this thesis, we propose a modification to (3.2) by defining the runtime constraint based on the *number of solutions found* rather than on the *number of visited nodes*. Counting the number of visited nodes, rather than the number of solutions found, potentially leads to a high power consumption especially at low SNR, where a large number of potential solutions are encountered. In the latter case, the counter is only updated when a solution is found, while in the former the counter is updated for every node visited irrespective of whether a solution has been found or not. Equation (3.2) can thus be modified as:

$$\widetilde{D}_{\max}^{(n)} = N\widetilde{D}_{\text{avg}} - \sum_{j=1}^{n-1} \widetilde{D}^{(j)} - (N-n), \tag{3.3}$$

where the second term in this case is the total number of solutions that have been found so far.

3.3.3 Simulation Results

Figure 3.5 shows the BER versus SNR simulation for the sphere decoder using different runtime constraints, using a block size of $N = 4$. The simulation was considered for $200,000$ transmitted symbol vectors and $50,000$ channel matrices. The figure shows that applying early termination on a *per-block* basis is advantageous in terms of the BER performance. The SD with $\widetilde{D}_{\text{avg}} = 100$ exhibits the same performance as the unbounded SD with $\widetilde{D}_{\max} = \infty$. This suggests that the SD is inefficient at certain channel conditions

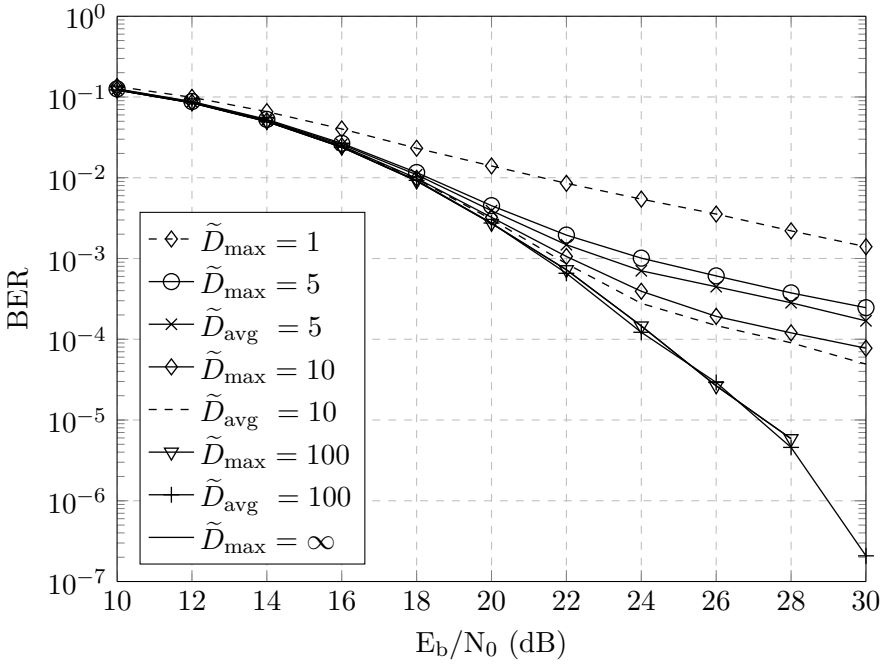


Figure 3.5: BER versus SNR simulation for the sphere decoder using per-symbol and per-block runtime constraints

as a longer processing time may not always result in an appreciable improvement to the BER. With $\widetilde{D}_{\max} = 1$, the SD has a similar complexity to the ZF-SIC detector and offers the maximum data rate possible. However, the performance loss is appreciable at high SNR. The SD with $\widetilde{D}_{\text{avg}} = 10$ provides a near-optimal performance up to about a BER of 10^{-3}, while incurring a much lower average complexity than the SD with $\widetilde{D}_{\text{avg}} = 100$. The actual choice of $\widetilde{D}_{\text{avg}}$ will depend on a combination of performance and throughput considerations.

3.4 Hardware Implementation

In this section, the VLSI implementation of the SD, based on the proposed runtime constraints, is presented. The SD architecture is divided into a single controller unit and a datapath, which allows separate testing and a more maintainable design to be achieved. The real channel model will be adopted in order to achieve a low-complexity implementation. The implementation of the SD is explained in more detail in subsequent sections.

3.4.1 Controller

The controller unit oversees the entire operation of the detector and is implemented as a finite state machine (FSM). The controller has two main operations: *forward trace* and

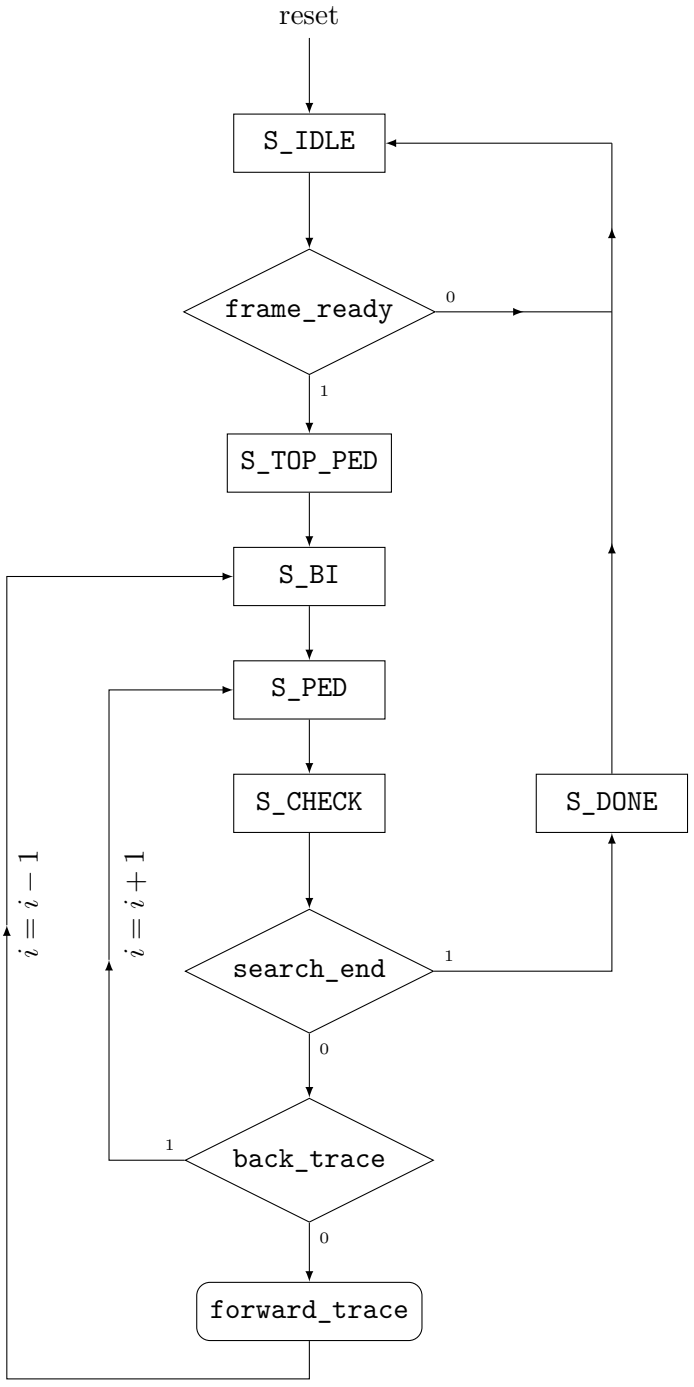


Figure 3.6: ASM chart for the sphere decoder

back trace. A trace refers to the movement of the SD from one tree level to another based on the accumulated metric of the current path. The detector is in forward tracing mode any time the metric of a visited node is less than the metric of the Babai point, that is, $T(s_i) < T_{\text{Babai}}$. This is also the case if the controller is currently processing the top level nodes, i.e. the constellation points, where a backward trace is not possible. The detector is also automatically in a forward trace mode during the initial tree traversal before the bottom level is reached. Any time a visited node satisfies the sphere constraint, the level

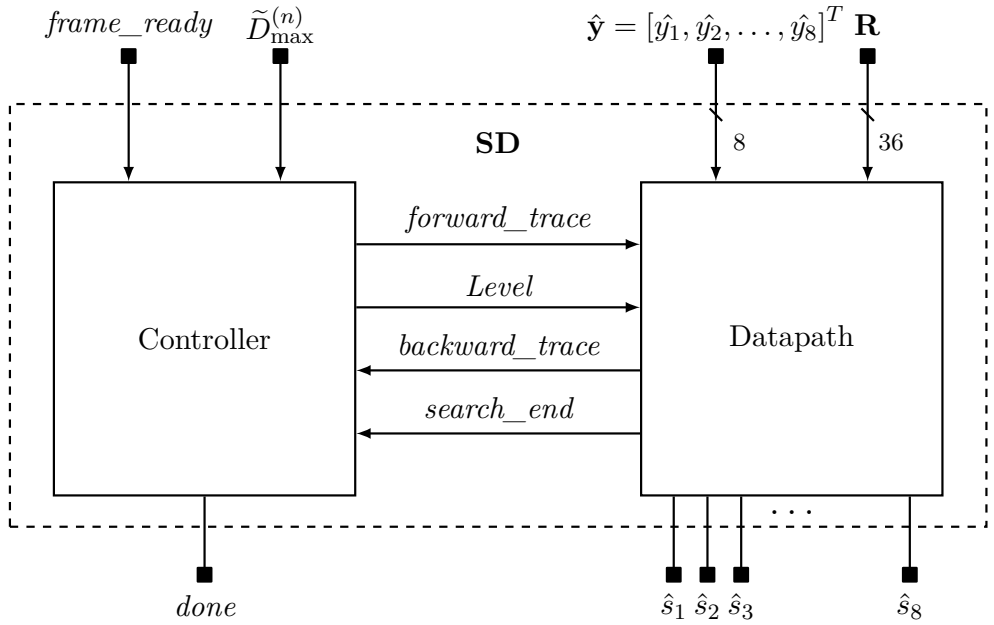


Figure 3.7: High-level architecture of the SD showing the controller and datapath. For a 4×4 MIMO system, there are 8 signal entries and 36 channel matrix entries, assuming a real-valued channel model is employed.

indicator, `Level`, is decremented, and the detector fetches the best child of the current node for the next visit at the subsequent level.

On the other hand, the detector enters the back trace mode any time the sphere constraint is violated. This can only happen in levels $i = 2N_T - 1$ to 1. In the bottom level, $i = 1$, a back trace is performed irrespective of whether the sphere constraint is violated or not, in order to try another path. Since the nodes at a given level are visited according to their metrics, a sphere constraint violation on a given node automatically implies that all its siblings will also violate the sphere constraint, and as such are not worth visiting. Any time a back trace occurs, the level counter is incremented, and the sibling of the parent of the current node is marked for the next visit. In the proposed implementation, all the children of a node at a given level are stored into a shift register, and its output at any point corresponds with the next best child. A dummy symbol is also stored into the shift register, which is used to check if all the children of a node have been exhausted.

The algorithmic state machine (ASM) chart for the controller unit is shown in Fig. 3.6. The controller comprises 6 states namely: `S_IDLE`, `S_TOP_PED`, `S_BI`, `S_PED`, `S_CHECK` and `S_DONE`. All the state registers are encoded using one-hot encoding, which assigns one flip-flop to each state, resulting in a fast state decoding. The states of the controller ASM are described as follows.

- `S_IDLE` : The detector is put to an *idle* state upon the assertion of an asynchronous reset signal. The detector remains in this state until the synchronous `frame_ready`

input signal is asserted. Counters are used to keep track of the tree level and the number of symbol vectors found so far. The symbol vector counter is updated any time the last level is reached, that is, $i = 1$. In the idle state, the level and symbol counters are restored to their default values of $2N_T$ and 0 respectively. Similar to the state registers, the level counter is represented using one-hot encoding, such that an independent bit location corresponds with a level of the tree search.

- `S_TOP_PED` : In this state, the constellation points, $s \in \mathcal{D}$, are expanded, which marks the start of a new path. As mentioned previously, a back trace is not possible in this state, since the symbols at the top level have no ancestors. An SE enumeration is also performed at this level in order to locate the Babai point quickly.

- `S_BI` : In the proposed SD, a node is expanded over 2 cycles. In the first cycle, the interferences of previously detected symbols are computed and cancelled from the current layer, and in the second cycle, the PED of the current symbol is computed. The former operation is executed in this state. By the decoupling the node expansion in this way, the proposed SD incurs a higher number of clock cycles compared to the ONPC SD [45]. A higher throughput can be achieved by expanding the nodes in one cycle, albeit, at the expense of a longer critical path. Different approaches for computing b_i are described in Section 3.4.3.2. For levels $i = 2N_T - 1$ to 1, this state marks the beginning of a new level.

- `S_PED` : The metric of the current node is computed in this state, using the b_i output of `S_BI` as an input. After the metric is computed, the datapath, shown in Fig. 3.7, issues a `back_trace` signal depending on whether the sphere constraint has been violated or not. The datapath also issues a `search_end` signal when a top level node has violated the sphere constraint[1] or if all viable branches have been visited. Similar to `S_TOP_PED`, an SE enumeration is also performed in this state, which ensures that the best child is always visited first.

- `S_CHECK` : In this state, the `back_trace` and `search_end` signals generated from `S_PED` are examined. If a back trace is triggered, then the level counter is incremented. The detector then returns to `S_PED`, and iteratively back tracks until a viable node satisfying the sphere constraint is found. If `back_trace` is not asserted, the level counter is decremented and the controller moves to `S_BI`. If `search_end` is triggered, the controller moves to `S_DONE`, which ends the detection for the current RSV.

[1] If a top level node has violated the sphere constraint it implies that all its branches, as well as any subsequent top level nodes, will violate the sphere constraint as well. In the former case, this is as a result of the monotonically increasing nature of the PED, which dictates that any branch emerging from a node will have a larger PED, while in the latter case, it is as a result of the Schnorr-Euchner enumeration, which arranges sibling nodes according to their metrics.

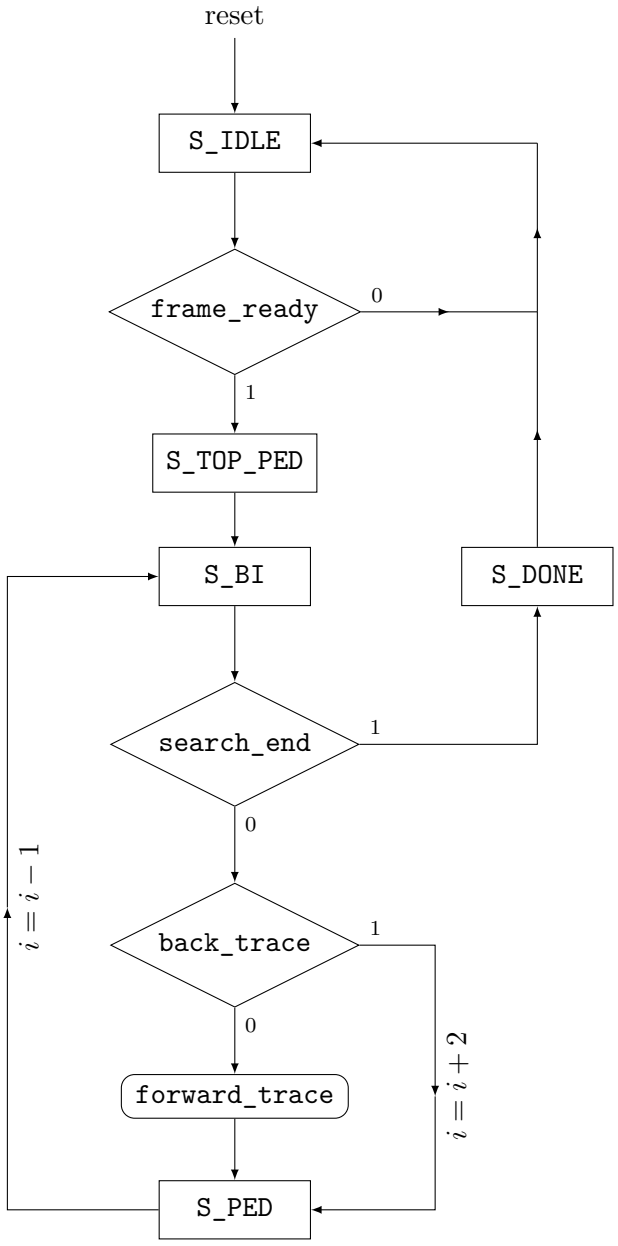


Figure 3.8: Modified ASM chart for the sphere decoder with look ahead

In the next section, we will propose a more efficient implementation of the controller unit.

3.4.2 Controller with Look-Ahead

The controller ASM presented in the previous section requires an additional cycle to check if a back-trace has occurred, which can result in a large latency especially at low SNR. In this section, we will present a modification to the controller discussed previously,

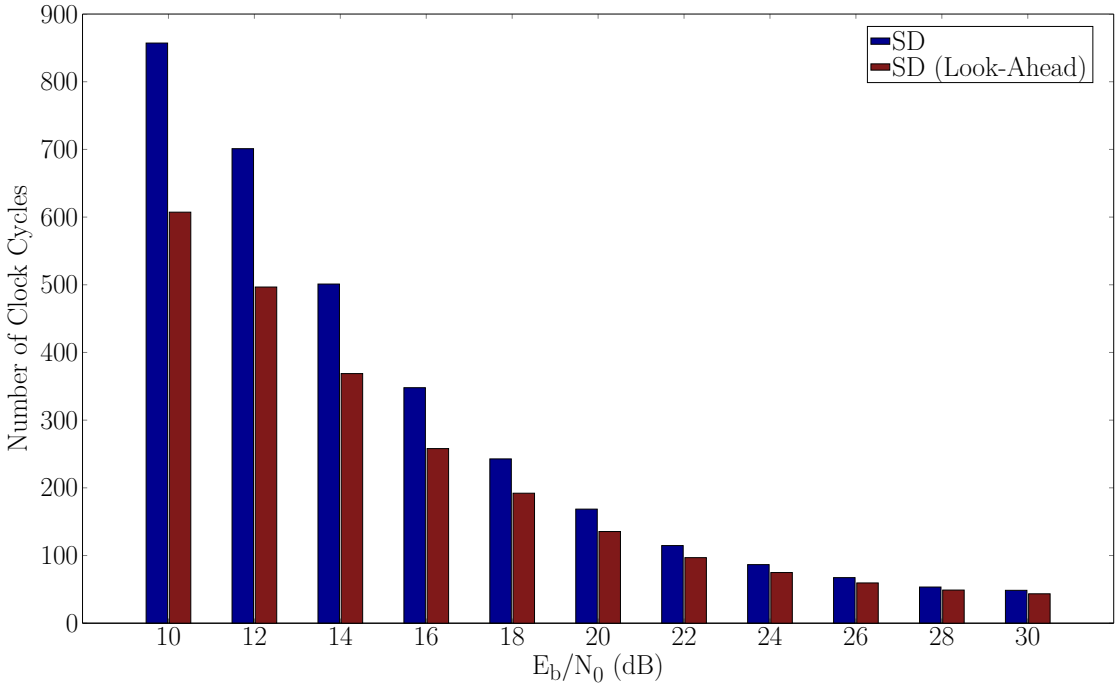


Figure 3.9: Impact of "look-ahead" on the speed of the sphere decoder

by adopting a "look-ahead" strategy,[2] where the controller immediately transits to the next level, $i-1$, even before it is known if a back-trace has occurred or not. In other words, the PED computation as well as the next level transition occur within the same cycle. Unlike the conventional SD, the direction of the tree search (whether to increment or decrement the level counter) is only known after the PED has been computed and compared with the metric of the Babai point.

The tree traversal using the look-ahead strategy is illustrated in Fig. 3.10. In the conventional SD, the PED and sphere-constraint check (i.e. $T_i < T_{\text{Babai}}$) are computed serially, which is represented by the horizontal lines within the tree nodes (represented by circles in the figure). Meanwhile, the PED and sphere-constraint check are computed *concurrently* in the modified SD with look-ahead, which is indicated by the vertical line within the tree nodes. "1" and "2" represent the PED and sphere-constraint check operations respectively. Observe that when a back-trace occurs in the modified SD, the SD actually still descends to the next level, $i-1$. This is unlike the conventional SD, where the SD immediately back-tracks to the previous level, $i+1$, once it is known that the sphere constraint has been violated. However, since a back-trace had been flagged in the previous level, the node $s_{i-1}^{[1]}$ is never processed. Instead, the SD traverses to the sibling of $s_{i+1}^{[j]}$, i.e., $s_{i+1}^{[j+1]}$ by back-tracing *twice*. This operation effectively makes Fig. 3.10(d) equivalent to Fig. 3.10(b). It should be mentioned that in both the conventional SD and the SD with look-ahead, a back-trace incurs the same number of clock cycles (i.e. 2), as

[2] An SD with "look-ahead" was also presented in [89], where the SD attempts to predict the next node at level $i-1$. However, this results in an increase in the number of clock cycles. Look-ahead, as used in this chapter, refers to making a forward trace without checking if the sphere constraint has been violated or not, which reduces the number of clock cycles as shown in Fig. 3.9.

such the look-ahead technique does not incur any additional clock cycles compared with the conventional SD.

Figure 3.8 shows the ASM chart for the SD using the look-ahead strategy. In the modified ASM chart, the `search_end` and `back_trace` checks are moved from `S_CHECK` to `S_BI`. If $T(s_i)$ is eventually less than T_{Babai}, then it means a correct decision has been made and an extra cycle is saved. However, if an incorrect decision has been made, then this must be corrected, which is achieved by incrementing the level counter twice, which back-traces the controller to level $i+1$.

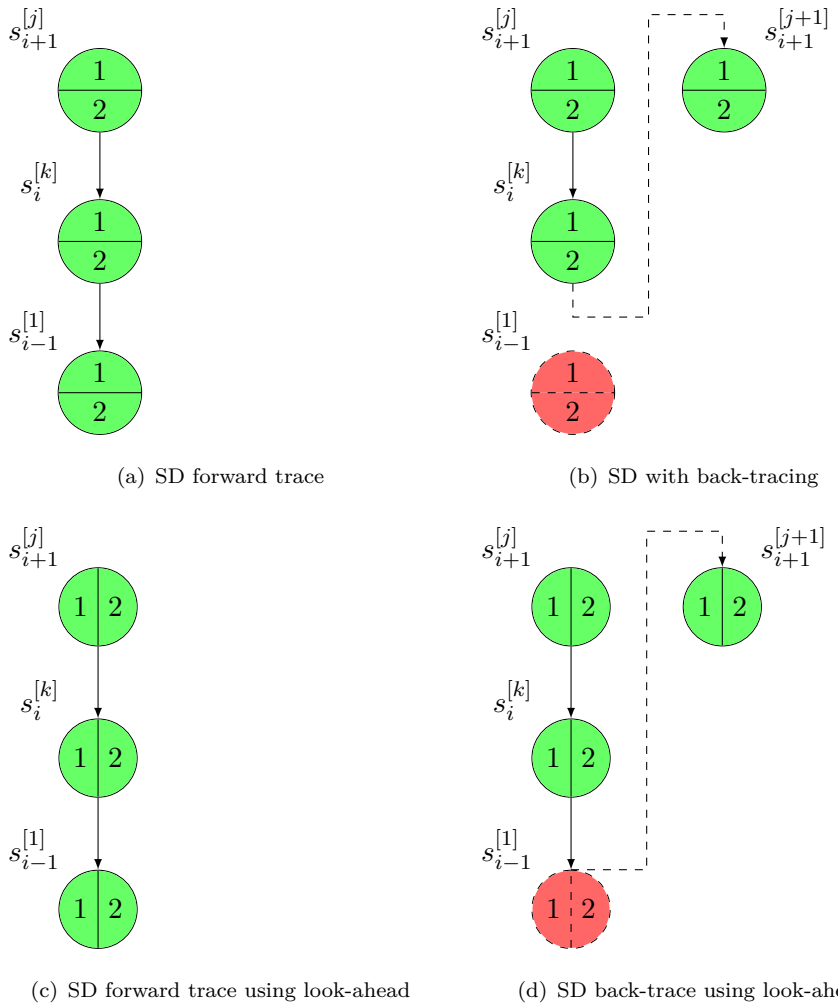
Figure 3.9 compares the number of clock cycles required by the conventional SD with that of the SD with look-ahead. The RTL implementation of the SD is simulated using $100,000$ symbol vectors for both the conventional and modified look-ahead strategies, and the average of the number of clock cycles required to detect a single symbol vector is then taken for each SNR point. In spite of the occurrence of erroneous forward traces, the number of clock cycles using the look-ahead strategy is significantly reduced compared to the original SD. Since the forward trace is accelerated through the use of the look-ahead strategy, and a back trace incurs the same number of cycles in both the conventional and modified tree search, the look-ahead will always reduce the total number of clock cycles required by the SD. As expected, the reduction in the number of clock cycles is higher in low SNR regions due to the larger number of solutions observed in those regions. On average, a percentage reduction of 25.5% is achieved in the number of clock cycles over an SNR range of 0 to 30 dB by applying look-ahead.

3.4.3 Datapath

The main function of the datapath is to expand the nodes and enumerate them based on their metrics. The datapath also sends control signals back to the controller to signal a "back-trace" or an end to the tree search as shown in Fig. 3.7. After expanding a node, it is saved to a temporary $2N_T \times 1$ symbol register, $\mathbf{T}$, if its metric is less than the metric of the best solution found so far. If a leaf node has a metric that is less than that of the best solution found so far, then the contents of $\mathbf{T}$ are copied to the symbol register, $\mathbf{S}$, which holds the current best solution. The symbol vector counter is also updated at this point, and its value is compared with $\widetilde{D}_{\max}^{(n)}$ before the start of a new tree search. If the symbol vector counter exceeds $\widetilde{D}_{\max}^{(n)}$, then the current content of $\mathbf{S}$ is presented as the hard-detection output. The implementation of the datapath is explained in more detail in subsequent sections.

3.4.3.1 Tabular Enumeration

In the Schnorr-Euchner SD, the nodes at each level are visited in accordance with their metrics, which speeds up the tree traversal. In Chapter 2, we explained that the metrics



(a) SD forward trace

(b) SD with back-tracing

(c) SD forward trace using look-ahead

(d) SD back-trace using look-ahead

Figure 3.10: Illustration of SD tree traversal with look-ahead. The "1" in the circles represents the PED computation, while "2" represents the sphere constraint check. The dashed line represents a back-trace operation.

of the nodes do not need to be explicitly computed in order to enumerate them. Instead, the nodes can be arranged by iteratively comparing their distances from a so-called SE centre. In detectors based on a complex model, this will require the computation of trigonometric functions or their approximations [18], which increases the complexity of the datapath.

In the proposed detector, the SE enumeration is computed based on a lookup table (LUT) as proposed by Wiesel, Mestre, Pages, *et al.* [67]. Instead of computing the SE enumeration based on the SE centre, c_i (see Section 2.19), it is computed based on b_i, which dispenses with the need for a division. The $r_{i,i}s_i$ axis is divided into discrete regions, and the enumeration is determined based on the location of b_i on the axis. A tabular enumeration for 64-QAM, using the real-channel model, is illustrated in Table 3.2, where the axis is divided into 14 regions. On close examination, it can be seen that the enumeration of the negative axis mirrors that of the positive axis. Therefore, only

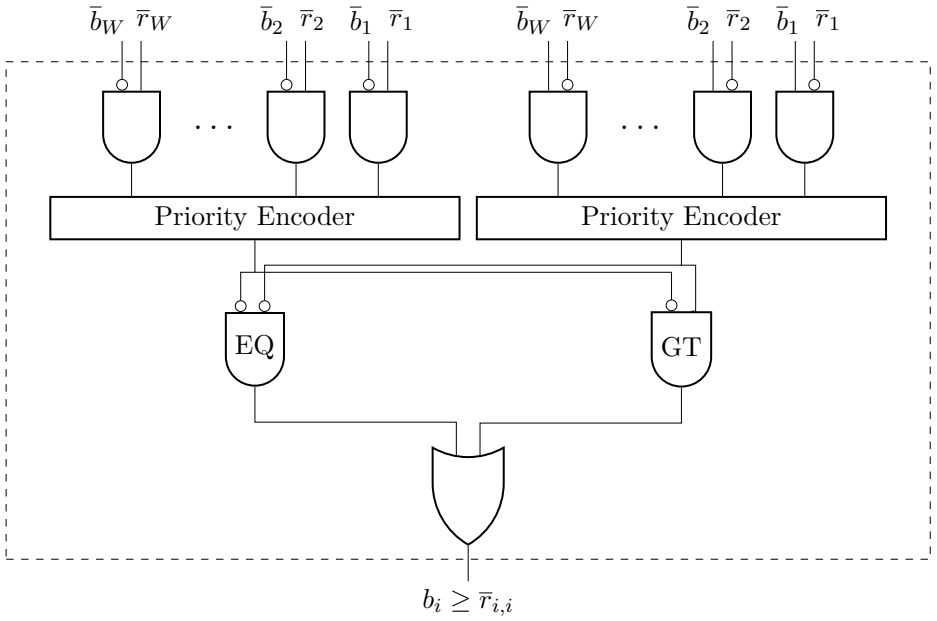


Figure 3.11: Determining enumeration region for b_i using bitwise comparisons

one half of the enumeration table needs to be stored into the LUT. This saves the area by about 57% compared with the full enumeration unit. The final enumeration is then determined based on the polarity of b_i.

Figure 3.12 illustrates the simplified circuit diagram of the SE enumeration unit, where the enumeration is determined by the positive half of the enumeration table. It should be noted that a comparator is not explicitly required for the region $(0, +|r_{i,i}|\,]$, since the comparison $|r_{i,i}| >= 0$ is redundant. Thus, only 6 comparators are required. A 6-to-3 priority encoder is used to determine the location of b_i on the $r_{i,i}s_i$ axis, which is then used to select the appropriate enumeration from the LUT.

In order to reduce the complexity of the SE enumeration unit, b_i is only compared with the left boundary of the decision regions shown in Table 3.2. A straightforward approach to implement the comparison is to use a subtractor and then note if the result is positive or negative. In the proposed implementation, the comparator is implemented by comparing individual bits of b_i and integer multiples of $r_{i,i}$ in parallel and then forwarding the results of the individual comparisons to a priority encoder.

Two priority encoders are required for determining the location of b_i as shown in Fig. 3.11. The wordlength is indicated by W in the figure, integer multiples of $r_{i,i}$ are denoted by $\overline{r}_{i,i}$, and the jth bit for b_i and $\overline{r}_{i,i}$ is denoted by $\overline{b}_j$ and $\overline{r}_j$ respectively. If the output of right encoder is "1" and the left encoder is "0", it implies that b_i is greater than (GT) $\overline{r}_{i,i}$. The leftmost AND gate checks if b_i is equal (EQ) to $\overline{r}_{i,i}$. An additional OR gate is required to check if $b_i \geq \overline{r}_{i,i}$. It should be noted that the decimal points of b_i and $\overline{r}_{i,i}$ need to be aligned to ensure a correct comparison. The proposed comparator reduces the critical path length by approximately 13% compared with the direct unoptimised

Table 3.2: Tabular enumeration for 64-QAM

Region	**Enumeration**							
$(-\infty, -6\lvert r_{i,i}\rvert\,]$	-7	-5	-3	-1	$+1$	$+3$	$+5$	$+7$
$(-6\lvert r_{i,i}\rvert, -5\lvert r_{i,i}\rvert\,]$	-5	-7	-3	-1	$+1$	$+3$	$+5$	$+7$
$(-5\lvert r_{i,i}\rvert, -4\lvert r_{i,i}\rvert\,]$	-5	-3	-7	-1	$+1$	$+3$	$+5$	$+7$
$(-4\lvert r_{i,i}\rvert, -3\lvert r_{i,i}\rvert\,]$	-3	-5	-1	-7	$+1$	$+3$	$+5$	$+7$
$(-3\lvert r_{i,i}\rvert, -2\lvert r_{i,i}\rvert\,]$	-3	-1	-5	$+1$	-7	$+3$	$+5$	$+7$
$(-2\lvert r_{i,i}\rvert, -\lvert r_{i,i}\rvert\,]$	-1	-3	$+1$	-5	$+3$	-7	$+5$	$+7$
$(-\lvert r_{i,i}\rvert, 0\,]$	-1	$+1$	-3	$+3$	-5	$+5$	-7	$+7$
$(0, +\lvert r_{i,i}\rvert\,]$	$+1$	-1	$+3$	-3	$+5$	-5	$+7$	-7
$(+\lvert r_{i,i}\rvert, +2\lvert r_{i,i}\rvert\,]$	$+1$	$+3$	-1	$+5$	-3	$+7$	-5	-7
$(+2\lvert r_{i,i}\rvert, +3\lvert r_{i,i}\rvert\,]$	$+3$	$+1$	$+5$	-1	$+7$	-3	-5	-7
$(+3\lvert r_{i,i}\rvert, +4\lvert r_{i,i}\rvert\,]$	$+3$	$+5$	$+1$	$+7$	-1	-3	-5	-7
$(+4\lvert r_{i,i}\rvert, +5\lvert r_{i,i}\rvert\,]$	$+5$	$+3$	$+7$	$+1$	-1	-3	-5	-7
$(+5\lvert r_{i,i}\rvert, +6\lvert r_{i,i}\rvert\,]$	$+5$	$+7$	$+3$	$+1$	-1	-3	-5	-7
$(+6\lvert r_{i,i}\rvert, +\infty\,]$	$+7$	$+5$	$+3$	$+1$	-1	-3	-5	-7

implementation. After the appropriate SE enumeration region is determined, a single XOR gate in Fig. 3.12 decides whether to "flip" or "keep" the generated enumeration based on whether b_i and $r_{i,i}$ have different or the same signs respectively.

3.4.3.2 Partial Euclidean Distance

In Chapter 2, we introduced the partial Euclidean distance (PED) computation for tree search (TS) algorithms. The PED plays a crucial role in determining the performance and complexity of TS algorithms, and several low-complexity approximations to the PED have been investigated for the purpose of VLSI implementation. Most notably, a number of norm approximations to the PED were proposed by Burg, Borgmann, Wenk, *et al.* [45], which significantly reduced the area cost of the SD at a small penalty to the BER. For this implementation, the ℓ^1-norm approximation is employed, which computes the PED as follows:

$$T_i = T_{i+1} + \lvert b_i - r_{i,i}s_i\rvert\,.$$

Table 3.3 shows the conventional computation of the interference term in (2.14) at each level, i, for $N_T = 3$. It is immediately clear that at large N_T the interference can become a throughput bottleneck due to the large number of additions required in series at lower levels. For example, at the last level, 4 adders are required in order to compute $\sum_{j=2}^{6} r_{i,j}s_j$, which can negatively impact the attainable clock frequency. A simplistic solution to this problem is to compute $\sum_{j=i+1}^{2N_T} r_{i,j}s_j$ incrementally over $2N_T - i$ cycles. However, this will incur a large latency resulting in a significant throughput degradation.

Kang and Park [90] proposed an interesting solution to this problem by computing the interference terms immediately a new symbol has been detected. Any time a new symbol,

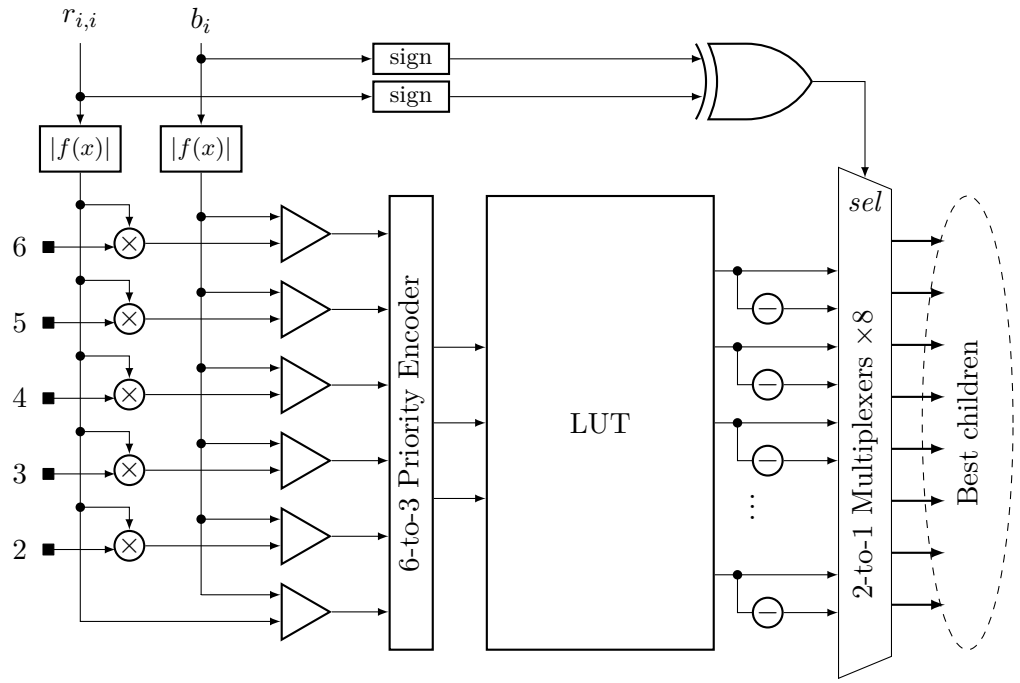


Figure 3.12: Reduced-complexity SE enumeration computation unit for 64-QAM using a lookup table. The enumeration is computed based on the positive half of the $r_{i,i}s_i$ axis for reduced complexity.

s_j, is detected, all the dependent $r_{i,j}s_j$ terms from level $i-1$ to level 1 are computed and stored into registers for later use. In other words, the $r_{i,j}s_j$ terms in Table 3.3 are computed "vertically" instead of "horizontally". For example, in level 5, all the 5 terms in the first column of Table 3.3 are computed in the same clock cycle and stored into registers. Since the interference terms are used for the computation of independent T_i, the number of adders required in series is constant at all the tree levels irrespective of the number of antennas. The result is that only 2 adders are required for the b_i computation at level 1 instead of 6 adders in the direct implementation as shown in (3.4) and (3.5) respectively:

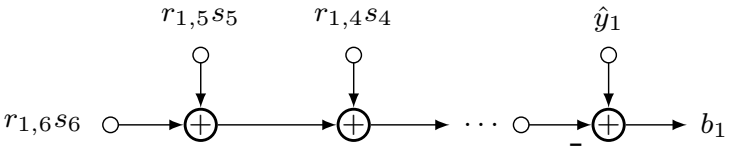
$$b_1 = \hat{y}_1 - b_1^2 \tag{3.4}$$

$$b_1 = \hat{y}_1 - \sum_{j=2}^{6} r_{1,j}s_j, \tag{3.5}$$

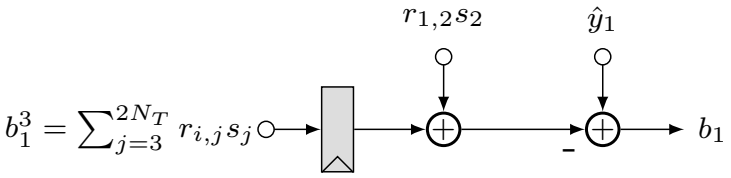
where b_1^2 represents the partial b_i value for level 1 that has been accumulated and stored into registers from level $2N_T$ down to level 2, that is, $b_1^2 = \sum_{j=2}^{2N_T} r_{i,j}s_j$. The complexity of the interference cancellation unit (ICU) can also be reduced by reusing the same $r_{i,j}s_j$ multipliers for different tree levels. For example, in level 1 of Table 3.3, five multipliers are required for computing $\sum_{j=2}^{6} r_{1,j}s_1$, while only a single multiplier is required in level 5. Overall, 15 multipliers are required for the direct implementation of $\sum_{j=1}^{6} r_{i,j}s_j$. However, since the levels are processed in different clock cycles, it would be possible to share the multipliers across different levels. In the proposed VPED implementation, all

Table 3.3: Conventional interference computation for $N_T = 3$

i	$\sum_{j=i+1}^{2N_T} r_{i,j}s_j$
6	N/A
5	$\underline{r_{5,6}s_6}$
4	$\underline{r_{4,6}s_6} + r_{4,5}s_5$
3	$\underline{r_{3,6}s_6} + r_{3,5}s_5 + r_{3,4}s_4$
2	$\underline{r_{2,6}s_6} + r_{2,5}s_5 + r_{2,4}s_4 + r_{2,3}s_3$
1	$\underline{r_{1,6}s_6} + r_{1,5}s_5 + r_{1,4}s_4 + r_{1,3}s_3 + r_{1,2}s_2$



(a) Conventional Horizontal PED computation



(b) Modified Vertical PED computation

Figure 3.13: Horizontal and vertical PED computations at level 1

the $r_{i,j} \times s_j$ entries in the same row of Table 3.3 are computed by the same multiplier, which reduces the total number of multipliers from 15, in the direct implementation, to 5 in the modified implementation.

Figure 3.13 illustrates the computation of b_i at level 1 using the conventional "horizontal" and modified "vertical" approaches. The block diagram of the ICU for a MIMO system employing $N_T = 4$ is shown Fig. 3.14, where several ICU blocks, each corresponding to a tree level, are connected in a feed-forward manner. The output of the ICU is sent as an input to the PED computation unit as shown in Fig. 3.15. The absolute value function, $|f(x)|$, in the figure, is implemented by checking the MSB of the result of $b_i - r_{i,i}s_i$ and taking the two's complement of the result if it is negative.

For complexity reduction, the multiplication $r_{i,i} \times s_i$ is implemented using adders and shifters as described in Table 2.2. Furthermore, only the absolute value, $|r_{i,i}s_i|$, is computed, which reduces the number of adders and shifters required. An XOR gate determines if the result of the multiplication, $r_{i,i} \times s_i$ is negative, by checking if the MSBs of $r_{i,i}$ are s_i are different. All possible $|r_{i,i}s_i|$ results are generated and a compare and select (C & S) block determines the correct output based on the value of $|s_i|$.

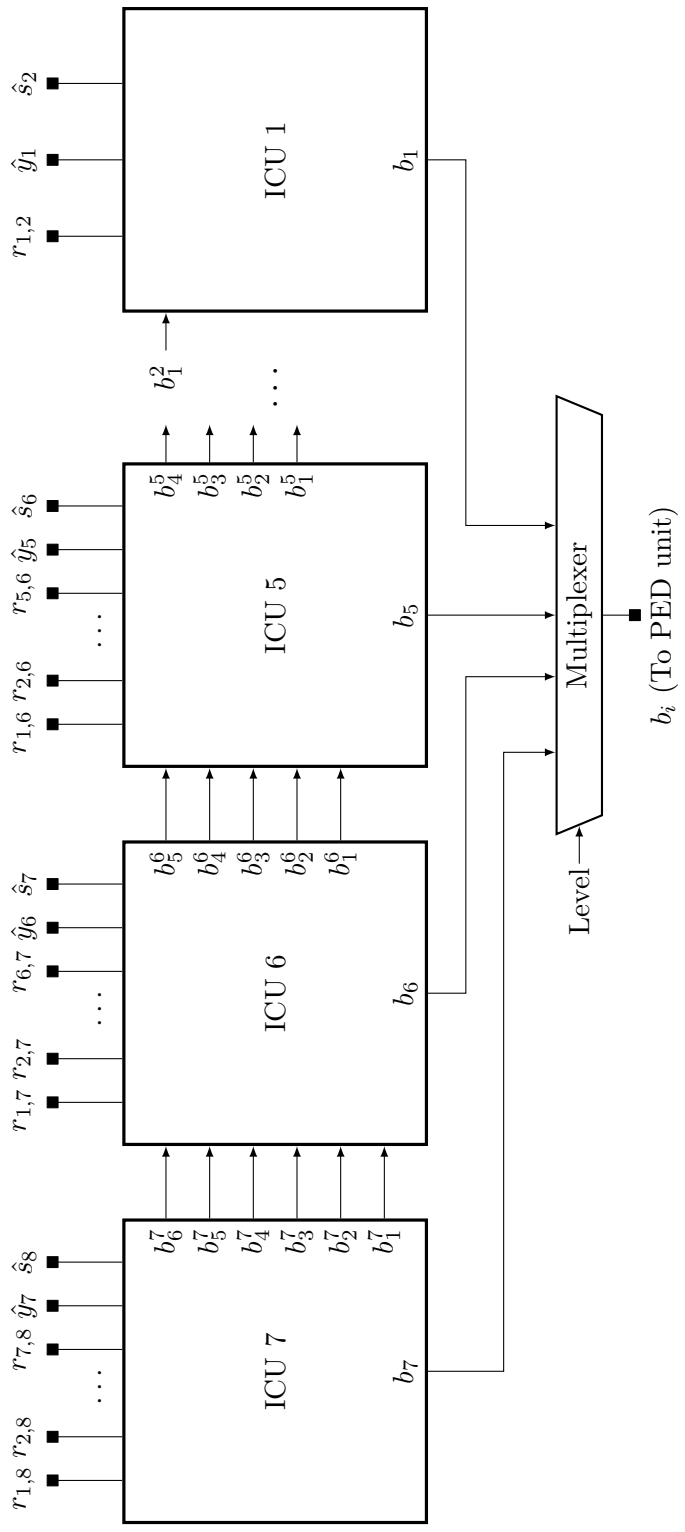


Figure 3.14: Interference cancellation unit for the sphere decoder. The ICU consists of several smaller ICU blocks, each of which computes the interferences for a given level. The ith ICU computes b_i as well as $b_j^i = r_{i,j}\hat{s}_{i+1}$ for $j = i-1, i-2, \ldots, 1$, which are then passed as inputs to the $(i-1)$th ICU.

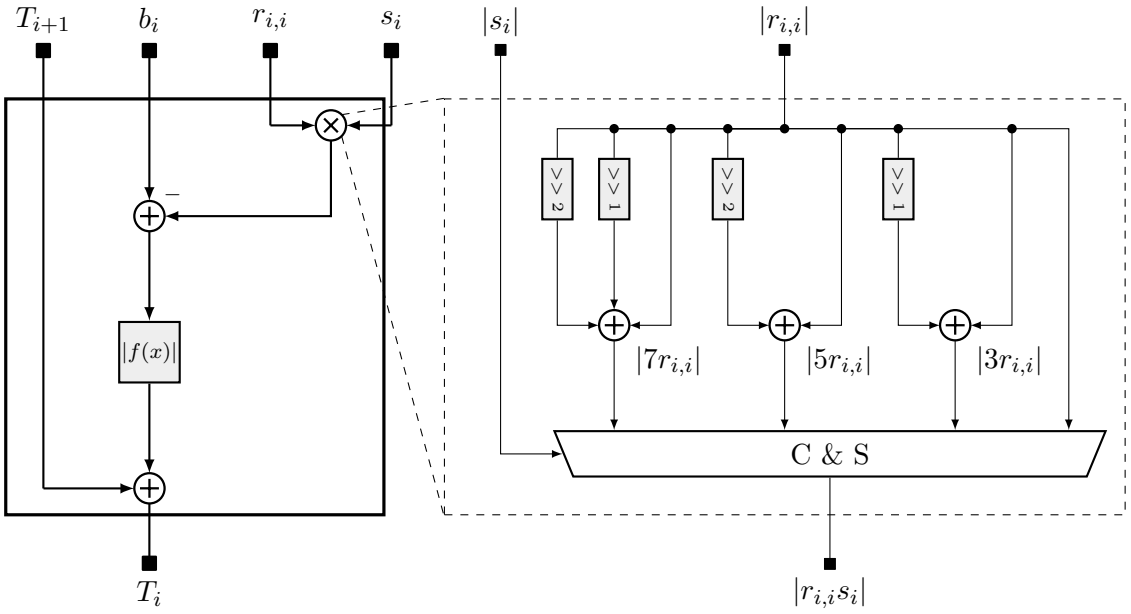


Figure 3.15: Architecture of the PED computation unit. The structure of the $r_{i,i} \times s_i$ multiplier is shown on the right. The multiplication is performed on positive symbols only. The result for negative symbols is obtained by taking the two's complement of the output of this module.

3.5 Results and Discussion

In this section, we will present the results of the VLSI implementation of the proposed SD for a MIMO system employing 64-QAM and 4×4 antennas. The results are shown in Table 3.4. Two different designs based on the conventional horizontal PED (HPED) computation and the modified vertical PED computation are presented. The designs are synthesised using the ST 65 nm CMOS technology at a supply voltage of 1.05 V. Both detectors are based on the ℓ^1-norm PED computation proposed in [45], and the look-ahead tree traversal, which has a negligible (less than 10%) impact on the complexity of the controller unit, and less than 1% overall area cost, compared with the conventional SD implementation.

As expected, the sphere decoder employing the vertical PED computation (SD-VPED) achieves an improved clock frequency compared with the conventional SD (i.e. SD-HPED). The proposed detectors achieve their maximum throughputs, $\Phi_{\max}$, at $D_{\max} = 1$, which corresponds to a latency of 18 clock cycles.[3] This is the number of clock cycles required to find the Babai point and corresponds with maximum throughputs of 275 Mbps and 344 Mbps for the SD-HPED and SD-VPED respectively. The throughput advantage of SD-VPED comes at the cost of a slightly higher power and area consumption,

[3]For $N_T = 4$, the top level is processed in 1 clock cycle, while the remaining 7 levels are processed using 2 clock cycles each. One further clock cycle is needed to check if the next constellation point at the topmost level violates the metric of the Babai point. Two clock cycles are required to transit to the next symbol vector.

Table 3.4: Implementation results for the sphere decoder for 4×4 MIMO. Power, throughput and energy-per-bit are scaled to the 65 nm 1.05 V technology according to Equations (2.22) and (2.24).

Detector	**SD-HPED**	**SD-VPED**	**EURASIP'05** [91]	**ESPC'06** [92]	**ISLPED'07** [68]	**GIMM'12** [93][a]	**ISCAS'13** [66][a]
Tech [nm]	65	65	FPGA	130	180	65	90
V_{dd} [V]	1.05	1.05	N/A	N/A	1.8	N/A	N/A
Modulation	64-QAM	64-QAM	16-QAM	16-QAM	16-QAM	64-QAM	64-QAM
Area [kGE]	59.71	68.82	N/A	90	175	N/A	153.93
Power [mW]	6.03	7.90	N/A	N/A	50	15	28.75
f_{clk} [MHz]	206	258	50	333	128	N/A	109
Φ_{max} [Mbps]	275	344	114.5	1522	443	58	569
E_{bit} [pJ/bit]	22	23	N/A	N/A	312.57	259	70

[a] Energy-per-bit and power consumption not scaled

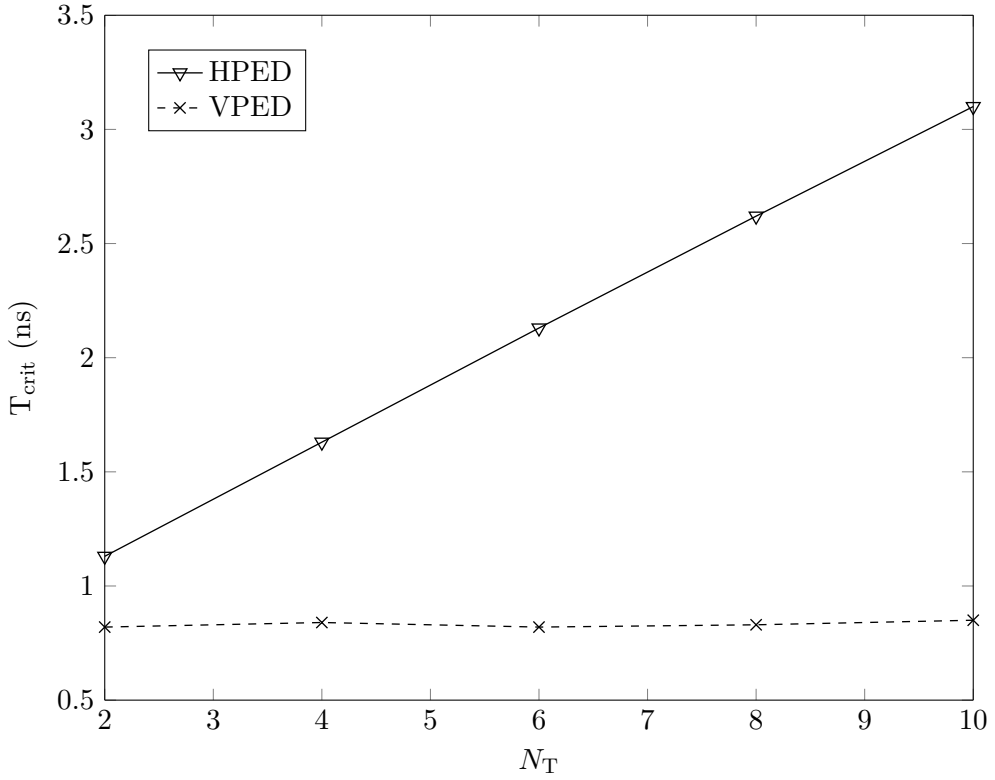


Figure 3.16: Critical path delay of b_i versus N_T [15]

which can be attributed to the extra pipeline registers required by the VPED technique. At the 4×4 MIMO configuration considered, the impact of the VPED technique appears quite marginal. However, at larger MIMO configurations, the improvement to the achievable maximum clock frequency due to the vertical PED computation could be expected to be more significant as shown in Fig. 3.16.

3.5.1 Comparisons with State-of-the-Art

Table 3.4 presents the results of other notable VLSI implementations of the sphere decoder for a 4×4 MIMO configuration. ISCIT'13 [66] uses a similar tabular enumeration method to the proposed implementation and achieves a higher maximum throughput. However, our implementation has the advantage of a lower area and power consumption. ISLPED'07 [68] applies parallel processing, where several input signals are processed at once, to achieve a maximum throughput of 443 Mbps. But this comes at the cost of a relatively high energy consumption per bit of 312.57 pJ/bit. ESPC'06 [92] achieves an impressive throughput of 1.5 Gbps through the use of a complex channel model, one-node-per-cycle architecture and parallel processing. The comparatively low throughput of our implementation is due to the multi-cycle node processing and single-tree processing, where independent RSVs are processed sequentially. The throughput of the proposed detectors can easily be improved almost two-folds by eliminating the pipeline

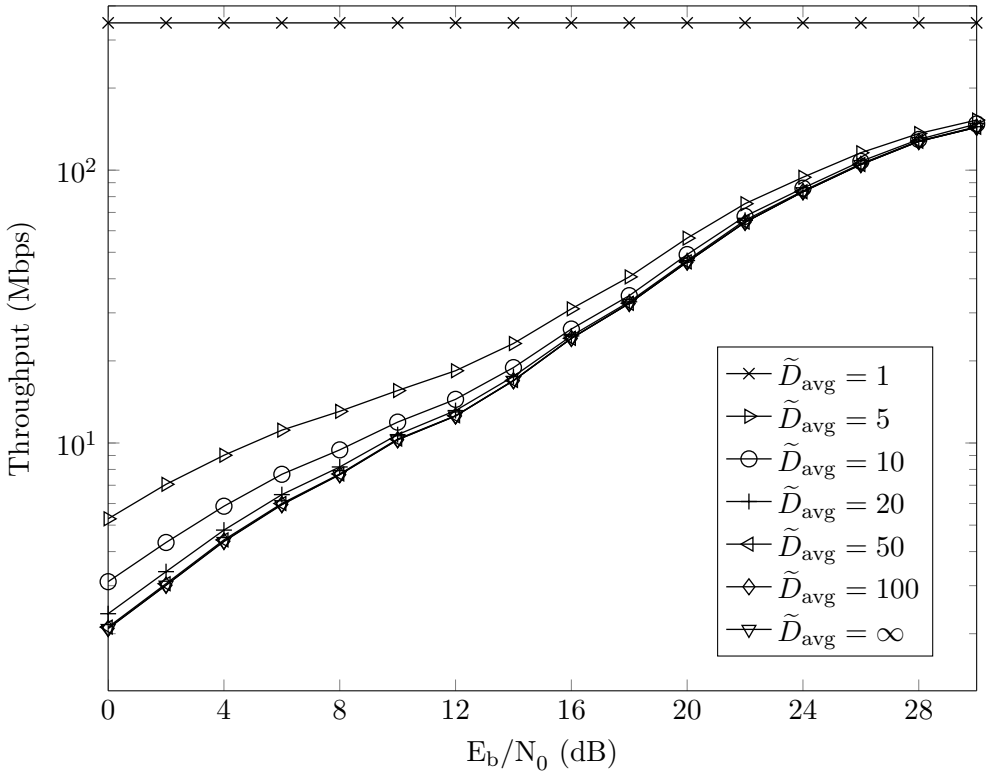


Figure 3.17: Throughput versus SNR for the sphere decoder using different values of $\widetilde{D}_{\text{avg}}$. At $\widetilde{D}_{\text{avg}} = 1$, the throughput is constant irrespective of the SNR.

register between the computations of b_i and T_i, which will reduce the number of clock cycles required for finding the Babai point from 18 to 11 clock cycles. As a result of the simple architecture, our proposed implementations achieve comparatively small area and power consumptions. The low energy-per-bit figures achieved also indicate that the implementations will be suitable for energy-efficient communications.

3.5.2 Impact of Runtime Constraints

Figure 3.17 shows the variation of the throughput of the SD with $\widetilde{D}_{\text{avg}}$. The hardware implementation of the SD with look-ahead is simulated using test vectors generated from MATLAB, and the throughput is computed according to (3.1). The number of clock cycles, N_{clk}, is obtained by counting all the state transitions required to detect one symbol vector, from the `S_IDLE` state to `S_DONE` as shown in Fig. 3.8. Unfortunately, the maximum achievable throughput also coincides with a performance degradation. In practice, several solutions will need to be explored in order to achieve an acceptable BER performance. Using $\widetilde{D}_{\text{avg}} = 10$, the average number of clock cycles is 509, corresponding to a throughput of 9.71 and 12.17 Mbps for the SD-HPED and SD-VPED respectively. As the SNR increases, the impact of the runtime constraints on the throughput reduces in significance. This is because the SD, especially when the Schnorr-Euchner enumeration is

employed, is quite efficient at finding the solution quickly, using its natural unconstrained tree search.

Going back to Fig. 3.5, we observe that the impact of the runtime constraints on the BER, at low SNR values, is not significant. For example, the SD with $D_{\max} = 1$ offers a near optimal performance up to an SNR of 14 dB for the Rayleigh fading channel considered. Therefore, $D_{\max} > 1$ is not required for SNRs below 14 dB. As such, we can adopt different runtime constraints for different channel conditions to achieve a more efficient detection. For example, if the runtime constraint is selected as follows:

$$\widetilde{D}_{\text{adaptive}} = \begin{cases} 1 & \text{if SNR} \leq 14\,\text{dB} \\ 10 & \text{if SNR} > 14\,\text{dB} \end{cases}$$

then the BER is similar as that of the SD using $\widetilde{D}_{\text{avg}} = 10$ throughout the SNR range considered. Selecting the runtime constraint thus, the average number of clock cycles is reduced from 509 to 63 clock cycles for the entire SNR range, corresponding to approximately 90% complexity reduction compared with the unconstrained SD. The average throughput of the SD using this adaptive scheme is 79 and 98 Mbps for the SD-HPED and SD-VPED respectively.

3.6 Summary and Conclusion

In this chapter, we have analysed the performance and throughput of the sphere decoder using different runtime constraints. Simulation results show that an acceptable BER performance can be achieved by applying runtime constraints to the SD, which also results in appreciable throughput gains. We have also presented a novel look-ahead tree search, which reduces the number of clock cycles required by the SD by about 25%, using a real channel model. Unlike the complex channel model, which increases the throughput of the SD at the expense of an increased complexity, the complexity cost of the proposed look-ahead technique is negligible. Furthermore, an optimised PED computation technique has been presented, where the interference terms of the PED are computed incrementally, which results in a constant critical path length irrespective of the number of transmit antennas employed. To improve the average throughput of the SD, we have also proposed the use of adaptive runtime constraints, which varies the maximum number of visited nodes depending on the SNR. The SD with the modified PED computation achieves an average and maximum throughput of 98 Mbps and 344 Mbps respectively, at a clock frequency of 258 MHz. In the next chapter, we will present the VLSI implementation of the K-best algorithm. Unlike the SD, the K-best algorithm features a forward-only tree-search, which results in a throughput that is constant irrespective of the channel condition.

Chapter 4

VLSI Implementation of a Single-Stage K-best Detector

4.1 Introduction

In Chapter 3, we presented the hardware implementation of the sphere decoder. The sphere decoder is notable for its ML performance and high throughput at high SNR. However, it also suffers from a severe throughput degradation at low SNR. To combat this, runtime constraints are applied in practice to terminate the sphere decoder within a more reasonable time frame. Unfortunately, this also degrades the performance of the SD, and as such, the ML performance is not achieved in practice.

The K-best algorithm [13], which carries out the MIMO detection using a breadth-first tree search was presented in Chapter 2 as an alternative to the sphere decoder. Due to its breadth-first search, the K-best detector is able to achieve very high throughput performances, since multiple tree nodes can be processed in parallel, and the tree search does not involve any back traces. Furthermore, because the number of visited nodes is fixed with respect to the channel condition, the K-best detector is able to avoid the throughput degradation problem suffered by the sphere decoder in low-SNR scenarios.

In this chapter, we will study the K-best algorithm in more detail and also present its hardware implementation. The K-best detector is classified into two basic architectures namely, single-stage and multi-stage architectures. The single-stage architecture is quite similar to the SD and will be the focus of this chapter. The multi-stage architecture will be presented in Chapter 5, and a detailed comparison between the two architectures, in terms of energy and hardware efficiency, will be presented. The main objectives of the chapter are as follows:

1. Exploit the inherent regularities of the K-best algorithm to implement a folded single-stage architecture, similar to the SD, where one processing element is reused across different tree levels in order to achieve low complexity.

2. Compare the hardware implementation results of different sorting algorithms. This will be used to select the most appropriate sorting algorithm for our proposed detector implementation.

3. Implement a low-complexity sorting algorithm suitable for high-throughput MIMO detection. The Batcher's sort algorithms [94] will be used for the implementation.

The chapter is organised as follows. In Section 4.2, the BER performance of the K-best detector is compared with the SD. A reduced-complexity K-best algorithm is also presented in this section. In Section 4.3, we present a number of sorting algorithms and compare their hardware implementations. In Section 4.4, the overall architecture of the proposed K-best detector is presented. Finally, the implementation results are presented in Section 4.5

4.2 Performance Analysis

In this section, we will compare the performance of the K-best detector with the sphere decoder. For the fixed-point representation of the K-best implementation, the same values as provided in Table 3.1 are adopted. We will also present the implementation of a reduced-complexity K-best algorithm and its impact on the BER performance.

4.2.1 Comparison with the Sphere Decoder

Figure 4.1 compares the floating-point performance of the K-best detector with the sphere decoder. Different K values and average runtime constraints, $\widetilde{D}_{\text{avg}}$, for the SD are considered, for a total of $200,000$ transmitted symbol vectors. The results clearly show that the sphere decoder outperforms the K-best detector in terms of the BER performance. This result is expected, as the K-best detector lacks the ability to make a back-trace to retry other possible solutions. In order to improve the performance of the K-best algorithm, a larger K value may be employed, or detection ordering may be applied to begin processing substreams with the largest signal to interference and noise ratios (SINR). In the figure, the K-best detector with $K = 32$ achieves a similar performance to the sphere decoder with $\widetilde{D}_{\text{avg}} = 10$. However, this large value of K has a negative impact on the area consumption, and potentially on the critical path, if a merge-sort procedure is employed for determining the best K nodes. For the proposed K-best detector, $K = 16$ is adopted,[1] which provides a reasonable performance with

[1]This choice of K also simplifies the sorting unit (see Section 4.3.5).

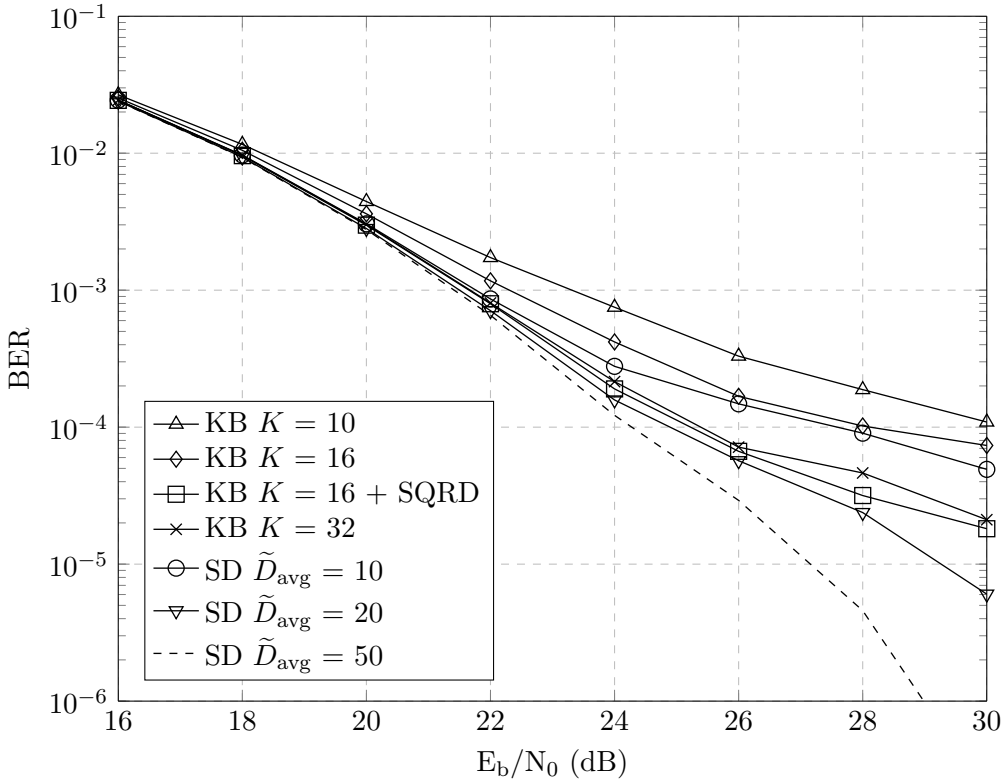


Figure 4.1: BER performance of the K-best algorithm versus SD for different values of K and $\widetilde{D}_{\text{avg}}$

much reduced complexity compared to $K = 32$. At a BER of 10^{-3}, the SNR loss of the K-best detector with $K = 16$ with respect to the SD with $\widetilde{D}_{\text{avg}} = 10$ is less than 1 dB. With a sorted QR decomposition [19] in the preprocessing stage, the K-best detector exhibits similar performance as the SD with $\widetilde{D}_{\text{avg}} = 20$ up to a BER of about 10^{-4}. The results indicate that with an appropriate choice of K, and preprocessing, the K-best detector can approach the performance of the SD in practice. In the next section, we will present a strategy for further reducing the complexity of the K-best detector.

4.2.2 Reduced-Complexity K-best Detector

The number of children per parent node in the tree search can be utilised as a parameter to reduce the complexity of the K-best algorithm. If the number of children per parent is denoted by λ, then the K-best detection can be characterised as $\text{KB}(K, \lambda)$, where $\text{KB}(K, \sqrt{M})$ represents the original K-best algorithm. If $\lambda = 1$, then the K-best detector essentially reduces to a SIC-based detection, where each of the K paths extends only its best child. A similar reduced-complexity K-best detector was proposed by Kim and Park [73], which was based on the ORVD channel model [38]. As a result of the channel model employed in that work, two layers are processed in parallel, which results in almost a $10\times$ increase in the number of PED increment computations compared with the proposed

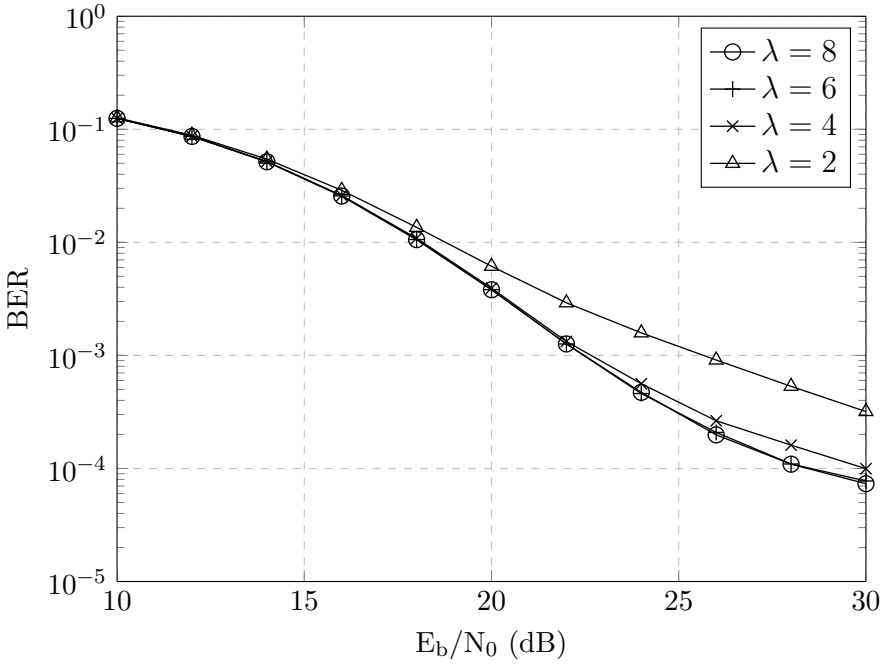


Figure 4.2: BER performance of the K-best algorithm with $K = 16$ using different values of λ

implementation. Furthermore, the ORVD incurs a BER penalty, as the resulting tree search is not equivalent to the conventional RVD-based scheme [39].

Figure 4.2 shows the BER simulation for the proposed K-best detector using $K = 16$ for different values of λ. The K-best detector using $\lambda = 4$ displays similar performance to the original K-best detector up to a BER of 10^{-3}. On the other hand, the K-best detector using $\lambda = 2$ suffers a significant SNR loss of approximately 3 dB compared with the original K-best detector at a BER of 10^{-3}. The result of Fig. 4.2 is noteworthy, as the choice of λ has an impact on the sorting complexity as will be discussed in the next section.

4.3 Sorting

Sorting plays a prominent role in many digital signal processing applications. In the K-best algorithm, sorting is required to select the best candidates at each level of the tree search. The choice of sorting algorithm has an impact on the complexity and performance of the K-best detector. Single-cycle sort algorithms are suitable for high-throughput applications (particularly for single-tree detection), while multi-cycle sort algorithms typically incur a lower complexity but at the expense of a longer latency. Table 4.1 shows a number of design trade-offs for different sorting algorithms. In the next sections, the sorting algorithms will be discussed in more detail.

Table 4.1: Design trade-offs for different sorting algorithms

Algorithm	**Bubble** [13]	**Distributed** [71]	**Relaxed** [15]	**Batcher's**
Time	$O(N^2)$	$O(K)$	$O(1)$	$O(1)$
Area	$O(N)$	$O(N)$	$O(N)$	$O(N(\log N)^2)$
Performance	Exact	Exact	Approx.	Exact

4.3.1 Bubble Sort

The bubble sort is one of the oldest and most well-known sorting algorithms. Given an unsorted list, $\mathbf{a}$, of length N, the bubble sort (BS) compares adjacent elements and swaps them if they are out of place. With each pass of the algorithm, the larger of the two elements compared is shifted towards the end of the list giving rise to a "bubble" like effect. An implementation of the Bubble sort algorithm is provided in [95] as follows:

```
for j = N-1 to 1
    for k = 1 to j-1
        if a[k] < a[k+1]
           swap(a[k], a[k+1])
        end if
    end for
end for
```

where a represents an input list having n elements, and $\mathbf{a}[k]$ is the kth element of $\mathbf{a}$. Due to its large $O(N^2)$ worst-case time-complexity, the bubble sort is rarely applied in practical systems requiring high throughput. However, the bubble sort is well suited to smaller constellation sizes, such as 4-QAM, which do not require large values of K.

4.3.2 Distributed Sort

The distributed sort algorithm [70] (also referred to as the winner path extension in [96]) was initially proposed by Wenk, Zellweger, Burg, *et al.* [37], as a low-latency alternative to the bubble sort for larger constellation sizes. The algorithm works by dividing the input list into groups, and then comparing the best element of each group in each iteration until the desired number of sorted elements is obtained. K iterations are required to obtain the best K candidates. The operation of the distributed sort (DS) algorithm is illustrated in Fig. 4.3. In the first cycle, the minimum-metric child of parent node, p_1 ($c_{1,1}$), is compared against the minimum-metric child of p_2 ($c_{2,1}$). In this case, $c_{1,1}$ is the winner and is disregarded in the next cycle and replaced by its next best sibling (i.e. $c_{1,2}$). The attractive feature of this method is that the number of clock cycles required to produce the K-best candidates depends only on the K value and is constant irrespective

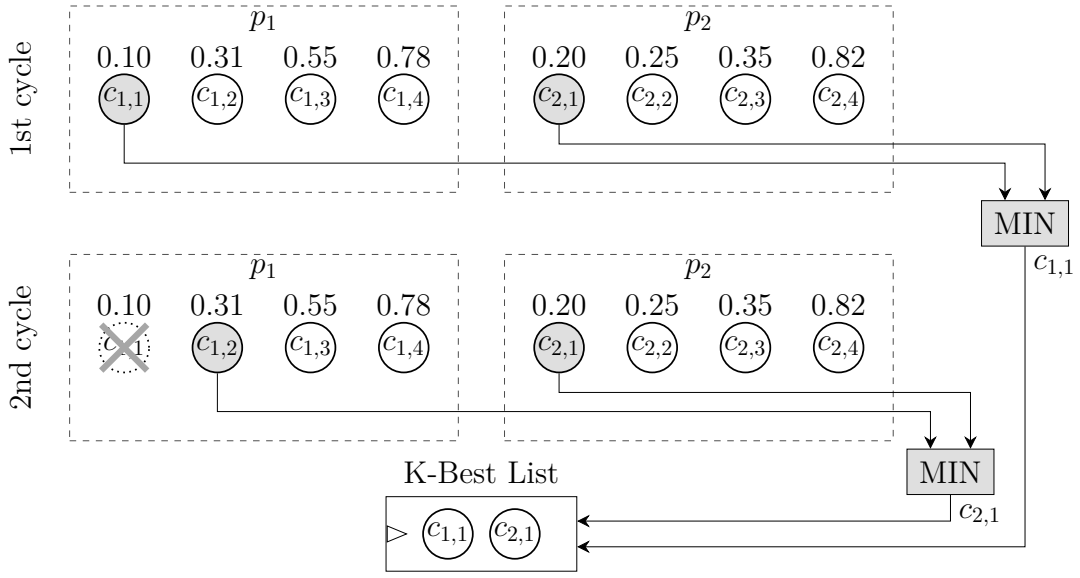


Figure 4.3: Distributed K-best sorting [70] for 16-QAM system with $K = 2$ showing two parent nodes p_1 and p_2. The respective PEDs are shown above each of the child nodes.

of the constellation size that is employed. The distributed sort algorithm can be implemented using an on-demand expansion, where the best child is computed only when required [71]. Initially, K best children are expanded from the K best candidates in the first iteration. In the subsequent $K-1$ iterations, the next best child of the winning path is extended, resulting in $2K-1$ path extensions overall. An alternative to on-demand expansion is tabular enumeration [67], which reduces the complexity of computing the next best child in each iteration through the use of a pre-computed lookup table.

4.3.3 Relaxed Sort

In order to reduce the complexity of the sorter, a "relaxed" sorting (RS) procedure may be adopted, which produces the result in an approximate order. One of such techniques was proposed by Kim and Park [73]. In this technique, the list is divided into K groups, and a minimum (MIN) search is carried out only for elements within the same group. The groups are formed in such a way as to ensure that the best candidates are retained until the next stage with a high probability. In order to avoid duplicate results, an element is only sent to one MIN unit. For the kth MIN unit, the inputs are selected according to $c_{((j-1)\,\%\,K)+1,i}$, where $j = k, k+1, \ldots, k+\lambda-1$, $i = 1, 2, \ldots, \lambda$, $c_{k,i}$ represents the ith child node of the kth parent, and % represents the modulus operation. This arrangement prevents the best children of the parent nodes from being compared against each other, which may result in the best path being lost during the sorting operation. An architecture for the relaxed sorter for a K-best detector employing $K = 4$ and $\lambda = 4$ is provided in Fig. 4.4. The MIN unit, shown on the right in the figure, successively compares two elements at a time, until the element with the smallest PED is obtained.

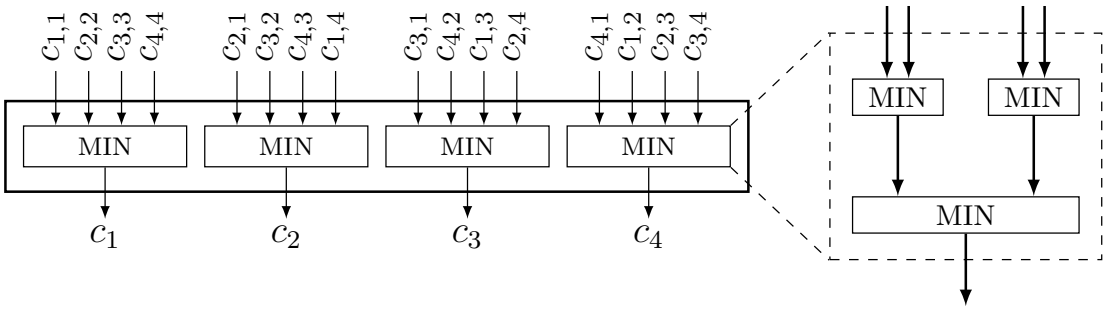


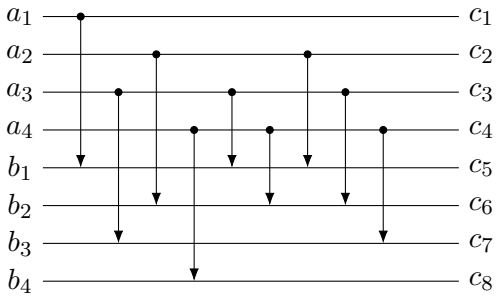
Figure 4.4: Architecture for a relaxed sorter with $K = 4$ and $\lambda = 4$ [15]. The 4-input MIN circuit is constructed from three 2-input MIN circuits to reduce the critical path length.

4.3.4 Merge Algorithms

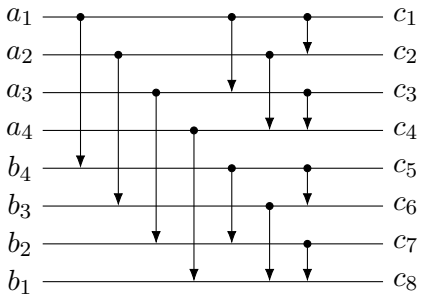
Traditional sorting algorithms, such as the bubble sort, require a large number of clock cycles, which is not suitable for high throughput applications. Merge algorithms on the other hand, sort a data sequence in a single step and as such the desired sorted result can be obtained within one cycle. Merge algorithms typically employ a "divide-and-conquer" approach to sorting, by dividing the data to be sorted into sublists and then merging the sorted sublists in parallel. After each iteration, a larger sublist is formed and the merge process is repeated until a single list is obtained, which corresponds to the sorted result. In this way, a merge network is formed whose depth is proportional to the number of sublists at the input. Unfortunately, this may also result in a large combinational delay, which limits the attainable maximum clock frequency. To solve this, pipeline registers can be inserted at selected points in the merge network to shorten the combinational delay from the input to the output. In this thesis, we will limit ourselves to the well-known Batcher's merge (BM) networks [94], which are discussed in the subsequent sections.

4.3.4.1 Odd-Even Merge

Given two length-N lists, $\mathbf{a} = [a_1, a_2, \ldots, a_N]$ and $\mathbf{b} = [b_1, b_2, \ldots, b_N]$, where $a_1 < a_2 < \ldots < a_N$ and $b_1 < b_2 < \ldots < b_N$, the odd-even sorter splits the two lists into their odd and even-indexed components and sorts them independently by comparing two items at a time and then swapping them if they are out of place. The first item of the odd merge is also the first item of the final result, while the last item of the even merge is the last item of the final result. After the odd and even lists have been sorted, the odd-even sorter iteratively compares the $(i+1)$th item of the odd merge with the ith element of the even merge to get the remaining items of the final length-$2N$ result. By comparing two sublists at a time, a larger merge network with 2^p inputs (where p is some integer) can be constructed. Figure 4.5(a) illustrates the odd even merge for two 4-item sublists, where each arrow represents a compare-and-exchange operation. The top and tip of each arrow correspond with the smaller and larger number of the comparison respectively.



(a) 8-input odd-even merge



(b) 8-input bitonic merge

Figure 4.5: Illustration of Batcher's merge algorithms

4.3.4.2 Bitonic Merge

Closely related to the odd-even merge is the bitonic merge (BM), which sorts one ascending list and one descending list, otherwise known as a bitonic sequence. Thus, $a_1, a_2, \ldots, a_N, b_N, b_{N-1}, \ldots, b_1$ forms a bitonic sequence if $a_1 < a_2 < \ldots < a_N$ and $b_N > b_{N-1} > \ldots > b_1$. If two lists are constructed as follows:

$$\begin{aligned} c_{\min} &= \min(a_1, b_N), \min(a_2, b_{N-1}), \ldots, \min(a_N, b_1) \\ c_{\max} &= \max(a_1, b_N), \max(a_2, b_{N-1}), \ldots, \max(a_N, b_1), \end{aligned} \tag{4.1}$$

then it can be shown that all the elements in the first list are less than the elements of the second list. Furthermore, each of $c_{\min}$ and $c_{\max}$ is bitonic. After the sequence is split into $c_{\min}$ and $c_{\max}$, the bitonic sorter compares two elements of each sublist at a time, swapping them if they are out of place. This process is continued iteratively until the final length-$2N$ list is obtained. Unlike the odd-even merge, which has unequal paths from the inputs to the outputs, the input-output lines of the bitonic merge all have equal lengths. However, the bitonic sorter requires more compare-and-exchange elements for a given input sequence, which increases its complexity in hardware.

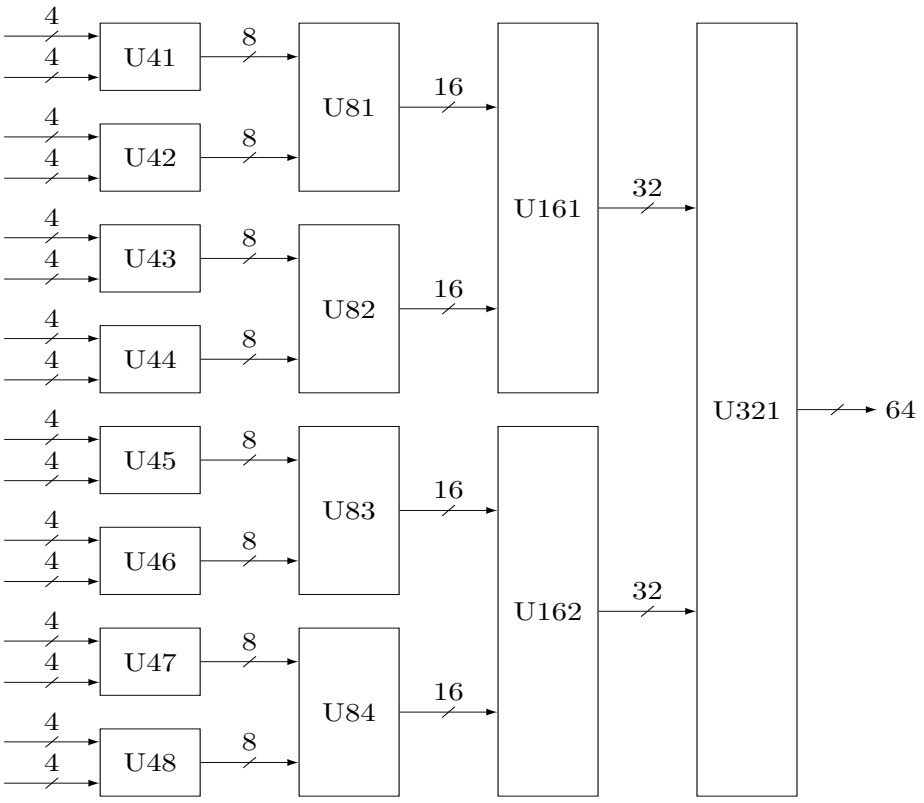


Figure 4.6: Full unoptimised merge network for 64 candidates. All the candidates are included in the final sorted result and the best K candidates are taken to the next level.

4.3.5 Implementation of the Merge Network

In this thesis, the Batcher's merge networks will be adopted, as they produce the sorted result within one clock cycle, which is suited to high-throughput applications. The odd-even merge is selected to construct the merge network due to its lower complexity compared to the bitonic merge. The merge network sorts the candidates in pairs of two λ-length sublists, where each candidate is organised as $(s_{i,j,k}, T_{i,j,k}, k)$, where k denotes the parent path, $s_{i,j,k}$ is the jth child of the kth parent after SE enumeration, and $T_{i,j,k}$ is its corresponding metric. Before the merge operation at the current level, k simply takes a value from the ascending sequence $1, 2, ..., K$. After the merge operation, k is updated according to the indices of the sorted PEDs at the current level.

Figure 4.6 illustrates the merge network for 16 sublists each comprising four elements. The merge units labelled U4X, U8X, U16X and U32X denote merge units for 4×4, 8×8, 16×16 and 32×32 inputs respectively. With $K = 16$, the candidates are merged in pairs of two by eight U4X units operating in parallel, and the outputs are successively doubled at every stage until the final 64 sorted result is obtained. Thereafter, the upper 16 results are selected and forwarded to the next level. The operation of the Batcher's merge

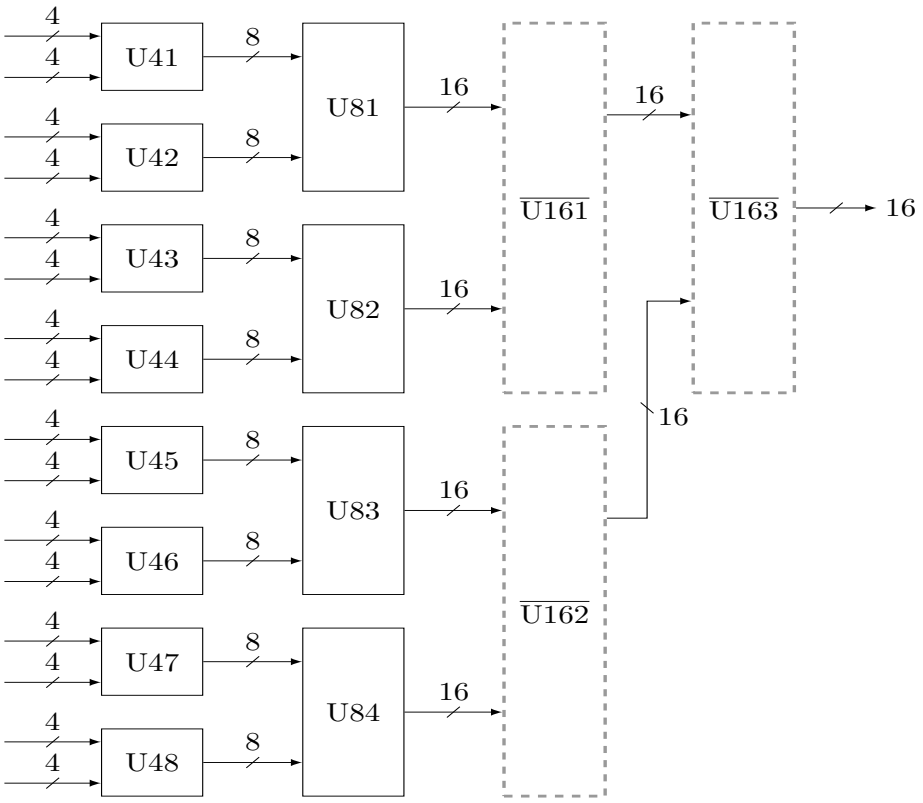


Figure 4.7: Optimised merge network for 64 candidates. The merge units in the third and final stages are modified to produce only the top 16 results, while discarding the bottom results. The implementation of the modified U16 merge units (indicated by the overline) is presented in Figs. 4.8 and 4.9.

network makes it more convenient to adopt K values that are powers of two; however, it is possible to construct a merge network with non-power-of-2 K values by simple architectural modifications. For example, to construct a merge network for $K = 10$, U46, U47, U48 and U84 can be discarded and the bottom inputs to U83 and U162 can be replaced with dummy candidates having $T_i = \infty$. These dummy candidates will be automatically relegated to the bottom of the 64-length sorted output at the end of the merge operation.

4.3.5.1 Area Optimisation

The merge network in Fig. 4.6 includes several redundant candidates in the final sorted result, which will eventually get discarded and play no further role in the detection process. This is inefficient and leads to an unnecessary increase in the detector complexity. Since only $K = 16$ candidates are required, U321 can be replaced with a simpler U16 unit as shown in Fig. 4.7. Similarly, U161 and U162 can be replaced with simpler U16

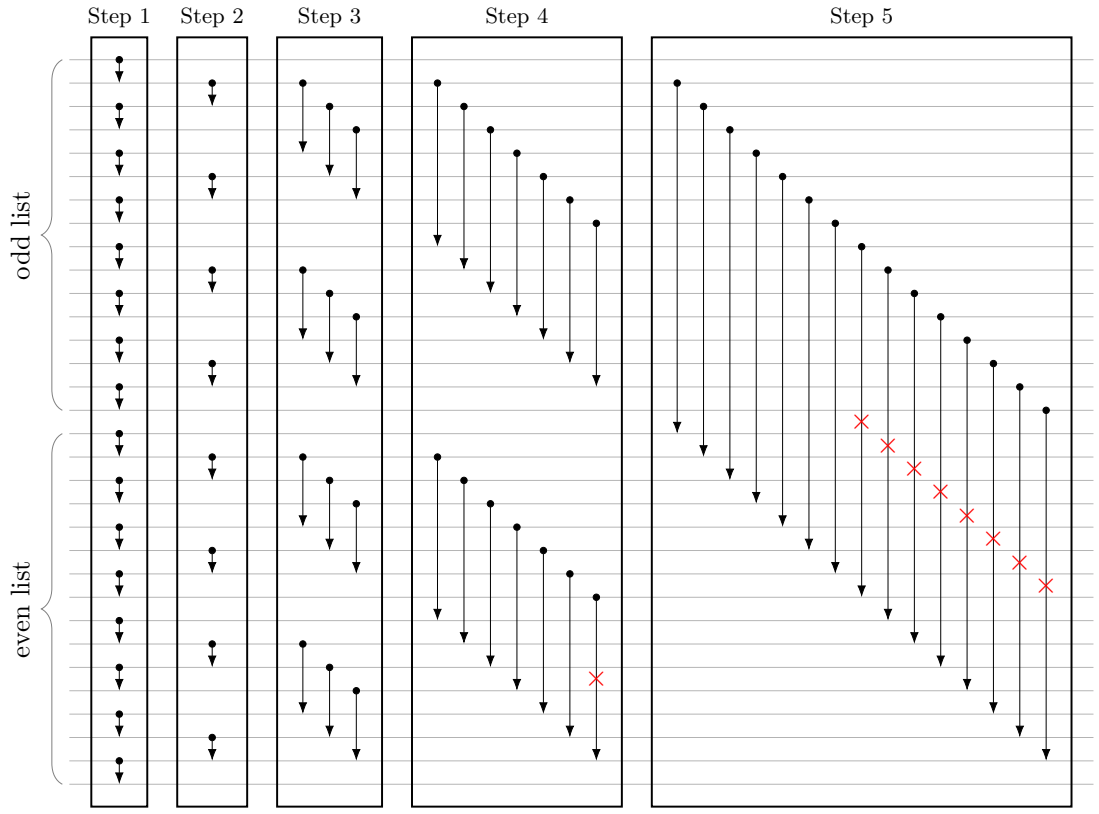


Figure 4.8: Modified U16 using the odd-even merge. Only 1 comparator can be eliminated from the bottom half of the 4th step given K = 16.

equivalents having only 16 outputs, instead of the 32 outputs in the original merge network. This also has a timing advantage, as the U16 element requires one less comparator stage compared with U32.

To construct the optimised U16 unit, all the comparators that do not contribute to the final output are eliminated. Interestingly, the bitonic sorter requires fewer comparators to implement the optimised U16 than the odd-even sorter. This is due to the unique property of the bitonic sorter whereby the upper and lower halves of the sorted result are generated in the first step of the merge process as shown in (4.1). Therefore, the lower half can be discarded early in the sorting, and the subsequent sort operations can be carried out on the upper half only. By contrast, although the odd-even sorter requires fewer comparators in general, the two halves of the final sorted result are only determined in the last stage.

The modified U16 unit, implemented using the odd-even and bitonic merge units, is shown in Figs. 4.8 and 4.9 respectively, showing the individual comparator operations. While the bitonic U16 unit is able to eliminate up to 32 comparators, the odd-even U16 unit is only able to eliminate 9 comparators. Thus, a hybrid merge network can be constructed with a bitonic merge for the third and final stages and odd-even merge for the remaining stages. The optimised U16 unit with 16 outputs is denoted by $\overline{\text{U16}}$. Overall, the hybrid merge unit achieves an area saving of approximately 30% compared

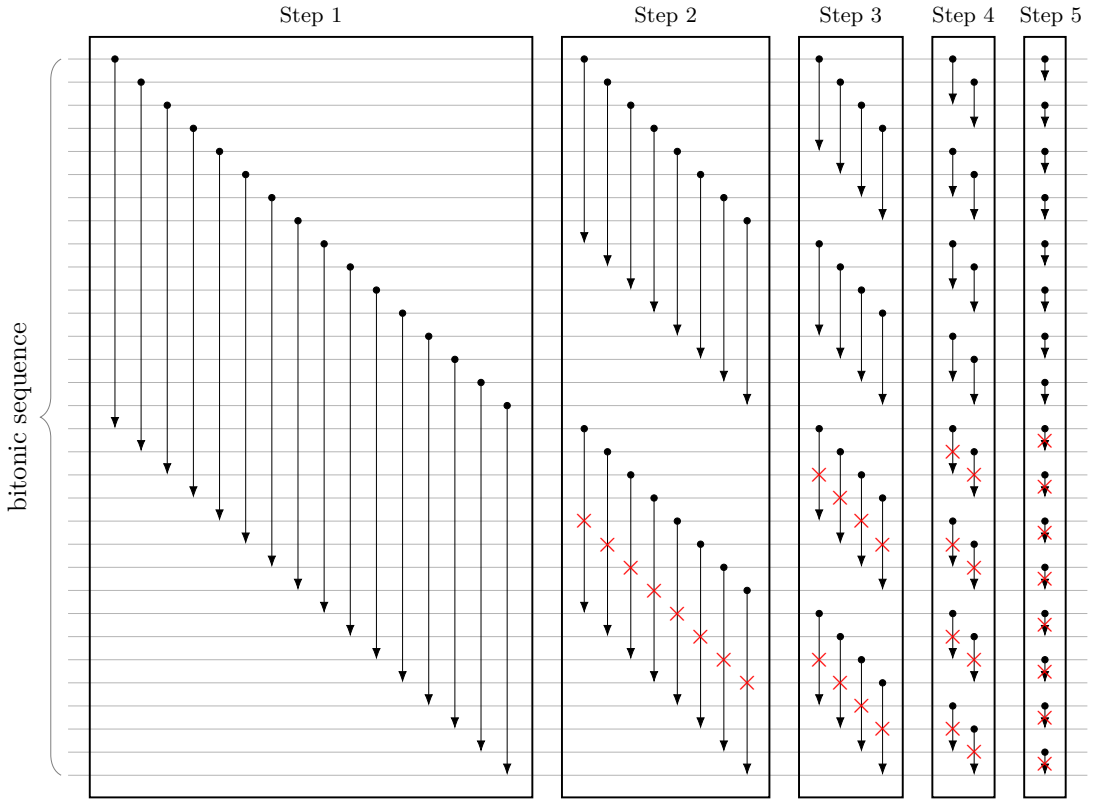


Figure 4.9: Modified U16 using the bitonic merge. All the comparators in the lower halves of steps 2 to 5 can be eliminated using the bitonic algorithm given K = 16.

with the unoptimised network using odd-even merge for all the stages. It should be noted that the merge unit at level $2N_T - 1$ is further reduced since the total number of available candidates is $\sqrt{M} \times \sqrt{M}$ rather than $K\sqrt{M}$ in the lower levels. The optimised hybrid merge unit is compared with the unoptimised bitonic and odd-even merge networks in Table 4.2.

4.3.5.2 Pipelining the Merge Network

The inputs to the proposed merge network described in the previous section pass through a total of 12 comparators in series before emerging at the output. This results in a large combinational delay, which limits the attainable clock frequency of the detector. The combinational delay of the merge network can be reduced by inserting one or more pipeline registers at suitable locations, thereby splitting the merge network into two or more smaller merge networks. This will however entail an increase in the area consumption of the design. Furthermore, the latency of the detection for one signal vector is increased by one clock cycle. However, this is compensated by an increased clock frequency, which allows a higher throughput to be achieved overall. If pipeline registers are inserted after the first step of the U16 blocks in the 3rd stage of the merge network in Fig. 4.7, then the merge network can be split into two nearly identical merge networks with 7 and 5 comparators.

Table 4.2: Comparison of full and optimised bitonic and odd-even merge networks for $K = 16$ and $\lambda = 4$

Sorter	**BM**		**OM**		**HM**
Implementation	Full	Optimised	Full	Optimised	Optimised
Area [kGE]	114.7	72.7	93.2	69.7	62.6
Comparators	576	432	463	340	316

4.3.5.3 Results and Discussion

Table 4.3 compares the hardware implementation results of the proposed hybrid merge (HM) network with other sorting algorithms. The relaxed sorter incurs the smallest area consumption, however, this is at the expense of a degradation to the BER performance. Despite its simplicity, the bubble sorter incurs a relatively large area, which is as a result of the registers required for storing the temporary sorting results as well as the input elements. Furthermore, a counter needs to be created to keep track of the number of iterations, which is the largest among the algorithms compared. The distributed sort algorithm was implemented with the assumption that all the child nodes are already enumerated via a table lookup. In order to find the MIN element in each iteration, assuming $K = 16$, two elements are compared at a time, resulting in a critical path of 8 comparators in series. This makes the distributed sort to have a longer critical path length than the bubble sort, which only needs to compare two elements in each cycle. As expected, the proposed hybrid merge incurs the largest area consumption and longest critical path. The application of pipelining increases the area by 15.5%, while reducing the critical path length by 55.3%, which is almost the same as that of the distributed sort algorithm.

The comparatively large area of the bubble sort, relative to the distributed and relaxed sort algorithms, as well as its large latency, further underscores its unsuitability to the K-best algorithm. For small constellation sizes, however, such as 4-QAM, the bubble sort is ideal since only a few iterations are required to get the final sorted result. The distributed sorter is suited to larger constellation sizes since the number of clock cycles required depends on K only. However, it should be noted that larger constellation sizes tend to require larger values of K since the BER performance degrades with the modulation order. Where area is not critical, the hybrid merge appears to be attractive as it achieves an exact sorting and also produces the sorted results within a single cycle. As noted, the hybrid merge can be pipelined in order to reduce the critical path at the expense of a larger area and additional clock cycles to obtain the sorted results. In the next section, the hardware implementation for the K-best detector, based on the pipelined hybrid merge network, will be presented.

Table 4.3: Comparison of hardware implementations of different sorting algorithms

Sorter	**BS**	**DS**	**RS**	**HM**	**HM (Pipelined)**
Area [mm^2]	0.085	0.034	0.010	0.13	0.15
Area [kGE]	40.83	16.21	5	62.6	72.37
T_{crit} [ns]	1.78	4.43	3.43	9.50	4.25

4.4 Hardware Implementation

The K-best algorithm can be implemented using different architectures depending on various criteria as summarised in Table 4.4. The classifications of tree search algorithms according to the pruning strategy, channel model and K value, in the case of the K-best detector, have already been discussed in Chapter 2. It should be mentioned that these classifications are not orthogonal as a single K-best implementation can incorporate these techniques jointly, in order to meet the design specification. In this thesis, we are more interested in the implementation of the K-best algorithm according to the mapping of the tree search to the processing elements, as well as the number of tree searches that the detector can execute concurrently. This classification results in two main architectures namely: single and multi-stage. The number of stages that the K-best architecture has, influences the number of tree searches that can be executed concurrently. The proposed detector in this chapter is implemented using a single-stage architecture, which we discuss in more detail in the next section.

4.4.1 Single-Tree Single-Stage Architecture

The operations at each level of the K-best algorithm are quite similar, and with suitable multiplexing, it is possible to reuse the same computational elements for all levels of the tree search, which results in a similar architecture to the SD as shown in Fig. 4.10. The figure shows a single-stage detector for a MIMO system employing a real-valued channel model, which requires $K \times 2 \times N_T$ symbol registers, which are served by a single processing element. The detector takes $\mathbf{R}$ and $\hat{\mathbf{y}}$ as inputs, which are assumed to be computed by an external processor. In contrast to the SD, K nodes instead of a single node, are processed in parallel in each iteration of the tree search. We will refer to this architecture as *single-tree single-stage* (STSS), since only one tree is processed at a time and the architecture utilises only one stage. The K-best algorithm can also be implemented using a multi-stage architecture, which is presented in Chapter 5. The folded architecture results in a much smaller detector compared to the multi-stage architecture and allows the same detector to be reused for different MIMO configurations, with only minor architectural modifications. The architecture for the proposed STSS detector is demarcated into a datapath, which comprises the main arithmetic units and a controller, which is implemented as an FSM.

Table 4.4: Classification of K-best MIMO detector architectures

Criteria	Description	Architecture	Example
Pruning	This describes how non-viable solutions are eliminated from the tree search. Fincke-Pohst K-best requires a zero-forcing estimate at the preprocessing stage.	**Fincke-Pohst**	Wong, Tsui, Cheng, *et al.* [13]
		Schnorr-Euchner	Wenk, Zellweger, Burg, *et al.* [37]
Channel model	This describes whether the input signal and channel matrix are complex or real. This has an impact on the hardware complexity, as well as the throughput especially in single-tree detectors.	**Real**	Shabany and Gulak [71]
		Complex	Mahdavi and Shabany [97]
The value of K	If the same value of K is used for all stages.	**Constant**	Shabany and Gulak [71]
		Non-constant	Moezzi-Madani and Davis [46]
Pipelining	This describes how many input vectors are serviced simultaneously.	**Single tree**	Kim and Park [73]
		Multiple trees	Huang and Tsai [77]
Tree mapping	This describes how the tree search is mapped to hardware resources.	**Single stage**	Moezzi-Madani, Thorolfsson, Chiang, *et al.* [98]
		Multi-stage	Wong, Tsui, Cheng, *et al.* [13]

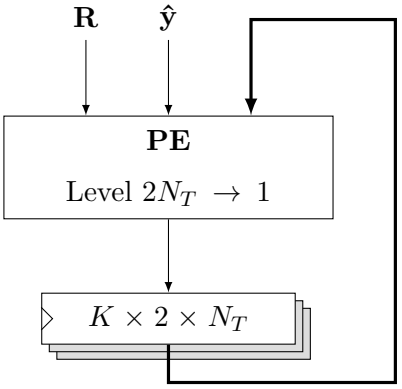


Figure 4.10: Single-tree single-stage architecture

4.4.2 Controller

The ASM chart of the controller, annotated with relevant datapath operations, is shown in Fig. 4.11. The states of the FSM are implemented using one-hot encoding, which simplifies the state decoding operation. The controller has seven states as follows:

- `S_IDLE`: The detector is put in the idle state by an asynchronous system reset, which is de-asserted synchronously. In this state, the level indicator is set to $2N_T$. The controller transits to `S_TOP_PED` upon receiving a new frame of data, indicated by the `frame_ready` signal.

- `S_TOP_PED`: In this state, all the constellation points are expanded to form the initial $\sqrt{M}$ paths, which will be expanded to K paths in the subsequent level. No sorting operation or interference computation is carried out in this state.

- `S_BI`: The interferences of previously detected symbols along all K paths are computed in this state. In level $2N_T$-1, only $\sqrt{M}$ b_i results need to be computed, since there are only $\sqrt{M}$ paths from level $2N_T$. The `en_BI` signal is asserted to enable the ICU's registers. At $i = 2N_T$-1, only $\sqrt{M}$ ICUs are asserted, while K ICUs are asserted in subsequent levels. The complexity of the b_i computation is higher in later stages.

- `S_PED`: In this state, the children of all the K parents are enumerated using a table lookup and expanded by the PED units. SE enumeration is required to present a presorted candidate list to the merge network. Both the PED computation and the sorting are carried out in the same cycle. The `en_PED` signal is asserted to enable the registers in the PED units. This is not necessary for the correct functioning of the circuit, however, but it is required to reduce power consumption.

- `S_PED`$^+$: Due to the pipelining of the merge network, an extra cycle is required to get the final sorted result. In this state, no PED or SE enumeration operations are executed. If the current level is the last, the controller asserts a

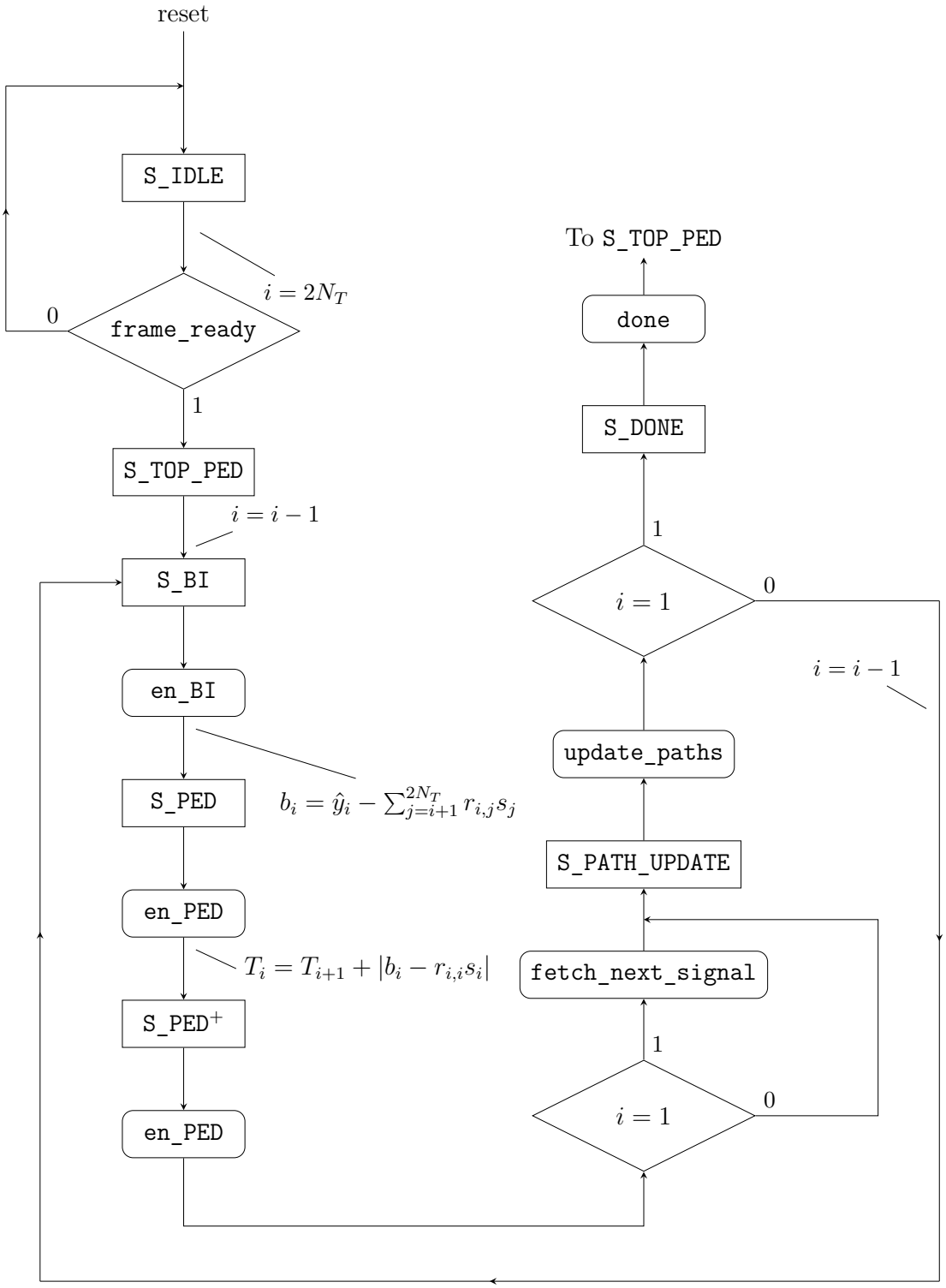


Figure 4.11: Simplified ASMD chart for the single-stage K-best detector

`fetch_next_signal` flag, which loads the next signal vector and its corresponding channel entries into memory. In the proposed implementation, the next signal needs to be fetched and loaded into memory 2 cycles before the end of the current processing.

- `S_PATH_UPDATE`: In this state, the detector updates the previously detected symbols, that is, $[\hat{s}_{2N_T}, \ldots, \hat{s}_{i+1}]^T$ for each path, according to the newly sorted results at the current level. Path updating is only carried out from $2N_T$-1 to 1 since no sorting is carried out at level $2N_T$. For hard detection, only the best path needs to be updated at the last level. An `update_paths` signal is asserted, which instructs the datapath to update the symbol registers according to the sorted result determined in the previous state. The controller also checks to see if the last level has been reached in this state: if the last level has been reached then the controller transits to the final state, otherwise, it decrements the level counter and goes back to `S_BI`.

- `S_DONE`: This is the last state of the proposed K-best controller. At each path update stage, the controller checks if the last level has been reached, in which case, it transits to this state. If `frame_ready` is asserted (not shown in the figure), the detector returns to `S_TOP_PED` to begin processing the next RSV. The latency for one symbol vector can be expressed as

$$N_{\text{clk}} = \sum_{i=1}^{2N_T} C_i, \tag{4.2}$$

  where C_i is the number of clock cycles required to process the ith level. In the proposed detector, the constellation set at the beginning of the search is expanded in one cycle, which implies that $C_{2N_T} = 1$. For the remaining levels, $1 \leq i < 2N_T$, $C_i = 4$. An extra cycle is required for transiting to the next signal vector. Thus, the total latency for one MIMO symbol vector is 30 clock cycles for 4×4 MIMO.

4.4.3 Datapath

The datapath receives control signals from the controller, which synchronises its operations with the relevant states of the FSM. The datapath of the K-best detector is quite similar to that of the SD, with the notable exception of a sorter, which is necessary to generate the K-best results and a path update unit for updating previously detected symbols according to the sorted results at the current level. The datapath of the K-best detector is explained in more detail in the subsequent sections.

4.4.3.1 Processing Element

The processing elements (PE) execute the main arithmetic operations of the detector and constitute the main blocks of the datapath. The general architecture of a PE is shown in Fig. 4.12, and comprises the merge network, implemented as described in previous sections, and K expansion units operating in parallel. Each expansion unit comprises an

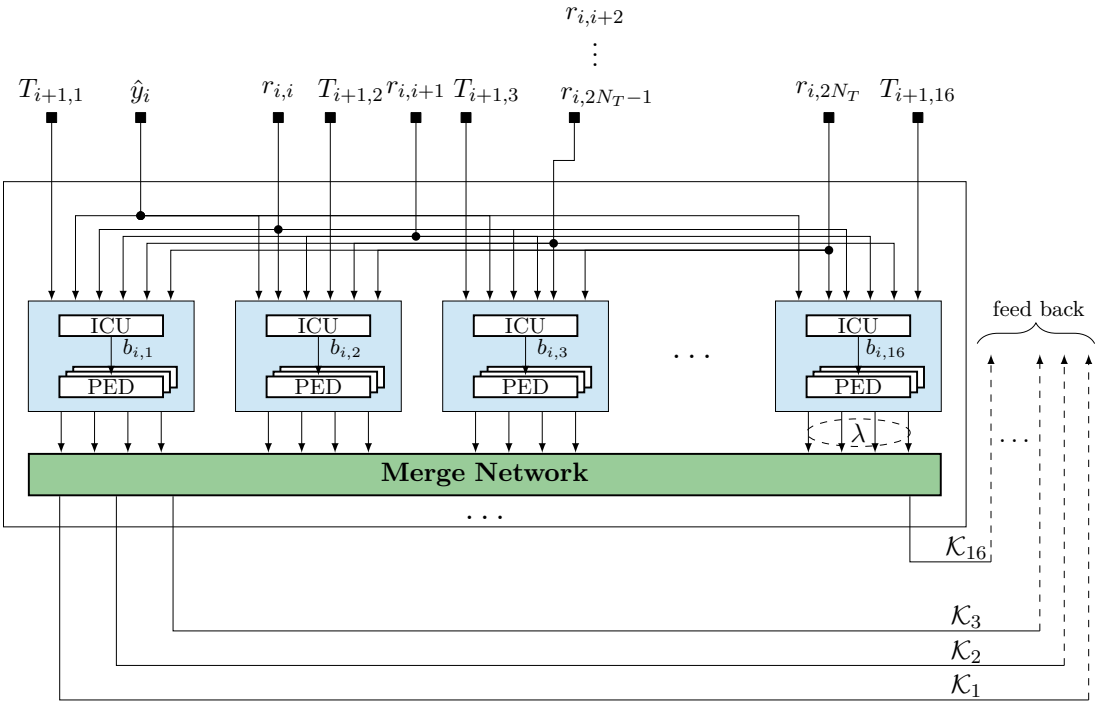


Figure 4.12: General architecture of a K-best processing element

interference computation unit and λ PED units. The merge network produces a sorted candidate list, denoted by $\mathcal{K} = [\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_{16}]$, where each candidate is formed from a concatenation of a symbol, its PED and its parent path, as $\left(s_{i,j,k'}, T_{i,j,k'}, k'\right)$, where k' is the path index after sorting. In the single-stage architecture, the sorted candidates are fed back to the input of the processing element for subsequent processing. It should be noted that the level indicator (not shown) also needs to be sent to the PE from the controller so as to correctly compute the PEDs for that level.

In the previous chapter, two different PED computation techniques were presented based on how the interference terms are computed. For the K-best detector, the VPED technique proposed for the SD cannot be directly applied since the K-best paths change according to the sorted results, which potentially makes the partial b_i results accumulated at a previous level unusable for the currently detected symbol. A technique modifying the VPED for the K-best detector was proposed in [15], where all the K interference terms are computed incrementally and fed to a total of K $K-$to-1 multiplexers for the computation of the next level interference terms. The modified computation of b_i for the K-best detector at level 1 is shown in Fig. 4.13. However, for the proposed K-best detector, the critical path is along the merge network, and not in the computation of the interference terms, as such, the proposed K-best detector is implemented based on the conventional HPED approach described in the previous chapter.

Like the SD, the nodes in the K-best algorithm are visited according to an SE enumeration, which is precomputed and stored into a lookup table. The architecture of the SE enumeration unit for the K-best detector is the same as that of Fig. 3.12, with the

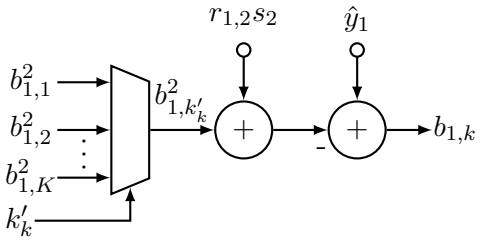


Figure 4.13: Modified Vertical PED computation block for the K-best detector proposed in [15]. The partial PED result for the PED computation at level i, accumulated up to the jth level and for the kth path, is denoted as $b^j_{i,k}$. $\mathbf{k}' = [k'_1, k'_2, \ldots, k'_K]$ represents the list of sorted indices of the K-best candidates from the previous level.

exception that only $\lambda = 4$ children are enumerated according to the reduced-complexity K-best algorithm proposed in Section 4.2.2. The PED of the K-best detector is also computed using the ℓ^1-norm proposed by Burg, Borgmann, Wenk, *et al.* [45], which achieves an appreciable reduction in complexity compared with the original Euclidean-norm-based computation at the cost of a small penalty to the BER.

4.4.3.2 Path Update Unit

After the candidates have been sorted, a path update operation is necessary to reorder the previous K-best paths according to the PEDs at the current level. If the path index k at level $i+1$ is equal to the updated path index, k'_k, then the kth symbol registers, $[\hat{s}_{i+1,k} \ldots \hat{s}_{2N_T,k}]^T$, retain their values, where $\hat{s}_{i,k}$ is the detected symbol at the ith level and kth path. This simply means that the kth best path in level $i+1$ remains the kth best path in level i after sorting. However, if $k \neq k'_k$, then the contents of the kth symbol registers are updated with the contents of the k'_k symbol registers in the next clock cycle.

Figure 4.14 illustrates the architecture of the path update unit. A feedback line is connected to the outputs of all the symbol registers and sent to a bank of multiplexers, which are fed back to the input of the symbol registers. The sorted path indices list is then connected to the multiplexers, which connects the new path indices to the appropriate symbol registers depending on the current level of the tree search. Observe that the path update operation ends at $\hat{s}_2$ in the figure. This is because there are no further sorting operations to alter the computed K-best list in level 1. During the path update operation, all the symbol registers below the current level, $\hat{s}_{i-1}$ to $\hat{s}_1$, are disabled in order to reduce power consumption.

4.5 Results and Discussion

The proposed single-stage K-best architecture is implemented for a 64-QAM 4×4 MIMO system. The design is synthesised using Design Compiler targeting the 65 nm CMOS

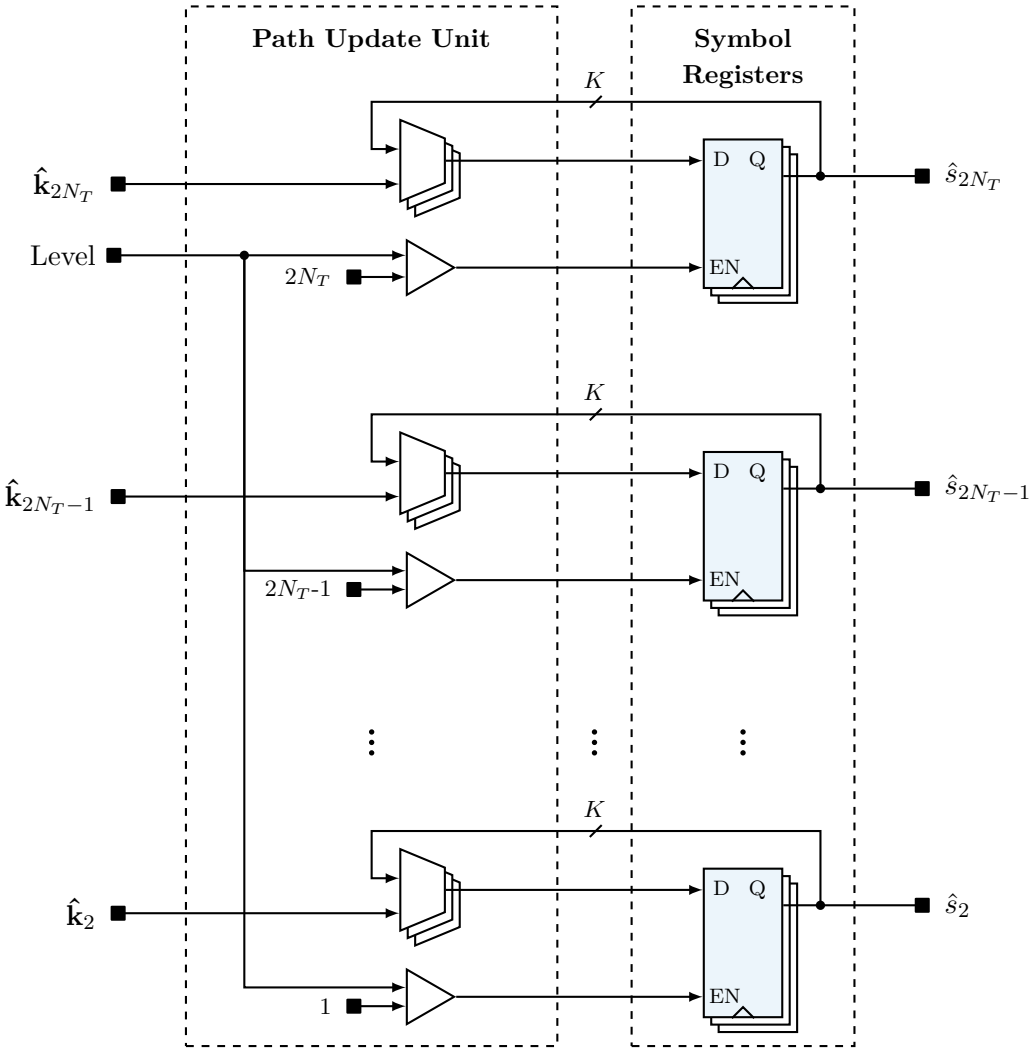


Figure 4.14: Path update operation for the K-best detector

technology and place-and-route is performed using Cadence Encounter. The results are provided in Table 4.5. The throughput of the K-best detector is computed as:

$$\Phi = \frac{\text{Total number of bits detected}}{\text{Latency of one signal vector}}$$

$$= \frac{N_T \times Q \times f_{\text{clk}}}{N_{\text{clk}}}, \tag{4.3}$$

where Q is the number of bits in a QAM symbol and N_{clk} is the latency required to process one signal vector. Timing analysis post place-and-route indicate that the detector is able to operate at a clock frequency of 137 MHz, which results in a throughput of about 109 Mbps. This throughput falls short of the gigabit data rates required by recent communications standards, such as the IEEE 802.11ac. In the next section, a technique for increasing the throughput of the single-stage K-best detector will be described.

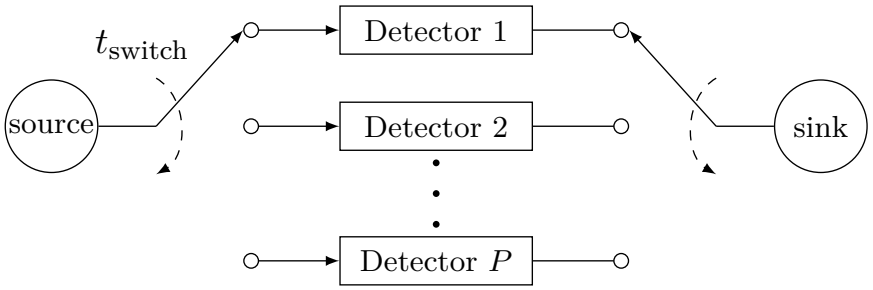


Figure 4.15: Illustration of detector interleaving

4.5.1 Detector Interleaving

The throughput of the single-stage K-best detector is limited to how fast it can accept a new signal vector. If N_{clk} is large, then the throughput deliverable by one detector is limited. To solve this problem, a complex channel model may be employed, which reduces the tree depth by half. The detector can also be operated at a higher frequency, but this is limited by the critical path constraints of the circuit, which may not always be improved. Operating the detector at a higher clock frequency may also have a detrimental effect on the power consumption.

Another alternative is to combine several detector units to form a larger detector as illustrated in Fig. 4.15 [73]. If P is the number of detector units, and Φ_{detector} is the throughput deliverable by a single detector, then the throughput of the interleaved detector is given as:

$$\Phi_{\text{interleaved}} \approx P \times \Phi_{\text{detector}}.$$

For example, to achieve 3120 Mbps, which is the maximum uncoded data rate for the IEEE 802.11ac in the 64-QAM configuration, approximately 30 detector cores will be required. Usually, the interleaved throughput will be less than $P \times \Phi_{\text{detector}}$ due to the delay experienced when switching from one received signal to the other. Theoretically, if the switching delay, t_{switch}, between one RSV to another is equal to 0, then the full interleaved throughput of $P \times \Phi_{\text{detector}}$ can be achieved. This however does not take into account the delay required at the output to arrange the detected symbol vectors according to their arrival times. However, if the delay is equal to N_{clk}, then the throughput is practically the same as that of a single detector unit and no interleaving advantage is obtained.

4.5.2 Comparison with the Sphere Decoder

In Chapter 3, the throughput of the SD was shown to be affected by the application of runtime constraints. Like the K-best detector presented in this chapter, the SD processes a single MIMO tree search at a time. At some arbitrarily high SNR, the SD will achieve its maximum throughput of 344 Mbps, which corresponds with $D_{\text{max}} = 1$ and a latency of

18 clock cycles. Unfortunately, this throughput is achieved at the expense of a reduced performance. By applying adaptive runtime constraints ($D_{\max} = 1$ below 14 dB and $\widetilde{D}_{\text{avg}} = 10$ above 14 dB), the SD achieves an average throughput of approximately 100 Mbps, which is slightly less than the throughput of the proposed single-stage K-best detector.

In Fig. 4.1, we see that the K-best detector achieves a performance close to the SD, especially with the application of SQRD at the preprocessing stage. Furthermore, the K-best detector achieves a higher average throughput than the SD, which is also unaffected by the SNR. Like the K-best detector, the throughput of the SD can be improved by interleaving, however, the interleaved SD is complicated as the symbol vectors may be generated out-of-order (i.e. in a different order to which they were received), or even received by the "sink" concurrently if 2 or more symbol vectors terminate at the same time. Furthermore, the per-block runtime constraints can no longer be applied, since the actual runtimes of previously-detected symbol vectors, which are required by (3.3), are not known. However, the SD has the advantage of a smaller area required to achieve a given throughput target.

Overall, this result suggests that the K-best detector is more suited to high throughput applications than the SD. This is despite the fact that the SD proposed in Chapter 3 achieves almost a two-fold maximum clock frequency advantage over the K-best detector. This result differs from the findings of Burg, Borgmann, Studer, *et al.* [64], where it was suggested that the SD achieves a higher throughput compared to the K-best. However, the comparison in their study was based on a complex SD versus the K-best detector based on a real channel model [37]. Since the channel model has an impact on the throughput, a fair comparison between the SD and the K-best detector should be with respect to a common channel model. To some extent, the application of pipelining can make the effects of the channel model less significant as will be explained in the next chapter.

4.5.3 Comparison with State-of-the-Art

The proposed detector is also compared with some results of interleaved single-tree single-stage architectures from the literature. All the implementations are targeted to a MIMO system employing 64-QAM. The proposed detector achieves a better energy efficiency and lower power consumption compared to TVLSI'07 [99] and TVLSI'10 [96]. TVLSI'07 [99] adopts a radius-based pruning strategy and as such does not have a fixed throughput. Despite running at almost twice the clock frequency of the proposed implementation, the detector of TVLSI'07 [99] achieves a throughput of 100 Mbps at a much larger area consumption than the proposed single-stage detector. With 3 detector cores, TVLSI'10 [96] achieves a larger area than our proposed implementation. However,

Table 4.5: Implementation results of single-stage K-best detectors for 64-QAM MIMO. Power, throughput and energy-per-bit are scaled to the 65 nm 1.05 V technology according to Equations (2.22) and (2.24).

Reference	**This work**		**TVLSI'07 [99]**†	**TVLSI'10 [96]**	**DATE'10 [75]**	**ISCIT'15 [15]**
# Cores	1	3	N/A	3	1	1
K	16	16	64	64	10	10
MIMO	4×4	4×4	4×4	4×4	2×2	4×4
Detection	Hard	Hard	Soft	Hard	Soft	Hard
Tech. [nm]	65	65	130	65	130	65
V_{dd} [V]	1.05	1.05	1.2	1	1.5	1.05
Area [kGE]	285	855	N/A	1760	24	206
f_{clk} [MHz]	137	137	270	158	287	287
Φ [Mbps]	109	327	100	100	214	300
P' [mW]	34.2	102.62	459	237.6	13.3	27.18
E'_{bit} [pJ/bit]	312	312	4594	2376	124.56	90.6

† Throughput is SNR dependent

their implementation achieves a better normalised area consumption,[2] $\frac{kGE}{K}$, which can be attributed to the merge networks employed by the proposed detector, which occupy almost 30% of the total area.

DATE'10 [75] achieves a throughput of 214 Mbps for a 2×2 antenna configuration with a relatively small area of 24 kGE. By comparison, our implementation requires 14 clock cycles to completely detect one symbol vector in the 2×2 configuration resulting in a better throughput performance of 234 Mbps. Assuming the proposed implementation consumes the same power in 2×2 as the 4×4 case, our implementation achieves an energy-per-bit of 145.62 pJ/bit, which is not significantly higher than that of DATE'10 [75], but at a higher throughput. Although ISCIT'15 [15] achieves a higher throughput than the proposed implementation, this is at the cost of a reduced BER performance due to the relaxed sorting algorithm that was employed.

4.6 Summary and Conclusion

In this chapter, the K-best algorithm has been implemented in hardware based on a single-tree single-stage architecture, which has a fixed number of computational units irrespective of the number of antennas. In order to achieve a high throughput, a pipelined hybrid merge network, combining the Batcher's odd-even and bitonic sort algorithms, was implemented, which produces the sorted results within two clock cycles. Detector interleaving has also been presented as a technique for increasing the throughput of the single-stage K-best detector. The proposed detector achieves a competitive energy efficiency compared to existing single-stage K-best detectors.

Compared with the SD, the K-best detector incurs a larger area, and in general, exhibits an inferior BER performance. However, the application of simple preprocessing techniques, such as SQRD, as well as the requirement of runtime constraints by the SD, closes the performance gap between the K-best detector and the SD. Furthermore, the K-best detector has a fixed throughput irrespective of the SNR, which is attractive for real-time hardware applications. The throughputs of both the SD and K-best detector can be improved by detector interleaving, however, the variable nature of the SD makes it less suited to this procedure. In the next chapter, the multi-stage architecture for the K-best algorithm will be presented, which will enable multiple-tree processing as well as a higher throughput to be achieved.

[2] The K value determines the sizes of the symbol registers, so the area of the K-best detector can be assumed to vary according to K. However, a linear relationship will not hold true for all cases, due to the differences in architectures.

Chapter 5

VLSI Implementation of a Fully-Pipelined K-Best Detector

5.1 Introduction

In Chapters 3 and 4, the VLSI implementations of the sphere decoder and the single-stage K-best detector were presented. Both detectors featured a single-tree processing, which limits their throughputs to the number of clock cycles required to traverse a search-tree. This is particularly problematic for the real channel model, which has been adopted in this thesis, as the depth of the tree is effectively doubled, which reduces the throughput by approximately 50% compared to the complex channel model. As a result, the SD and the single-stage K-best detectors are only relevant for applications requiring throughputs of a few 100s of megabits-per-second.

In this chapter, a fully-pipelined K-best detector, which processes several tree searches concurrently will be presented. The proposed detector achieves a throughput of one MIMO symbol vector per clock cycle, which makes the channel model employed less consequential to the final throughput unlike the detectors presented in the previous two chapters. In fact, the real channel model appears advantageous for such an architecture, since MIMO detectors based on a real channel model generally exhibit a much simpler datapath than corresponding complex channel implementations. The primary objectives of this chapter are as follows:

1. To implement a fully-pipelined K-best detector using a multi-stage architecture and explore ways of reducing its complexity
2. To compare the hardware and energy efficiencies of the interleaved implementation of the single-stage K-best detector, presented in the previous chapter, and a fully-pipelined single-core K-best detector

3. Evaluate the potential application of the proposed fully-pipelined K-best detector to recent high-speed wireless communication standards

This chapter is organised as follows. In Section 5.2, we will present implementation aspects of the pipelined K-best detector. In Section 5.3, the overall architecture of the proposed fully-pipelined K-best detector is presented. In Section 5.4, the implementation results of the proposed fully-pipelined K-best detector is presented. Furthermore, the potential application of the proposed detector to recent high-throughput wireless local area network standards is discussed. The chapter is concluded in Section 5.5.

5.2 Multiple-Tree Multi-Stage Architecture

In the previous chapter, a single-stage architecture for the K-best detector was presented, where the levels of the search-tree are mapped to a single processing element, which is shared among the tree levels in a time-multiplexed fashion. This architecture results in a small implementation that is not significantly affected by the number of antennas. More conventionally, however, the K-best detector is implemented using a multi-stage architecture, such that a processing element is uniquely mapped to a level of the tree as shown in Fig. 5.1. The multi-stage architecture can be utilised to process multiple tree searches concurrently, which is a significant advantage over the SD and the single-stage K-best detector. This architecture can therefore be referred to as multiple-tree multi-stage (MTMS) and supports the pipelining of the K-best detector as explained in the next sections.

5.2.1 Pipelining the K-Best Detector

Pipelining is a well-established technique for improving the throughputs of digital circuits by processing multiple data concurrently [100]. Pipelining as used in this chapter differs from the circuit-level critical path optimisations, where a large combinational circuit is split into two or more small combinational blocks, with the aid of pipeline registers. Both types of "pipelining" enable a faster circuit to be realised, but in this chapter, we are more interested in the former case, which allows an even more dramatic improvement to the throughput of the K-best detector based on the multi-stage architecture.

In a pipelined system, the circuit is partitioned into several pipeline stages, such that at each cycle, a pipeline stage is executing an independent instruction. The system incurs a latency at the beginning of the processing, where no output is produced. However, after the pipeline is full, the circuit is able to generate one output in every clock cycle. Although the latency required for a single instruction remains unchanged, compared to the unpipelined circuit, the throughput of the circuit is increased due to the concurrent

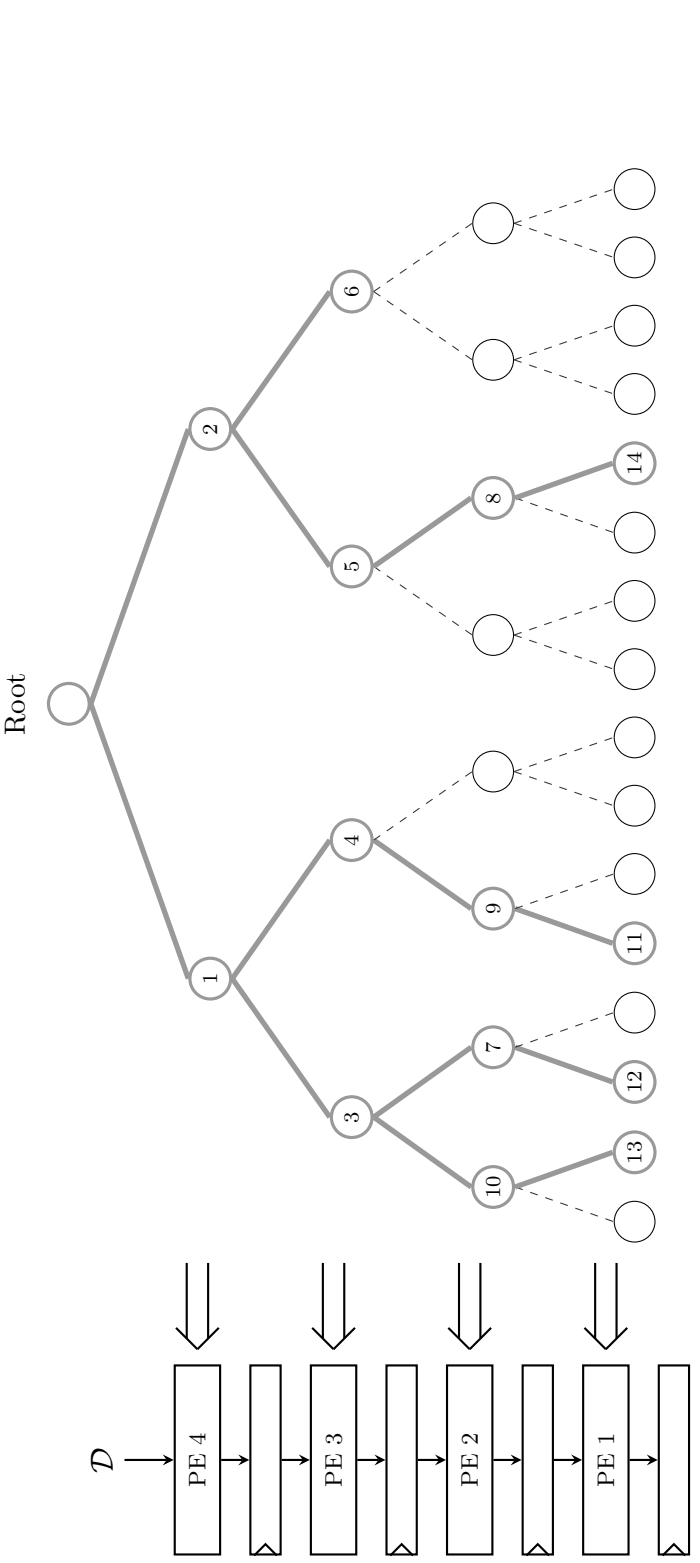


Figure 5.1: K-best multi-stage architecture with corresponding tree-mapping. The numbers within the nodes indicate a possible tree traversal. Each level of the tree is mapped to a processing element.

Signals	Time slots												
	1	2	3	4	5	6	7	8	9	10	11	12	13
$\hat{\mathbf{y}}^{(1)}$	PED	INFR	PED	PAU	INFR	PED	PAU	INFR	PED	PAU	PED		
$\hat{\mathbf{y}}^{(2)}$		PED	INFR	PED	PAU	INFR	PED	PAU	INFR	PED	PAU	PED	
$\hat{\mathbf{y}}^{(3)}$			PED	INFR	PED	PAU	INFR	PED	PAU	INFR	PED	PAU	PED
$\vdots$				$\ddots$									$\vdots$
$\hat{\mathbf{y}}^{(10)}$										PED	INFR	PED	PAU

Figure 5.2: Pipeline schedule for the K-best detector

processing of multiple instructions. We can apply this same concept to the K-best detector, by operating on several RSVs concurrently, where each RSV is at a different stage of its tree search. This results in an improved throughput performance compared with the detectors of the previous chapters. In the next sections, different strategies for pipelining the K-best detector will be discussed.

5.2.1.1 Fine-Grained Pipelining

Figure 5.2 shows the pipeline schedule for a fully-pipelined K-best detector. The operations of the K-best algorithm are completely unrolled such that no single operation takes more than one clock cycle to execute. For simplicity, the pipeline schedule is shown for a MIMO system employing two antennas at the transmitter. The shaded regions indicate the initial period when the pipeline is unfilled. After the pipeline is full, the detector outputs one symbol vector in every clock cycle.

In the first clock cycle, the topmost level PED for the first signal vector, denoted by $\hat{\mathbf{y}}^{(1)}$, is computed. In the second clock cycle, the interference (INFR) due to the expanded nodes at the topmost level, $\hat{s}_4$, is cancelled from the current signal, $\hat{\mathbf{y}}_3^{(1)}$, according to (2.16), in order to compute b_3. Meanwhile, the detector begins to concurrently expand the topmost level symbols for the next signal vector, $\hat{\mathbf{y}}^{(2)}$, which also marks the beginning of level 2 for the first signal, $\hat{\mathbf{y}}^{(1)}$. In the third cycle, the PED of the first signal vector is computed simultaneously with the interference computation for the second signal vector. The end of level 3 for $\hat{\mathbf{y}}^{(1)}$ is marked by the path update operation (PAU), which reshuffles the previously detected symbols, $\hat{s}_4$, according to the PED results of level 3. In this stage, the initial $\sqrt{M}$ paths from the root node are expanded to K paths, by selecting the best K paths from $\sqrt{M} \times \sqrt{M}$ candidates (or $\sqrt{M} \times \lambda$ candidates if a subset of the child nodes of a parent, $\lambda \leq \sqrt{M}$, is considered). This process is continued until

Signals	Time slots														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\hat{\mathbf{y}}^{(1)}$	PED	INFR	PED	PED^+	PAU	INFR	PED	PED^+	PAU	INFR	PED	PAU	PED		
$\hat{\mathbf{y}}^{(2)}$		PED	INFR	PED	PED^+	PAU	INFR	PED	PED^+	PAU	INFR	PED	PAU	PED	
$\hat{\mathbf{y}}^{(3)}$			PED	INFR	PED	PED^+	PAU	INFR	PED	PED^+	PAU	INFR	PED	PAU	PED
$\vdots$				$\ddots$											$\vdots$
$\hat{\mathbf{y}}^{(12)}$													PED	INFR	PED

Figure 5.3: Pipeline schedule using pipelined merge unit

the level 1 path update operation for $\hat{\mathbf{y}}^{(1)}$ at $t = 10$, which also signals the beginning of the last signal, $\hat{\mathbf{y}}^{(10)}$.

The start of every new signal vector is marked by a PED operation at level $2N_T$, in which the constellation points, $\mathcal{D}$, are expanded. This is indicated by the underline, PED, in the figure. To fill the pipeline, such that one symbol vector is detected in every clock cycle, there must be as many concurrent RSVs as pipeline stages. Apart from the topmost level, each level comprises three operations namely: INFR, PED and PAU. Thus, for a MIMO system employing $N_T = 2$, and using a real-valued detection, $1 + (2N_T - 1) \times 3$ or 10 clock cycles are required to process one signal vector.

If the merge unit is pipelined, as described in Section 4.3.5.2, then an extra cycle is required for the merge operation. This extra cycle is denoted by PED^+. In this cycle, the partially sorted candidates prior the U16 pipeline registers, are moved to the final stage of the merge network. No PEDs are actually computed in this cycle. Although pipelining the merge network increases the clock frequency, it also increases the number of clock cycles required to process one RSV, and thus, the time required to fill the pipeline. Figure 5.3 shows the modified pipeline schedule using the pipelined merge network, where it can be seen that the latency for one signal vector is increased from 10 to 12 clock cycles. It should be noted that further pipelining of the merge unit will lead to an increase in the number of pipeline stages and thus, the time required to achieve a full pipeline. The throughput deliverable by this architecture, after the pipeline is filled, is given as:

$$\Phi_{\text{pipeline}} = N_T \times Q \times f_{\text{clk}}.$$

Therefore, the throughput is no longer dependent on the latency, and is thus, only slightly affected by the channel model adopted. Since the complex channel model presents a more complicated datapath, and potentially a lower maximum clock frequency, this result suggests that the real channel model is more suitable for a fully-pipelined implementation.

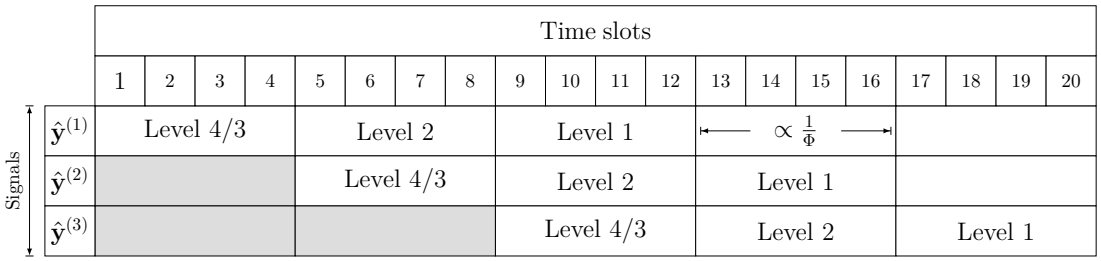


Figure 5.4: Pipelining at tree level. The throughput is inversely proportional to the number of clock cycles required to process a tree level.

The fully-pipelined architecture requires a large amount of hardware resources due to the number of RSVs processed. If a smaller area consumption is desired, and the throughput requirement is not stringent, it is possible to pipeline at a more coarse level, for example, with respect to the levels of the search tree. This is explained in the next section.

5.2.1.2 Coarse-Grained Pipelining

In the *coarse-grained* pipelining approach, a PE corresponding to a particular level is *fully engaged* to an RSV until that level is completely processed. This is unlike the *fine-grained* pipelining, presented in the previous section, where a single PE processes different RSVs concurrently, with each RSV at a different stage of that particular level. However, since a tree level typically requires several cycles, the throughput of this architecture will not achieve the processing rate of the fine-grained pipelined implementation, which achieves one MIMO symbol vector per clock cycle.

Figure 5.4 shows the pipeline schedule of the modified pipelined detector. It should be noted that if the number of clock cycles per level are unequal, then the pipeline stages may need to be balanced, in order to prevent stalling in the processing of any RSV.[1] For example in the figure, the top level (where only a PED computation is executed) can be collapsed into the second level such that $T_{2N_T-1} = T_{2N_T} + |b_i - r_{i,i}s_i|$ is computed over 2 cycles instead of 3 cycles. Thus, only $(2N_T - 1)$ concurrent signals are required, and a symbol is generated after every C_1 clock cycles, where C_1 is the number of clock cycles required to process the first pipeline stage. For the pipelined merge network adopted, $C_i = 4$, for all $i \in \{1, 2, \ldots, 2N_T\}$. Thus the throughput of the detector can be computed as:

$$\Phi = \frac{N_T \times Q \times f_{\text{clk}}}{4},$$

which is just a quarter of the throughput deliverable by the fine-grained implementation. An obvious advantage of the coarse-grained approach is that power consumption

[1] If the pipeline stages are unequal, No Operations (NOPs) will need to be introduced into the pipeline in order to prevent two or more instructions/RSVs from accessing a pipeline stage at the same time.

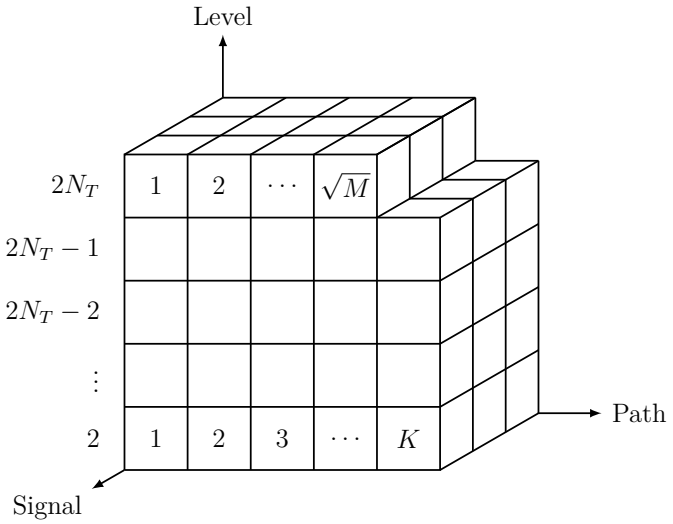


Figure 5.5: Memory layout for T_{i+1}

can be reduced since the pipeline processes fewer RSVs per time. However, in the proposed multi-stage architecture, we will adopt the fine-grained approach, described in the previous section, due to its ability to achieve the gigabit data rates required by future communication systems. The overall architecture of the fully-pipelined detector is described in the next section.

5.2.2 Implementing the Fully-Pipelined K-Best Detector

Implementing the K-best detector using the fine-grained pipelining technique described in Section 5.2.1.1 allows the K-best to be fully pipelined such that each pipeline stage processes only one RSV. A pipeline stage in the fine-grained pipelining technique performs only a single task, which takes only one cycle to complete before the pipeline stage switches to the next RSV. Based on this observation, we can identify two architectures for implementing the fully-pipelined detector depending on how the independent RSVs access the computation elements.

5.2.2.1 Multiplexer-Based Pipelined K-Best Detector

Due to the multiplicity of signals that need to be processed in the pipelined detector, certain registers need to be replicated for each of the concurrent signals. This is depicted graphically in Fig. 5.5 for the T_{i+1} register, where the register replication results in a three-dimensional memory layout, with a depth that is equal to the number of independent signal vectors. Each row represents the T_{i+1} inputs to the subsequent processing element. For example, the last row represents the best PEDs at level 2, which are passed as inputs to level 1. Each row has K columns corresponding to the K-best paths,

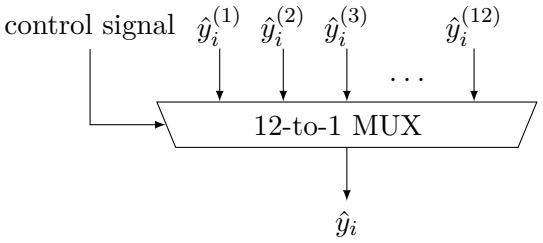


Figure 5.6: Input selection using the MUX-based pipelined detector for $N_T = 2$

with the exception of the first row, whose $\sqrt{M}$ columns correspond to the PEDs of the constellation points.

Figure 5.6 shows the arbitration of the independent signal vectors for a MIMO system with $N_T = 2$. In every cycle, a dedicated controller sends control signals to a bank of multiplexers, which chooses the signal vector that has access to the computation elements, where the signal vectors are chosen from Fig. 5.3. In the figure, the signal inputs at the ith level, $\hat{y}_i$, are connected to a 12-to-1 multiplexer, which chooses the appropriate RSV, to connect to the computation unit using a control signal. A similar multiplexing arrangement is also required for the channel matrix entries, $r_{i,j}$. Overall, 14 multiplexers are required for $N_T = 2$, while 44 multiplexers are required for $N_T = 4$.

A bank of symbol registers, with dimension $V \times K \times 2N_T$, is also required to store the temporary detected symbols, where V represents the number of concurrent RSVs. In a fully-pipelined detector, V is equal to the total number of clock cycles required to process one symbol vector (see Eqn. (4.2)). In this case, the total number of clock cycles required for the last level, C_1, is equal to 3 (instead of 4), since no sorting is carried out in the last level. Furthermore, the detector is able to switch immediately to the next RSV in the next cycle. Thus, the multi-stage detector achieves a slightly shorter latency (28 versus 30 clock cycles) than the single-stage detector presented in the previous chapter. At the output of the detector, multiplexers are also required to select the solution corresponding to the appropriate RSV from the symbol bank.

5.2.2.2 Shift-Register-Based Pipelined K-Best Detector

In the previous section, we presented a MUX-based pipelined K-best detector, which requires the PED and channel/signal registers to be replicated for each of the RSVs. However, that approach is quite inefficient and incurs a large area consumption. In this section, we will propose a more efficient method, by reusing the PED and channel/signal registers among different RSVs. This technique relies on the fact that an RSV does not require all its registers throughout the duration of the symbol detection. As such, unused registers may be assigned to other RSVs, as will be discussed subsequently.

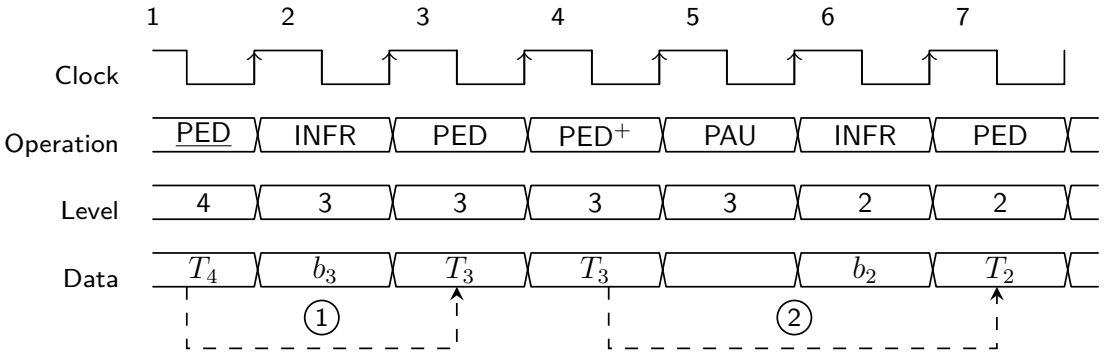


Figure 5.7: Duration of K-best variables during pipeline operation

PED Registers

Registers are required for storing the PEDs of the K-best results, which will be used as inputs to subsequent processing elements. In the MUX-based implementation, the PED registers will need to be reserved for each of the RSVs. However, a more efficient detector can be implemented by observing that the PED registers are not required for the entire duration of the pipeline, and thus can be reused for other RSVs.

The data movement of the PEDs in the pipeline is shown in Fig. 5.7 for a MIMO system employing $N_T = 2$. In the first cycle, the top level PED, T_4, is generated. In the second cycle, the interferences due to the top level symbols are computed according to (2.16). Finally, T_4 concludes its lifetime in the third cycle, where it is used in computing the second level PEDs as $T_3 = T_4 + |b_3 - r_{3,3}s_3|$. Thus, T_4 for a signal vector, $\hat{\mathbf{y}}^{(t)}$, can be saved into a register in the first clock cycle. In the second clock cycle, the register holds the computed T_4. At the same time, the detector computes T_4 for the next signal vector, $\hat{\mathbf{y}}^{(t+1)}$. At the third clock cycle, T_3 for $\hat{\mathbf{y}}^{(t)}$ is computed, while the content of the register is replaced by the computed T_4 for $\hat{\mathbf{y}}^{(t+1)}$. It should be noted that this data movement applies to all the available K paths.

The situation is different for T_3, which needs to be kept in memory for 2 clock cycles, in order to arrive just in time for the computation of T_2. Thus, a single shift register per path with a word size of 2, rather than 12 registers (or 28 registers in the case of $N_T = 4$) in the direct implementation is sufficient for all the RSVs. After T_3 is computed in the 3rd clock cycle, it is saved into the shift register. At each clock cycle, the value of T_3 is shifted to the right, while the computed T_3 for the subsequent RSV occupies its position. When the shift register is full, the T_3 values for all RSVs are correctly read during the computation of T_2. This analysis applies for all the PEDs at the remaining levels of the tree. Thus, with register sharing, the total number of PED registers required for storing the values of T_{i+1} is $(1 \times \sqrt{M}) + ((2N_T - 2) \times 2) \times K$. The number of PED registers required for storing T_{i+1} is plotted against N_T in Fig. 5.8 and compared with the direct implementation.

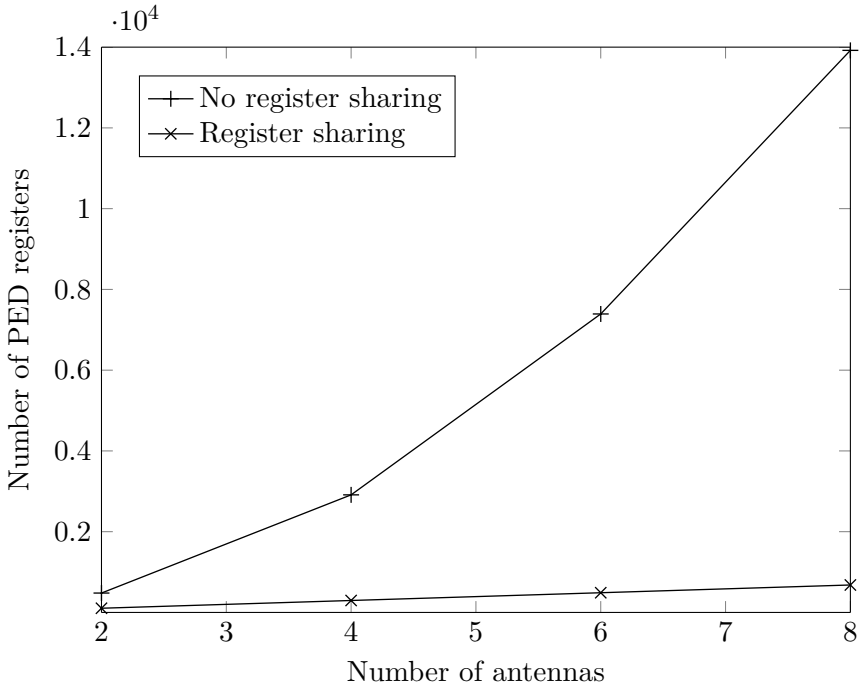


Figure 5.8: Number of T_{i+1} registers versus N_T

Channel and Signal Registers

In our proposed pipelined detector, we assume a new set of signal vectors and channel entries is sent to the detector at every clock cycle. When the pipeline is full, the detector must maintain memory for all the signal vectors and all the channel matrices. To ensure the correct functioning of the circuit, a control unit must decide which signal has access to the computation units at any given time. A straightforward (even if costly) approach is to save all the signal and channel entries for all the RSVs in memory and then use a multiplexer to select the appropriate signal. Similar to the PED registers, we note that the entries of the $\hat{\mathbf{y}}$ and $\mathbf{R}$ vectors are required for different lengths of time. For example, $\hat{y}_4$ and $r_{4,4}$ are only required in the first cycle. Meanwhile, $r_{1,1}$ must be held in memory for 11 clock cycles in order to compute the PEDs of the last level nodes. Figure 5.9 shows the signal and channel entries for the first RSV and the time and pipeline stages in which they are required. Instead of assigning dedicated registers for each channel and signal input, their values are saved into shift registers that are sized to ensure that the entries are correctly read at every clock cycle. Thus, a single register is sufficient for $\hat{y}_4$ and $r_{4,4}$. Meanwhile, a single shift register with a word size of 11 is used for storing the value of $r_{1,1}$ for all RSVs.

Symbol Registers

Symbol registers are used to store the intermediate detected symbols, $[\hat{s}_{2N_T}, \hat{s}_{2N_T-1}, \ldots, \hat{s}_1]^T$, for all the K paths. In a multiple-tree detection, each

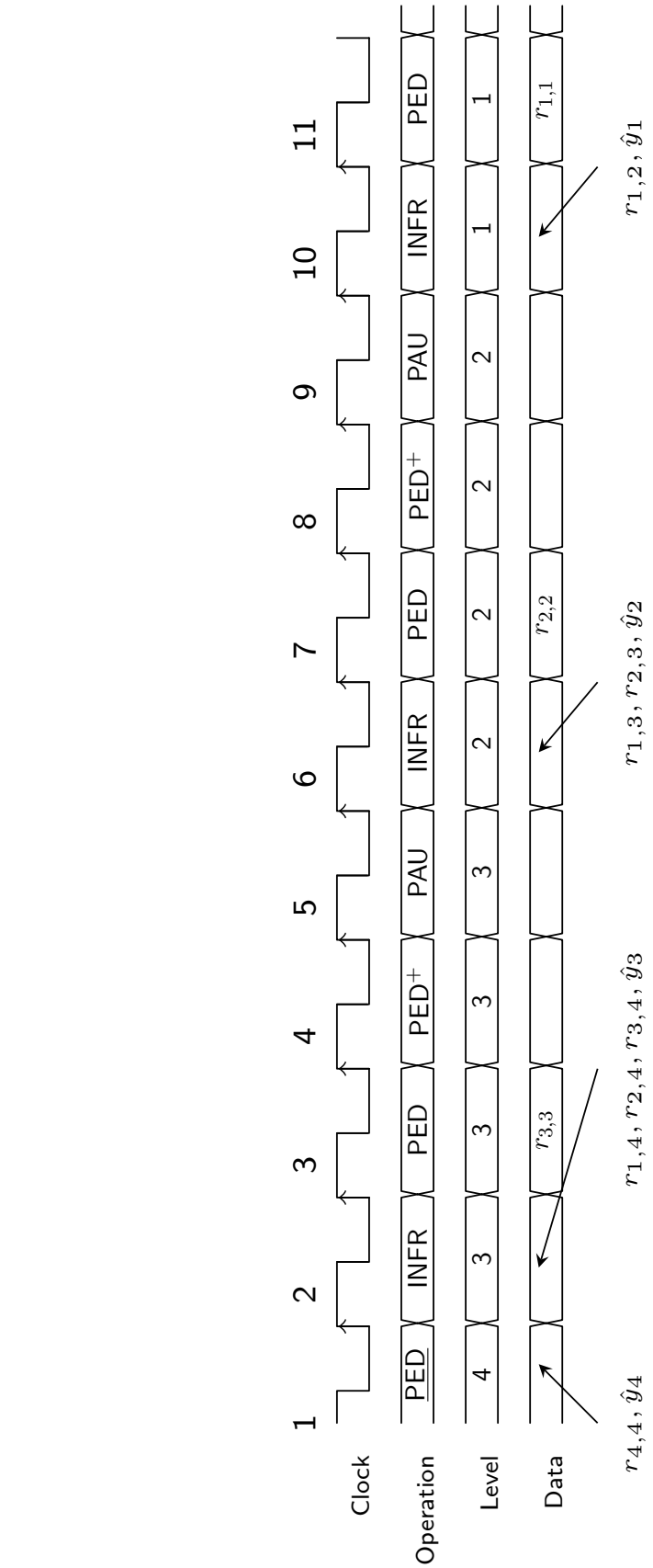


Figure 5.9: Data movement of channel and signal inputs in a 2×2 pipelined detector

Table 5.1: Symbol register sharing for $N_T = 2$. Four $\hat{s}_2$ symbol registers can be shared among all the RSVs, with three RSVs per symbol register, while a single $\hat{s}_1$ symbol register can be used for all the RSVs.

Register	**Number**	**RSVs**		
$\hat{s}_4$	N/A	Cannot be shared		
$\hat{s}_3$	N/A	Cannot be shared		
$\hat{s}_2$	1	$\hat{\mathbf{y}}^{(1)}$	$\hat{\mathbf{y}}^{(5)}$	$\hat{\mathbf{y}}^{(9)}$
✓	2	$\hat{\mathbf{y}}^{(2)}$	$\hat{\mathbf{y}}^{(6)}$	$\hat{\mathbf{y}}^{(10)}$
✓	3	$\hat{\mathbf{y}}^{(3)}$	$\hat{\mathbf{y}}^{(7)}$	$\hat{\mathbf{y}}^{(11)}$
✓	4	$\hat{\mathbf{y}}^{(4)}$	$\hat{\mathbf{y}}^{(8)}$	$\hat{\mathbf{y}}^{(12)}$
$\hat{s}_1$	1	$\hat{\mathbf{y}}^{(1)}$	...	$\hat{\mathbf{y}}^{(12)}$

RSV will need to be allocated a dedicated register bank for storing its symbols. Thus, a total of $K \times 2 \times N_T \times V$ symbol registers are required in the MTMS detector. However, unlike the PED and channel registers, all the symbols need to be in memory throughout the duration of the detection, since the final result is only determined after the path update operation at the last level. It is still possible however, to share symbol registers among different RSVs. For example, when an RSV is at the top level, then the lower level symbol registers may be used by another RSV that is currently processing those levels.

Table 5.1 shows a possible register sharing strategy for the symbol registers using $N_T = 2$. While the first two symbol registers cannot be shared (because they are required throughout the lifetime of a given RSV), four $\hat{s}_2$ symbol registers *per path* can be shared among all the RSVs. In Fig. 5.9, it can be observed that the $\hat{s}_2$ symbol register is only required in the 9th clock cycle during the path update stage of the first signal vector. Meanwhile, a single $\hat{s}_1$ register per path can be used for all the 12 RSVs, since it is utilised for a duration of one clock cycle only. Unfortunately, the multiplexing costs of such an arrangement, counter the area savings that can be achieved from sharing the symbol registers. However, the symbol register for the last level, $\hat{s}_1$, requires no multiplexing as it is utilised for only one clock cycle. Thus, in the proposed implementation, only the last level symbol registers are shared.

5.3 Hardware Implementation

In this section, we will present the VLSI implementation aspects of the MTMS K-best detector in more detail. We will adopt the shift-register-based architecture described in the previous section for the proposed implementation. The overall architecture is presented in Fig. 5.10. The controller and datapath of the proposed detector are discussed in the subsequent sections.

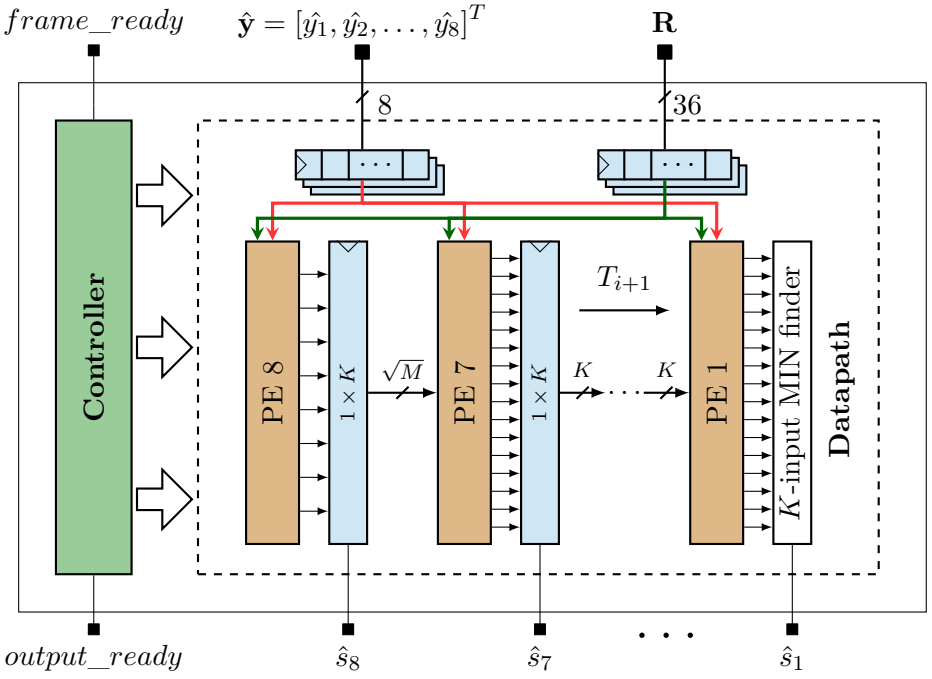


Figure 5.10: Overall architecture of the multiple-tree multi-stage architecture. The "MIN" finder in the last stage is constructed in a similar fashion to Fig. 4.4, in order to reduce the critical path.

5.3.1 Controller

Like the single-stage detector of the previous chapter, the architecture of the MTMS detector is separated into a single controller unit and a datapath engine. The implementation of the controller unit is relatively simple. Control signals specify which signal vector is carrying out a given operation at any point in time. Although two signal vectors can carry out the same operation concurrently, no two signal vectors carry out the same operation for the same tree level at the same time.[2] Once a given operation for the first signal vector has been determined, the controller repeats the same operation for all the remaining signal vectors, $\hat{\mathbf{y}}^{(2)}, \hat{\mathbf{y}}^{(3)}, \ldots, \hat{\mathbf{y}}^{(V)}$, in that order. After the last signal is reached, the controller rolls back and repeats the process starting from the first signal vector. A simple counter can be used to automatically determine the *next signal vector* for each operation at any time. However, a more efficient design can be achieved by specifying the next signal vectors explicitly in every state of the controller.

Figure 5.11 shows the ASM chart for the MTMS controller. For simplicity, the ASM chart is shown for 2 transmit antennas. The controller has 13 states, with 1 state for *idle* and the remaining 12 states corresponding to the start of a new signal. Unlike the single stage controller in Fig. 4.11, no dedicated *done* state is required. Instead, when the pipeline is full, the output is read in every clock cycle. The 12 states $\mathtt{S}_1$ to $\mathtt{S}_{12}$, correspond to clock cycles 1 to 12, during the *pipeline loading* stage. For the first RSV,

[2]For example, only one signal may be computing the first level PEDs at any given time.

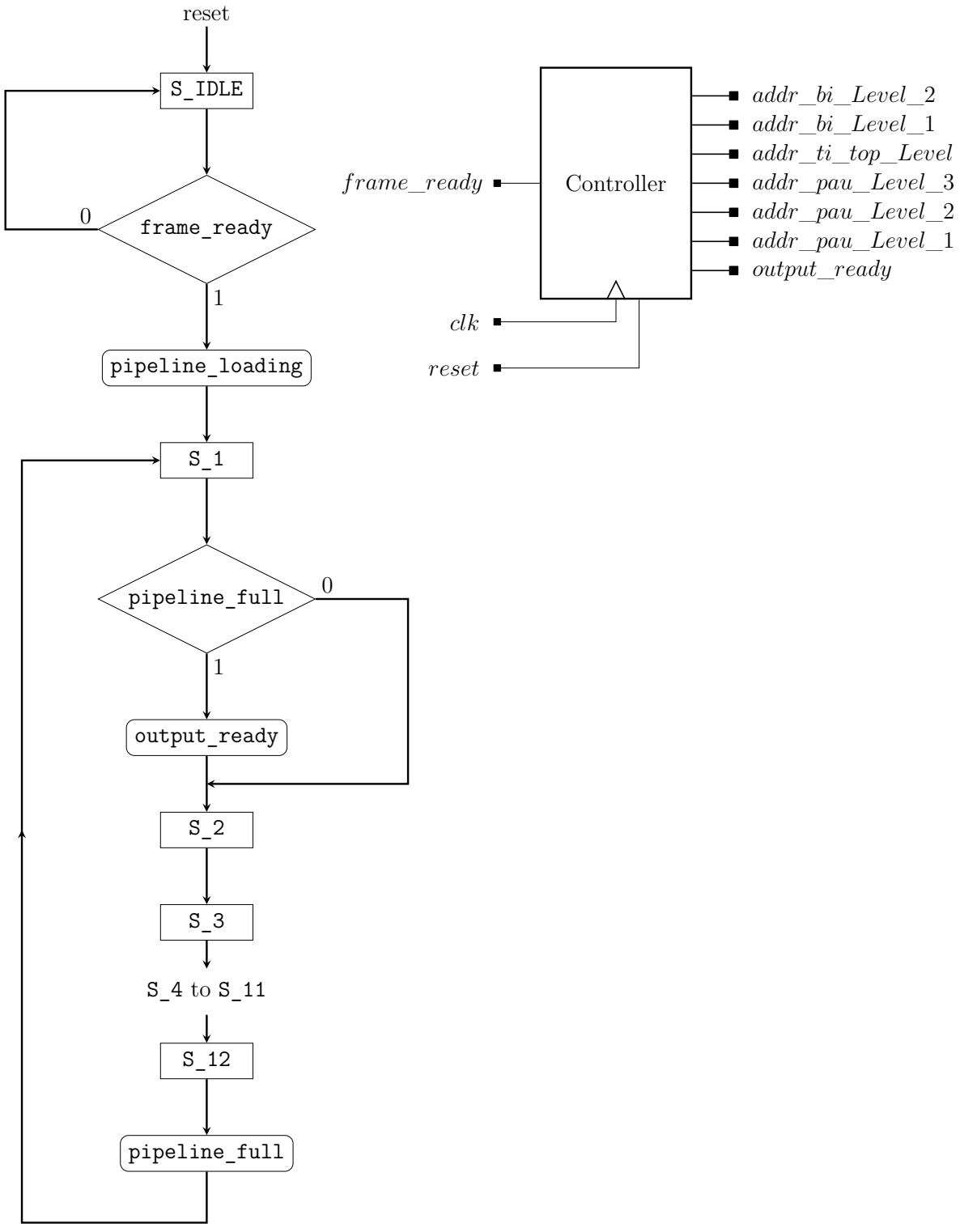


Figure 5.11: ASM chart for the K-best MTMS controller for a 2×2 MIMO system. The structural representation of the controller is also shown with the relevant control signals.

states $\mathtt{S_1}$ to $\mathtt{S_{12}}$ correspond to the full detection of one symbol vector. After the pipeline is full, the controller FSM restarts from $\mathtt{S_1}$, and the `pipeline_full` flag is asserted. The corresponding states of the other RSVs, $\hat{\mathbf{y}}^{(1<j\leq 12)}$, are cyclically offset from that of the tth state of the first RSV as follows:

$$\mathtt{S}_t^{(1)} \Longleftrightarrow \begin{cases} \mathtt{S}_{j+t}^{(j)} & j+t \leq V \\ \mathtt{S}_{V-j+t}^{(j)} & j+t > V, \end{cases}$$

where the superscript refers to the signal index in question. For example, state $\mathtt{S_2}$ for the first RSV corresponds to state $\mathtt{S_1}$ for $\hat{\mathbf{y}}^{(12)}$. Both states compute b_3 for $\hat{\mathbf{y}}^{(1)}$ and $\hat{\mathbf{y}}^{(12)}$ respectively.

The state and next state registers are coded using one-hot encoding to ensure a fast state transition. Also shown in Fig. 5.11, is the block diagram of the controller with the main input and output signals. Unlike the single-stage controller, no *Level* signal is explicitly sent to the datapath. The address control signals, denoted by "addr", indicate which signal vector is carrying out a given operation and at which level. All the address signals are 4-bit wide and thus can specify up to 16 signal vectors, although only 12 signal vectors are required for $N_T = 2$. The functions of the ports of the controller unit are described subsequently. i denotes the corresponding levels in which the control signal is issued.

- `addr_bi_Level_i`: These control signals indicate which RSV is currently computing b_i at the 1st and 2nd levels respectively. They are used to route the appropriate previously detected symbols, $[\hat{s}_{2N_T}, \ldots, \hat{s}_{i+1}]^T$, to the interference computation units.

- `addr_ti_top_Level`: This control signal specifies which RSV is currently computing the PEDs for the top level nodes (i.e. the constellation points). At state `S_1`, `addr_ti_top_Level` = 1, which indicates that the datapath will compute the PED of the first RSV in the next clock cycle. The control signal is incremented by 1 until `S_12`. The main function of this control signal is to route the appropriate symbol register to the output ports of the K-best detector. In the case of the MUX-based pipelined detector, additional control signals will need to be implemented to select the signal that is currently computing the PEDs for the remaining levels in the tree, i.e., $i = 2N_T - 1$ to $i = 1$.

- `addr_pau_Level_i`: These control signals are used in the path update stage to access the previously detected symbols for the current signal for the 1st, 2nd and 3rd levels respectively.

- `frame_ready`: This is an input signal that is used to launch the detector from the idle state, `S_IDLE`, to the first state, `S_1`. After this input signal is asserted, the

pipeline starts getting filled up, which is indicated by the `pipeline_loading` flag. At this point, the registers in the datapath are enabled.

- `output_ready`: This signal is asserted in the `S_1` state after the pipeline is full. Once this control signal is asserted, the detected MIMO symbols can be read from the detector in every cycle.

It should also be mentioned that for the MUX-based detector, the next signal, $\hat{y}_i$, as well as the channel entries, $r_{i,j}$, need additional "addr" buses to explicitly specify the current signal as shown in Fig. 5.6. However, this is not required in the shift-register-based detector as the next signals are determined automatically at every cycle with little intervention from the controller. However, both architectures do require the signals to be explicitly specified for accessing detected symbols from the symbol registers, for example, in order to compute b_i. This is achieved using the `addr_bi_Level_i` and `addr_pau_Level_i` buses.

5.3.2 Processing Element

The architecture of the processing element for the multi-stage architecture is basically the same as that of the single-stage architecture as shown in Fig. 4.12. However, in this case, there are no feedback paths from the merge network to the input of the PE; instead, the outputs of the merge network are fed to the next PE. Furthermore, the controller does not need to supply the current level to the processing element. The top level PE simply expands the constellation points and passes the PEDs to the second level PE. Since the number of constellation points $\sqrt{M}$ is less than K, no sorting is required at the top level PE. Furthermore, no SE enumeration is required. The second level PE expands the children of the constellation points and sorts them according to their PEDs. The total number of candidates expanded is $\sqrt{M} \times \lambda$, where λ is the number of children expansible per parent. An SE enumeration is also required for presenting the candidates to the merge network in a pre-sorted format. However, it should be noted that SE enumeration is not necessary for the K-best detection if a brute-force sorter (e.g. bubble-sort), which compares all the candidates, is employed.

The PEs at the remaining levels, $i = 6$ to 1, are quite similar: each PE comprises K expansion units operating in parallel, and one merge network. The expansion unit comprises an ICU and λ PED units. The main difference between the PEs is in the complexity of the ICUs, with PEs at lower levels requiring longer adder-chains in the ICUs than those at upper levels. Since each level is uniquely mapped to a PE, in the case of the multi-stage architecture, it is possible to reduce the complexity of a PE depending on the level it corresponds to. For example, in hard-output detection, the merge network can be eliminated in the last PE and replaced with a simpler MIN finder, which finds the minimum-metric path. Similar to the single-stage detector, a path update is required

at the end of every level, to ensure that the paths are arranged according to the sorted PEDs at the current level. However, unlike the single-stage detector, the path update operation in the MTMS detector is carried out in every clock cycle instead of at the end of every level, which contributes to an increase in the power consumption.

5.4 Results and Discussion

In this section, the implementation results of the fully-pipelined K-best detector are presented. The detector consumes a post-layout area of approximately 2.74 mm^2, which is equivalent to 1753 kGE in the ST 65 nm CMOS technology. The merge networks in PEs 6 to 2 contribute more than 30% of the total area consumption. As a result of the proposed pipeline scheduling, the area savings of the signal and channel memories compared with the MUX-based detector is 59%, while the reduction in the PED registers is 92%. The detector achieves a throughput of 3288 Gbps in the 64-QAM, 4×4 configuration.

Given 14 bits for the signal and channel matrix entries, the implemented detector has a total of 654 input and output pins. Since the number of entries in the upper triangular channel matrix has a quadratic relationship with the number of antennas,[3] the number of input/output pins might not be realisable on certain packages for larger MIMO configurations. While time-multiplexing might be used to read in the channel entries in the case of the SD and single-stage K-best detector (for example, by reading the upper triangular channel matrix in a column-wise manner), this is difficult to achieve in the proposed fully-pipelined implementation, since the pipeline begins reading all the entries of the channel matrix and received signal vector at the same time. This is thus a notable disadvantage of the fully-pipelined approach, which merits further investigation.

In the following sections, the significance of the implementation results will be evaluated based and compared with state-of-the-art VLSI implementations. The potential application of the implemented detector to recent high-throughput wireless communications standards will also be discussed.

5.4.1 Cost of Pipelining

The cost of pipelining is the overhead incurred by a pipelined implementation to achieve a given throughput gain over the corresponding unpipelined implementation. Obviously, to achieve an efficient pipelined implementation, the throughput gain with respect to the unpipelined implementation, must be greater than the area penalty. In order to assess

[3] There are $N_T + 2N_T^2$ channel entries overall when using the RVD. For a 4×4 channel matrix, 36 channel input buses are required, while as much as 136 channel input buses will be required in the case of an 8×8 channel matrix.

Table 5.2: Comparison of implementation results for MTMS, STMS and STSS K-best architectures

Architecture	**MTMS**	**STSS**		**STMS**	
Number of cores	1	1	30	1	30
Area [kGE]	1753	285	8550	1089	32,670
Φ [Mbps]	3288	109	3270	113	3390
TAR [Mbps/ kGE]	1.88	0.38	0.38	0.10	0.10
Power [mW]	580	34.20	1026	42	1260
E_{bit} [pJ/bit]	176	312	312	372	372

the efficiency of the proposed pipelined detector, it is compared with an unpipelined multi-stage architecture, otherwise known as the single-tree multi-stage (STMS) K-best detector, as shown in Table 5.2. The throughputs of all detectors are computed at a clock frequency of 137 MHz.

The STMS requires 29 clock cycles to fully detect one symbol vector, resulting in a throughput of 113 Mbps. Notice that the STMS requires one less clock cycle than the serial STSS detector presented in the previous chapter. This is because the last stage of the STMS requires a single clock cycle for finding the minimum-metric path, whereas, two clock cycles are required for getting the best results in all stages of the STSS. It should be noted that the STMS is simply the MTMS without the multiple tree search processing capability. Thus, to evaluate the pipelining cost, we recall from Section 2.8.1 that the hardware efficiency of a detector is defined as the ratio of the throughput to the area consumption (TAR). To compare the hard efficiencies of the pipelined and unpipelined detectors, the ratio of their respective TARs can be computed as follows:

$$\begin{aligned}\text{RTAR} &= \frac{\text{Hardware efficiency of pipelined detector}}{\text{Hardware efficiency of unpipelined detector}} \\ &= \left(\frac{\text{Pipelined throughput}}{\text{Pipelined area}}\right) \times \left(\frac{\text{Unpipelined area}}{\text{Unpipelined throughput}}\right), \end{aligned} \tag{5.1}$$

to derive the relative throughput to area ratio (RTAR). Since the TAR gives the throughput deliverable per unit area for a given implementation, the RTAR evaluates the throughput gain of the pipelined detector over the unpipelined detector using the *same area* consumption. The RTAR of the MTMS detector with respect to the STMS is thus $(3288/1753) \times (113/1089) = 19.5$. Therefore, with the same area, the fully-pipelined MTMS detector provides approximately 20 times the throughput deliverable by the unpipelined STMS implementation. In other words, the throughput is improved by a factor of 20$\times$ relative to the penalty incurred in the area consumption. The MTMS incurs an area increase of 61% compared with the STMS detector, while improving the throughput by a factor of 29$\times$. As expected, the MTMS detector incurs a higher power consumption compared to the unpipelined STMS detector. However, due to the vast throughput

improvement, the energy efficiency is actually significantly improved in the pipelined MTMS implementation.

5.4.2 Pipelining versus Interleaving

In the previous chapter, the single-stage K-best detector was presented, and detector interleaving was proposed to improve its throughput. Table 5.2 also compares the result of the STSS detector and the proposed MTMS detector. The results indicate that approximately 30 cores, with a combined area of 8550 kGE, are required to achieve the throughput of a single MTMS detector. Furthermore, the interleaved STSS detectors incur almost 2× the power consumption of the MTMS detector. However, the STSS achieves a better RTAR than the STMS detector discussed in the previous section, which suggests that the multi-stage architecture is only beneficial if multiple-tree processing is employed. At the same area consumption, the MTMS delivers approximately 5× the throughput of the STSS detector. The MTMS also achieves a lower energy-per-bit than the STSS detector. The foregoing results suggest that pipelining is a more effective technique than interleaving for meeting the high-throughput requirements of next-generation wireless systems. Despite this, the STSS detector is still relevant in applications requiring low data rates and low power consumption, such as wireless sensor networks.

5.4.3 Comparison with State-of-the-Art

The proposed detector is compared with state-of-the-art K-best detectors for a 64-QAM, 4×4 MIMO system in Table 5.3. TCSYS'14 [77] and JSPS'16 [102] feature a similar fully-pipelined architecture to the proposed implementation and report higher throughput figures. However, in the case of TCSYS'14 [77], a low complexity is obtained partly by employing variable K values, which degrades the performance to an extent and also limits its applicability to soft-output detection.

JSPS'16 [102] reports the highest throughput of 26 Gbps in the open literature. However, it requires costly division operations in the preprocessing stage, which need to be executed for each received signal. This will likely degrade the stated throughput and power consumption in a practical scenario. Thus, a fair comparison will include the preprocessing cost as well. By contrast, the proposed MTMS implementation requires no multiplication or division at the symbol-rate processing, due to the use of simple shift/addition-based multipliers and tabular enumeration. It should also be noted that unlike TCSYS'14 [77], the stated area of our implementation includes the channel matrix registers as well. For the 4×4 MIMO configuration considered, the architecture can support up to 28 independent channel matrices, which makes it suitable for both slow and fast-fading channel scenarios.

Table 5.3: Comparison with 64-QAM 4×4 K-best Detectors

Reference	**TVLSI'10 [96]**†	**ISCAS'12 [101]**	**TVLSI'13 [97]**†	**TCSYS'14 [77]**†	**ISCIT'15 [15]**†	**JSPS'16 [102]**†	**Proposed**
Architecture	STSS/Hard	MTMS/Soft	MTMS/Hard	MTMS/Hard	STSS/Hard	MTMS/Hard	MTMS/Hard
Model	Real	Real	Complex	Real	Real	Complex	Real
Pipelining	N/A	Partial	Partial	Full	N/A	Full	Full
Tech. [nm]	65	130	130	90	65	130	65
V_{dd} [V]	1	N/A	1.2	1.3	1.05	1.3	1.05
Area (kGE)	1760	223	340	232	157.5	950	1753
K	64	8	10	≤ 8	10	10	16
$\Phi^{'}$ [Mbps]	100	1710	2000	5649	300	26600	3288
Symbols / cycle	0.03	0.13	0.10	1	0.04	1	1
f_{clk} [MHz]	158	285	417	170	287	556	137
$P^{'}$ [mW]	237.6	258	850	144	23	510	580
$E^{'}_{\text{bit}}$ [pJ/bit]	2376	301	850	35.3	100	38.3	176

† Power, throughput and energy-per-bit are scaled to the 65 nm 1.05 V technology according to Equations (2.22) and (2.24).

Table 5.4: Optional and Mandatory Parameters for IEEE 802.11ac PHY Layer

Parameters	**Mandatory**	**Optional**
Channel Bandwidth (MHz)	20, 40, 80	160, 80 + 80
Forward Error Correction	Convolutional	LDPC
Number of streams	1	2 to 8
Modulation	BPSK, 4/16/64-QAM	256-QAM

5.4.4 Application to Current Wireless Standards

As has been highlighted in the introduction to this thesis, the growing user demand for high data-rate communications, has led to the development of sophisticated wireless standards, capable of achieving several gigabits per second in throughput. The use of MIMO technology in conjunction with multiuser support, allows a communication system to deliver vastly improved data rates compared to legacy SISO communication schemes. One of such recent wireless standards is the IEEE 802.11ac, which was developed in 2013, for wireless LAN communications [103]. In the following sections, we will discuss the potential application of the proposed fully-pipelined K-best detector to an IEEE 802.11ac wireless system.

5.4.4.1 Throughput Considerations

The IEEE 802.11ac standard builds upon IEEE 802.11n by providing support for up to 256-QAM and increasing the channel bandwidth from 40 MHz up to 160 MHz. IEEE 802.11ac also provides multiuser support as well as support for up to 8 downlink antennas. That is, multiple users, each with a different MIMO configuration (e.g. 2×2), can be served by a single access point with 8 antennas. The mandatory and optional parameters for IEEE 802.11ac are summarised in Table 5.4. In the 160 MHz channel bandwidth configuration, IEEE 802.11ac supports a maximum throughput of 650 Mbps per stream at a coding rate of 5/6 and 64-QAM modulation scheme [104]. This corresponds to a hard detection throughput of 3120 Mbps for a 4×4 MIMO system,[4] which suggests that the proposed detector is potentially applicable to an IEEE 802.11ac communication system. Assuming the same clock frequency, the detector can achieve over 4 Gbps, which makes it potentially applicable to the 256-QAM configuration, which offers the maximum data rate possible. An obvious bottleneck when forward error correction code is employed, is the need for a channel decoding step at the receiver end, which needs to match the throughput of the detector. Furthermore, a deinterleaver will be required to unscramble the decoded bits at the receiver end, if the information bits are interleaved prior to transmission. However, the effect of these additional processes on the overall throughput of the MIMO receiver unit has not be studied in this thesis.

[4]Hard detection throughput for 4 streams is given as: (throughput per stream $\times$ number of streams)/(coding rate) = $(650 \times 4)/(5/6) = 3120$ Mbps.

5.4.4.2 Performance Considerations

In a typical wireless communication system, channel coding is employed to make the data transmission more robust to errors. IEEE 802.11ac provides an optional low-density parity-check (LDPC) code in addition to the mandatory binary convolutional coding. The actual coding rate employed is a tradeoff between the performance and throughput requirement. Although only the VLSI implementation of the hard-output MIMO detector was presented in this thesis, the proposed detector can easily be extended to provide reliability information for each detected bit, using approximations to the max-log computation as provided in [59] (see Section 2.7.5).

In order to compute the reliability information, a candidate list needs to be built, which comprises several potential candidate solutions extended down to the final level. In K-best detection, this is straightforward, since several competing paths are generated in order to find the minimum metric candidate (i.e. the hard detection output). Unfortunately, the choice of $K = 16$ as adopted for the proposed implementation does not provide an adequate candidate list size to compute the LLRs accurately. An obvious approach will be to increase K to a large value, for example, $K = 256$, however, this is hardly practical, due to the large area overhead required by the sorters, which would also degrade the achievable maximum clock frequency. Another approach will be to simply extend a selected number of paths from a given level without any sorting, in a manner similar to the fixed-complexity sphere decoder. For example, if all paths from level 3 are extended, a maximum list size of 1024 can be obtained. That is, $\sqrt{M} \times \sqrt{M}$ children are extended in levels 1 and 2 from K best paths in level 3, resulting in a total of $K \times \sqrt{M} \times \sqrt{M}$ candidates. As with the case of the hard-output detector, the number of children per parent per level can be reduced, in order to control the size of the candidate list and achieve complexity-performance tradeoffs.

Figure 5.12 compares the hard and soft-output K-best detectors for 64-QAM 4×4 using a convolutional code, with a code rate of 1/2, for the forward error correction and a payload size of 1200 bytes. At this code rate, the achievable throughput of the soft-output detector is approximately $1/2\times$ the throughput deliverable by the hard-output detector. The figure shows that a large list size is required for maximising the BER performance compared to the hard-output detector. This observation also precludes simply increasing K to build the candidate list for computing the LLRs. At a list size of 3888, the SNR gain compared to the hard-output detector at a BER target of 10^{-4} is approximately 4 dB, while a list size of 1296 achieves approximately a 2 dB gain. A list size of 256 does not provide any appreciable performance gain beyond $\mathrm{E_b/N_0} = 25$ dB. This suggests that a substantial list size is required to take advantage of the convolutional code employed in an IEEE 802.11ac system. It is possible to achieve a smaller list size by performing two or more *detection-decoding* iterations at the expense of a reduced throughput. Although a large list size is desirable for improved performance, as shown in the plot, it also makes

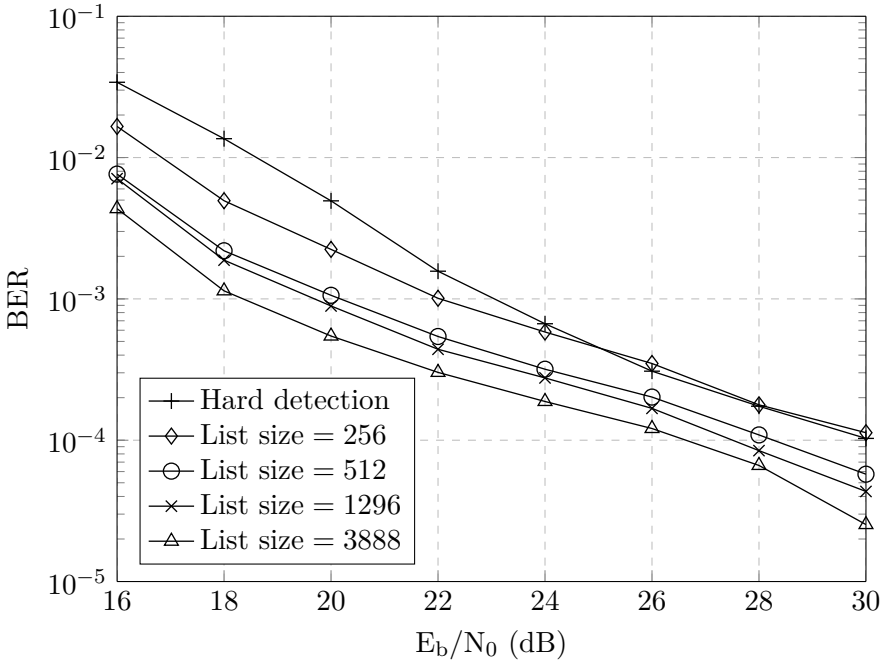


Figure 5.12: BER vs SNR comparison of hard and soft K-best detectors for 64-QAM 4×4 with code rate $= 1/2$, $K = 16$ and constraint length $k = 7$

computing the probability information for all the bits quite cumbersome. The integration of a low-complexity low-latency LLR unit with a fully-pipelined K-best detector is an interesting direction for future research.

5.5 Summary and Conclusion

In this chapter, the VLSI implementation of a fully-pipelined K-best detector has been presented. The detector processes several MIMO tree searches concurrently, which results in a processing rate of one MIMO symbol vector per clock cycle. Although it has been suggested by some authors that the sphere decoder achieves a better data rate than the K-best detector [64], [68], the results of this chapter indicate that the K-best detector is more suited to high-throughput applications compared to the sphere decoder. The proposed detector has also been compared with the single-stage K-best detector presented in Chapter 4, and the results indicate that a single fully-pipelined detector is more hardware and energy efficient than interleaving several unpipelined detectors. However, single-stage detectors still remain useful for applications requiring low power and low data-rates. The application of the proposed detector to the IEEE 802.11ac standard was also discussed, and a number of possible design challenges were identified.

The foregoing results suggest that the single-core implementation of the SD, as well as the single-stage K-best detector, are best suited to a complex channel model, while the fully-pipelined detector, is more suited to the real channel model. At a clock frequency of

137 MHz, the proposed fully-pipelined detector achieves a throughput of over 3 Gbps and a power consumption of 580 mW, which makes it well-suited to applications requiring gigabit data rates.

Chapter 6

Conclusions and Future Work

6.1 Summary and Conclusions

MIMO technology delivers several benefits to communication systems as highlighted in Chapter 1. However, MIMO also complicates the design of the receiver as a result of the multiple interferences at each receive antenna. In the worst case, the signal detection at the MIMO receiver scales exponentially with the number of transmit antennas, which is problematic in a real-time system. In this thesis, we have tackled two main objectives namely: achieving high-throughput detection and secondly, achieving low-power consumption. These two objectives are crucial to forthcoming 5G communication systems, and ASICs provide an attractive platform for realising these goals. We also posed a number of research questions, which motivated us to undertake the implementations of different MIMO algorithms and architectures, with the view of evaluating their impacts on the aforementioned objectives. As a result, a number of novel contributions have been achieved as follows:

1. In Chapter 3, we presented a modified SD algorithm, which achieves 25% reduction in the number of clock cycles compared to the conventional SD algorithm, based on a real channel model, through the use of a "look-ahead" strategy. We also presented the implementation of a modified PED computation unit for the SD based on the work of Kang and Park [90], which resulted in a lower area consumption and a higher clock frequency compared with the conventional implementation. Two VLSI implementations of the SD based on the aforementioned techniques were also presented.

2. Also in Chapter 3, we proposed the use of adaptive runtime constraints for the SD, which uses less stringent runtime constraints at SNR values below a cut-off SNR determined through simulations. This results in almost a 90% reduction in the number of clock cycles compared to the unconstrained SD.

3. In Chapter 4, we implemented a reduced-complexity hybrid-merge network, which combines the Batcher's odd-even and bitonic sorters to achieve a highly optimised design. The hybrid-merge algorithm achieves an area reduction of 30% compared with the merge network utilising odd-even sorters only. Furthermore, a reduced complexity K-best algorithm was proposed, which incurs minimal performance loss compared with the conventional K-best algorithm.

4. In Chapter 5, we implemented a fully-pipelined K-best detector, which achieves a throughput of 3.29 Gbps in the 64-QAM 4×4 MIMO configuration, which is one of the highest reported in the literature. Furthermore, the proposed implementation achieves one of the best energy-efficiency figures compared with other K-best detectors from the literature. The proposed detector features a novel pipeline scheduling, which results in area reductions of 59% and 92% to the signal/channel and PED registers respectively.

5. The interleaved and pipelined implementations of the K-best detector were also compared in Chapter 5. Based on the implementation results, we can conclude that although pipelining has area and power consumption overheads compared with unpipelined implementations, it is more hardware and energy efficient than corresponding interleaved implementations of unpipelined detectors. The fully-pipelined detector presented incurs an area overhead of 61% compared to the unpipelined detector, while significantly improving the throughput by a factor of 20×.

6.2 Design Guidelines

The choice of a MIMO detector is critical in the overall baseband processing at the receiver. In a coded transmission, the MIMO detector has an impact on the design of the channel decoder as well. In this thesis, we have studied two of the most popular MIMO detectors namely, the sphere decoder and the K-best detector. In Chapter 1, a number of design objectives were identified. Because these objectives are conflicting, it is unlikely that any one MIMO detection algorithm will be "best" for all communication requirements. Apart from the choice of detection algorithm, the architecture of the detector itself, as shown in this thesis, has a significant impact on the overall receiver performance. With the VLSI results of the aforementioned detectors, and discussions in the background chapter, a number of design guidelines can be provided as follows:

- **Area:** With the rapid technology scaling, this objective is mainly critical when targeting off-the-shelf platforms such as FPGAs and DSPs, which have a fixed number of resources, which cannot be optimised further. Typically, the area is sacrificed to meet other design objectives. The sphere decoder offers the smallest

area consumption among the tree-search algorithms. The K-best detector can be implemented using a similar folded architecture to the SD and would be an alternative if a fixed throughput is required in addition to a small implementation area. Strategies used for reducing the area consumption employed in the proposed detectors, include the use of single-stage architectures; elimination of division in determining the SE enumeration through the use of tabular enumeration; the use of the optimised partial Euclidean distance computation for the SD and the adoption of a low-complexity ℓ^1-norm approximation to the partial Euclidean distance.

- **Performance:** This is the most critical requirement of the identified objectives. However, due to practical reasons, the "optimal" BER performance cannot be achieved. Furthermore, different applications require different qualities of service (for example, audio can tolerate a less reliable transmission than textual data). In view of this, the performance of the MIMO detector can be traded off to achieve other objectives such low power and high throughput. The unconstrained sphere decoder (and best-first search algorithm) offers the best performance in an uncoded scenario. However, in practice, channel coding will also be required in addition to the hard-detection covered in this thesis. Thus, the natural extensibility of the K-best algorithm to soft-output generation is a factor worth considering.

- **Throughput:** The sphere decoder features a single-tree processing, which limits the throughput to the number of clock cycles required to detect one symbol vector. Furthermore, the sphere decoder suffers from a significant throughput degradation in poor channel conditions. Although runtime constraints can be used to improve the throughput of the sphere decoder, the K-best detector implemented with a comparable folded architecture appears to have an advantage. More significantly, however, the K-best detector can be implemented in a multi-stage architecture, which can support a partial or fully-pipelined processing, which is attractive for gigabit MIMO detection. The use of merge networks, rather than multi-cycle sort algorithms, has been shown to be advantageous for high-throughput implementations of single-tree architectures. Multi-cycle sort algorithms may be used in a fully pipelined implementation without an impact on the throughput, however, the latency is increased as a result. A complex channel model appears to be more suitable for single-tree detectors, while a real channel model is attractive for the fully-pipelined detector, since the reduced tree-depth of the complex channel model does not offer any throughput advantage in a fully-pipelined scenario, and the real channel model allows simpler computational units to be achieved. A comparison of the fully-pipelined detector using both the real and complex channel models is an interesting direction for future research.

- **Power:** Power consumption is critical in communication systems, particularly at the receiver-end. In a power-centric application, the sphere decoder appears to be the most suitable tree-search detection scheme. However, runtime constraints need

to be applied to the sphere decoder to prevent a near-ML tree search in low SNR conditions, which can increase the power consumption. In conclusion, the sphere decoder appears most suited for applications requiring low power and small area (e.g. medical devices and remote wireless sensor nodes), while the K-best detector is more suited for applications requiring a high throughput performance, such as wireless LANs.

6.3 Future Work

A number of techniques at both the circuit and algorithmic levels have been investigated in this thesis, for the implementation of low-complexity and high-throughput MIMO detectors. Considering the rapid-changing developments in the wireless communications field, it is obvious that further innovations will be required to meet future challenges. Some possible areas for further investigation are presented as follows:

- **Adaptive Modulation:** In a typical wireless system, the transmitter would employ different modulation schemes depending on the channel condition. For example, in the IEEE 802.11ac standard, BPSK is used for sending channel state information, which requires a high level of reliability but low data rate, while 64-QAM or 256-QAM are employed at high SNR to achieve a higher transmission rate. In this thesis, we have only considered the case of a single modulation scheme, which will not be suitable for all channel conditions. The presented MIMO detectors can be modified to accommodate different modulation schemes by instantiating different enumeration lookup tables for each modulation scheme and making the channel condition available to the MIMO detector.

- **Throughput:** One of the main objectives of this thesis is realising high-throughput MIMO detectors. To this end, several innovations, such as fine-grained pipelining and pre-computation using lookup tables have been employed in order to speed up the detector. However, there is still scope for further improvements in this aspect. For example, the merge network adopted in our proposed detectors constitutes a bottleneck in the attainable speed of the detectors due to its large combinational delay. The throughput of the detector can be improved by further pipelining the merge network; however, this would have an undesirable impact on the area as well as the dynamic power consumption. On the other hand, low-complexity sorting algorithms that have been so far investigated in the literature inevitably lead to a deterioration of the detection performance. We can therefore conclude that high-performance and low-complexity sorting algorithms will be vital in meeting the high-throughput requirements of future communication systems.

- **Preprocessing:** Preprocessing plays a crucial role in reducing the complexity of the MIMO detector as has been investigated in several works [38], [105], [106]. Unfortunately, however, preprocessing is itself quite computationally expensive, and this may be even more significant in frequently-changing channel conditions. While tree search algorithms require simple operations, preprocessing typically involves complicated operations such as division and matrix inversion. Furthermore, the preprocessing must be carried out with a high level of accuracy (since any errors will be propagated to the MIMO detector), and this limits the application of simpler but sub-optimal operations in the preprocessor unit. Considering the impact of preprocessing on the detection as a whole, novel low-complexity preprocessing techniques are required for achieving lower complexity MIMO detectors.

- **Soft-Output MIMO Detection:** In this thesis, we have considered the case of hard-output detection only. However, a typical wireless system would employ one or more forward-error codes in order to improve the reliability of the transmission. This will require a channel decoder at the receiver in addition to the MIMO detector. Fortunately, the K-best detectors presented in this thesis are very suitable for soft-output generation, since several solutions are generated as by-products of the hard-detection. The presence of multiple antennas, apart from complicating the MIMO detection, also makes the channel decoding more difficult. Thus, complete ASIC systems, combining MIMO detection as well as channel decoding, merit more attention from the future works.

- **Technology Scaling:** One of the major challenge encountered during this thesis was the implementation of a high-throughput sorting unit for the K-best detector. One possible approach for speeding up the sorting operation is by using more advanced transistor technology. The disadvantage of this, however, is that the circuit becomes increasingly susceptible to single-event upset (SEU) induced errors, which will need to be mitigated. Several SEU mitigation strategies at both the circuit and system levels have been widely investigated in the literature [107], [108]. An interesting research will be to model the degradation of the performance of the MIMO detector with transistor scaling and apply a combination of these mitigation techniques to improve the performance.

Appendix A

QR Decomposition

QR decomposition is a frequently applied technique for solving mathematical problems involving matrices. Consider an $N \times M$ matrix, $\mathbf{A}$. QR decomposition is applied to express $\mathbf{A}$ as a product of an $N \times N$ matrix, $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N]$, with orthogonal columns (i.e. $\mathbf{q}_i \mathbf{q}_j^T = 0, i \neq j$), and an $N \times M$ upper triangular matrix, $\mathbf{R}$. Due to the unitary nature of $\mathbf{Q}$, we have $\|\mathbf{QR}\| = \|\mathbf{R}\| = \|\mathbf{H}\|$, which allows $\mathbf{R}$ to be used in place of $\mathbf{H}$ in many situations. In the next sections, we will discuss some algorithms for QR decomposition.

A.1 Givens Rotation

Consider two rows in the matrix $\mathbf{A}$ as follows $\mathbf{a} = [0, 0, \ldots, a_{i,j}, a_{i,j+1}, \ldots, a_{i,M}]$ and $\mathbf{r} = [0, 0, \ldots, r_{k,j}, r_{k,j+1}, \ldots, r_{k,M}]$. Assuming, it is desired to annihilate the element $a_{i,j}$, then a rotation matrix, $\mathbf{G}(i, j, \theta)$, given as:

$$\mathbf{G}(i, j, \theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

can be constructed to rotate the vector $[r_{k,j}\, a_{i,j}]^T$ through an angle θ, where $\cos\theta$ and $\sin\theta$ are $r_{k,j}/\sqrt{r_{k,j}^2 + a_{i,j}^2}$ and $a_{k,j}/\sqrt{r_{k,j}^2 + a_{i,j}^2}$ respectively. It can be easily shown that

$$\mathbf{G}(i, j, \theta) \begin{bmatrix} r_{k,j} \\ a_{i,j} \end{bmatrix} = \begin{bmatrix} \sqrt{r_{k,j}^2 + a_{k,j}^2} \\ 0 \end{bmatrix}.$$

In order to completely triangularise $\mathbf{A}$, a series of $NM - M(M+1)/2$ such rotations are required. Since Givens rotation operates on only 2 rows at a time, it is attractive for parallel implementation. Furthermore, the trigonometric functions can be obtained without any square root operations by using CORDIC [34].

A.2 Gram-Schmidt Orthogonalisation

Gram-Schmidt orthogonalisation works by successively orthogonalising the columns of the matrix $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_N]$. Consider the matrix at the kth iteration as follows [109]:

$$\mathbf{A}^{(k)} = [\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_{k-1}, \ldots, \mathbf{a}_k^{(k)}, \ldots, \mathbf{a}_N^{(k)}],$$

where $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{k-1}$ denotes the columns of the orthogonal matrix, $\mathbf{Q}$. Then $r_{k,k}$ is determined as follows:

$$r_{k,k} = \sqrt{\mathbf{a}_k \mathbf{a}_k^T}$$

and

$$\mathbf{q}_k = \mathbf{a}_k^{(k)} / r_{k,k}.$$

The remaining columns of $\mathbf{A}$ are then orthogonalised against $\mathbf{q}_k$ as follows:

$$\mathbf{a}_k^{(k+1)} = \mathbf{a}_j^{(k)} - r_{k,j}\mathbf{q}_k$$

where $r_{k,j} = \mathbf{q}_k^T \mathbf{a}_j^{(k)}$ for $j = k+1, \ldots, N$. Gram-Schmidt orthogonalisation is attractive since the columns of $\mathbf{A}$ can be used for storing the columns of the orthogonal matrix as they are generated.

A.3 Householder Reflections

Unlike the Givens rotation, the Householder QR decomposition operates on $\mathbf{A}$ in a columnwise fashion and annihilates all the elements below the diagonal element. The matrix is triangularised by a series of multiplications with a transformation matrix as follows:

$$\mathbf{R} = \mathbf{P}^{(N)}\mathbf{P}^{(N-1)} \ldots \mathbf{P}^{(1)}\mathbf{A}.$$

Each multiplication results in a new matrix, $\mathbf{A}^{(k)}$, with the off-diagonal elements in the kth column annihilated. The Householder transformation matrix, $\mathbf{P}$, is given as

$$\mathbf{P}_{N\times 1} = \mathbf{I}_{N\times N} - 2\mathbf{v}\mathbf{v}^T,$$

where

$$\mathbf{v} = \frac{\mathbf{a}_k \pm \alpha e_1}{|\mathbf{a}_k \pm \alpha e_1|},$$

and α is the norm of the kth column of $\mathbf{A}^{(k-1)}$, $\|\mathbf{a}_k\|$, and the sign of α is chosen so as to prevent cancellation.

An attractive feature of the Householder method is its speed, since it eliminates several elements at once. It is also more numerically stable than the Gram-Schmidt orthogonalisation. However, constructing the transformation matrix is cumbersome, due to the square root computation involved. Furthermore, the Householder method is less parallelisable than the Givens rotation since each reflection affects the entire matrix.

Appendix B

MIMO Communication Testbed

The following code shows a sample MIMO environment used for simulating the algorithms described in the thesis. The code shows a complete MIMO system employing spatial multiplexing with 4×4 antennas. Zero-forcing linear detector is used for the detection at the receiver. It is assumed that there is no correlation between the parallel substreams from the transmitter i.e. the channel gains are completely random.

```
clc;
clear;

N = 4;             % Number of transmit and receive antennas
EbNoVec = 0:2:30;  % Eb/No in dB
M = 64;            % 64 QAM

constellation = qammod(0:M-1,M);

% Random stream used by random number generators for repeatability
hStr = RandStream('mt19937ar');

% Create QAM modulator and demodulator system objects
hMod = comm.RectangularQAMModulator(...
    'ModulationOrder',  M, ...
    'PhaseOffset',      0, ...
    'BitInput',         true, ...
    'SymbolMapping', 'Binary'...
);

hDemod = comm.RectangularQAMDemodulator( ...
    'ModulationOrder',  M, ...
    'PhaseOffset',      0, ...
    'BitOutput',        true, ...
    'SymbolMapping', 'Binary' ...
);

% Create error rate calculation system objects
hZFBERCalc = comm.ErrorRate;
BER_ZF = zeros(length(EbNoVec), 3);
bits_per_symbol = log2(M);
nchannels = 100e3;
```

```
% Number of symbol vectors per channel realisation
symbs_per_channel = 4;
L = nchannels * symbs_per_channel;

for idx = 1:length(EbNoVec)
    fprintf('EbNo = %d dB\n', EbNoVec(idx));

    % Reset error rate calculation system object
    reset(hZFBERCalc);

    % Calculate SNR from EbNo
    snrIndB = EbNoVec(idx) + 10 * log10(bits_per_symbol * N);
    hStr = RandStream('mt19937ar', 'Seed', EbNoVec(idx));

    indChans = 1/sqrt(2) .* (randn(hStr,N,N,nchannels) + 1j*randn(hStr,N,N,nchannels));
    H = indChans(:,:,ceil(1/symbs_per_channel:1/symbs_per_channel:nchannels));

    for j = 1:L
        msg = randi(hStr, [0 1], [N * bits_per_symbol, 1]);
        h = H(:,:,j);

        % Modulate data
        s = step(hMod, msg);
        y = awgn(h*s, snrIndB, 'measured', hStr);

        % MIMO detection using the ZF detector
        s_ZF = h^-1 * y;

        s_ZF_sliced = zeros(N,1);

        % Round soft values to nearest constellation point
        for i = 1:N
            [~, min_index] = min(abs(s_ZF(i) - constellation));
            s_ZF_sliced(i) = constellation(min_index);
        end

        % Demodulation
        x_ZF = step(hDemod, s_ZF_sliced);

        % Update BER
        BER_ZF(idx, :) = step(hZFBERCalc, x_ZF, msg);
    end
end

semilogy(EbNoVec, BER_ZF(:,1), 'ko-');
title('BER versus SNR');
xlabel('Eb/No');
ylabel('BER');
grid on;
```

Bibliography

[1] J. Andrews, S. Buzzi, W. Choi, S. Hanly, A. Lozano, A. Soong, and J. Zhang, "What will 5g be?" *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1065–1082, Jun. 2014.

[2] D. Gesbert, M. Shafi, D.-s. Shiu, P. Smith, and A. Naguib, "From theory to practice: An overview of MIMO space-time coded wireless systems," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 3, pp. 281 –302, Apr. 2003.

[3] G. J. Foschini, "Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas," *Bell Labs Technical Journal*, pp. 41–59, 1996.

[4] S. Alamouti, "A simple transmit diversity technique for wireless communications," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 8, pp. 1451 –1458, Oct. 1998.

[5] L. Hanzo, M. El-Hajjar, and O. Alamri, "Near-capacity wireless transceivers and cooperative communications in the MIMO era: Evolution of standards, waveform design, and future perspectives," *Proceedings of the IEEE*, vol. 99, no. 8, pp. 1343–1385, 2011.

[6] H. Bolcskei, "MIMO-OFDM wireless systems: Basics, perspectives, and challenges," *IEEE Wireless Communications*, vol. 13, no. 4, pp. 31–37, Aug. 2006.

[7] M. Damen, H. El Gamal, and G. Caire, "On maximum-likelihood detection and the search for the closest lattice point," *IEEE Transactions on Information Theory*, vol. 49, no. 10, pp. 2389–2402, Oct. 2003.

[8] D. Feng, C. Jiang, G. Lim, J. Cimini L.J., G. Feng, and G. Li, "A survey of energy-efficient wireless communications," *IEEE Communications Surveys Tutorials*, vol. 15, no. 1, pp. 167–178, 2013.

[9] B. Halak, M. El-Hajjar, O. H. Toma, and Z. Cheng, "Energy-efficient hardware implementation of an LR-aided k-best MIMO decoder for 5g networks," *Journal of Low Power Electronics and Applications*, vol. 6, no. 3, p. 12, Jul. 14, 2016.

[10] A. Amara, F. Amiel, and T. Ea, "FPGA vs. ASIC for low power applications," *Microelectronics Journal*, vol. 37, no. 8, pp. 669–677, Aug. 2006.

[11] R. Puri, L. Stok, J. Cohn, D. Kung, D. Pan, D. Sylvester, A. Srivastava, and S. Kulkarni, "Pushing ASIC performance in a power envelope," in *Proceedings of the 40th Annual Design Automation Conference*, ser. DAC '03, New York, NY, USA: ACM, 2003, pp. 788–793.

[12] E. Viterbo and J. Boutros, "A universal lattice code decoder for fading channels," *IEEE Transactions on Information Theory*, vol. 45, no. 5, pp. 1639–1642, 1999.

[13] K.-w. Wong, C.-y. Tsui, R.-K. Cheng, and W.-H. Mow, "A VLSI architecture of a k-best lattice decoding algorithm for MIMO channels," in *IEEE International Symposium on Circuits and Systems, 2002. ISCAS 2002*, vol. 3, 2002, III–273–III–276 vol.3.

[14] I. A. Bello, B. Halak, M. El-Hajjar, and M. Zwolinski, "A survey of VLSI implementations of tree search algorithms for MIMO detection," *Circuits, Systems, and Signal Processing*, pp. 1–31, 2015.

[15] I. A. Bello, B. Halak, M. El-Hajjar, and M. Zwolinski, "VLSI implementation of a scalable k-best MIMO detector," in *2015 15th International Symposium on Communications and Information Technologies (ISCIT)*, Oct. 2015, pp. 281–286.

[16] M. El-Hajjar and L. Hanzo, "Multifunctional MIMO systems: A combined diversity and multiplexing design perspective," *IEEE Wireless Communications*, vol. 17, no. 2, pp. 73–79, 2010.

[17] D. Gesbert, T. Ekman, and N. Christophersen, "Capacity limits of dense palm-sized MIMO arrays," in *IEEE Global Telecommunications Conference, 2002. GLOBECOM '02*, vol. 2, Nov. 2002, 1187–1191 vol.2.

[18] B. Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Transactions on Communications*, vol. 51, no. 3, pp. 389–399, Mar. 2003.

[19] D. Wubben, R. Bohnke, V. Kühn, and K.-D. Kammeyer, "MMSE extension of v-BLAST based on sorted QR decomposition," in *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, vol. 1, Oct. 2003, 508–512 Vol.1.

[20] H. Anton, *Elementary linear algebra*. Wiley. com, 2010.

[21] S. Sugiura, S. Chen, and L. Hanzo, "MIMO-aided near-capacity turbo transceivers: Taxonomy and performance versus complexity," *IEEE Communications Surveys Tutorials*, vol. 14, no. 2, pp. 421–442, 2012.

[22] P. Wolniansky, G. Foschini, G. Golden, and R. Valenzuela, "V-BLAST: An architecture for realizing very high data rates over the rich-scattering wireless channel," in *1998 URSI International Symposium on Signals, Systems, and Electronics, 1998. ISSSE 98*, Oct. 2, 1998, pp. 295 –300.

[23] B. Gestner, X. Ma, and D. Anderson, "Incremental lattice reduction: Motivation, theory, and practical implementation," *IEEE Transactions on Wireless Communications*, vol. 11, no. 1, pp. 188–198, 2012.

[24] H. Yao and G. W. Wornell, "Lattice-reduction-aided detectors for MIMO communication systems," in *IEEE Global Telecommunications Conference, 2002. GLOBECOM '02*, vol. 1, 2002, 424–428 vol.1.

[25] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, no. 4, pp. 515–534, 1982.

[26] D. Wubben, D. Seethaler, J. Jalden, and G. Matz, "Lattice reduction," *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 70–91, May 2011.

[27] Y. H. Gan, C. Ling, and W.-H. Mow, "Complex lattice reduction algorithm for low-complexity full-diversity MIMO detection," *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2701–2710, 2009.

[28] B. Gestner, W. Zhang, X. Ma, and D. Anderson, "Lattice reduction for MIMO detection: From theoretical analysis to hardware realization," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 4, pp. 813–826, 2011.

[29] H. Vetter, V. Ponnampalam, M. Sandell, and P. A. Hoeher, "Fixed complexity LLL algorithm," *Signal Processing, IEEE Transactions on*, vol. 57, no. 4, pp. 1634–1637, 2009.

[30] M. Shabany, A. Youssef, and G. Gulak, "High-throughput 0.13µm CMOS lattice reduction core supporting 880 mb/s detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 5, pp. 848–861, May 2013.

[31] D. Wu, J. Eilert, and D. Liu, "Lattice-reduction aided multi-user STBC decoding with resource constraints," in *IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007*, Sep. 2007, pp. 1–5.

[32] C. Windpassinger, L. Lampe, R. F. H. Fischer, and T. Hehn, "A performance study of MIMO detectors," *IEEE Transactions on Wireless Communications*, vol. 5, no. 8, pp. 2004–2008, Aug. 2006.

[33] A. Murugan, H. El Gamal, M. Damen, and G. Caire, "A unified framework for tree search decoding: Rediscovering the sequential decoder," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 933–953, Mar. 2006.

[34] P. K. Meher, J. Valls, T. B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, Sep. 2009.

[35] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm i. expected complexity," *IEEE Transactions on Signal Processing*, vol. 53, no. 8, pp. 2806–2818, 2005.

[36] J. Jalden and B. Ottersten, "On the complexity of sphere decoding in digital communications," *IEEE Transactions on Signal Processing*, vol. 53, no. 4, pp. 1474–1484, Apr. 2005.

[37] M. Wenk, M. Zellweger, A. Burg, N. Felber, and W. Fichtner, "K-best MIMO detection VLSI architectures achieving up to 424 mbps," in *2006 IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006. Proceedings*, 2006, 4 pp.–1154.

[38] L. Azzam and E. Ayanoglu, "Reduced complexity sphere decoding for square QAM via a new lattice representation," in *IEEE Global Telecommunications Conference, 2007. GLOBECOM '07*, Nov. 2007, pp. 4242–4246.

[39] T.-H. Liu, "Comparisons of two real-valued MIMO signal models and their associated ZF-SIC detectors over the rayleigh fading channel," *IEEE Transactions on Wireless Communications*, vol. 12, no. 12, pp. 6054–6066, Dec. 2013.

[40] B. Wu and G. Masera, "A novel VLSI architecture of fixed-complexity sphere decoder," in *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD)*, 2010, pp. 737–744.

[41] U Fincke and M Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis," *Math. Comp*, vol. 44, pp. 463 –471, 1985.

[42] L. Babai, "On lovász lattice reduction and the nearest lattice point problem," *Combinatorica*, vol. 6, no. 1, pp. 1–13, 1986.

[43] C. P. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems.," in *Math. Programming*, 1993, pp. 181–191.

[44] A. Burg, M. Wenk, M. Zellweger, M. Wegmueller, N. Felber, and W. Fichtner, "VLSI implementation of the sphere decoding algorithm," in *Solid-State Circuits Conference, 2004. ESSCIRC 2004. Proceeding of the 30th European*, 2004, pp. 303–306.

[45] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 7, pp. 1566 –1577, Jul. 2005.

[46] N. Moezzi-Madani and W. R. Davis, "High-throughput low-complexity MIMO detector based on k-best algorithm," in *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, 2009, pp. 451–456.

[47] P.-Y. Tsai, W.-T. Chen, X.-C. Lin, and M.-Y. Huang, "A 4x4 64-QAM reduced-complexity k-best MIMO detector up to 1.5gbps," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 3953–3956.

[48] L. Barbero and J. Thompson, "A fixed-complexity MIMO detector based on the complex sphere decoder," in *IEEE 7th Workshop on Signal Processing Advances in Wireless Communications, 2006. SPAWC '06*, 2006, pp. 1–5.

[49] ——, "Rapid prototyping of a fixed-throughput sphere decoder for MIMO systems," in *IEEE International Conference on Communications, 2006. ICC '06*, vol. 7, 2006, pp. 3082–3087.

[50] ——, "Fixing the complexity of the sphere decoder for MIMO detection," *IEEE Transactions on Wireless Communications*, vol. 7, no. 6, pp. 2131–2142, Jun. 2008.

[51] J Anderson and S. Mohan, "Sequential coding algorithms: A survey and cost analysis," *Communications, IEEE Transactions on*, vol. 32, no. 2, pp. 169–176, 1984.

[52] Y. Dai and Z. Yan, "Memory-constrained tree search detection and new ordering schemes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 3, no. 6, pp. 1026–1037, Dec. 2009.

[53] C.-H. Liao, T.-P. Wang, and T.-D. Chiueh, "A 74.8 mW soft-output detector IC for 8 x 8 spatial-multiplexing MIMO communications," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 2, pp. 411–421, 2010.

[54] C.-A. Shen, A. M. Eltawil, S. Mondal, and K. N. Salama, "A best-first tree-searching approach for ML decoding in MIMO system," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 2010, pp. 3533–3536.

[55] Y. Dai, S. Sun, and Z. Lei, "A comparative study of QRD-m detection and sphere decoding for MIMO-OFDM systems," in *IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005*, vol. 1, Sep. 2005, pp. 186–190.

[56] C. Studer and H. Bolcskei, "Soft-input soft-output single tree-search sphere decoding," *IEEE Transactions on Information Theory*, vol. 56, no. 10, pp. 4827–4842, 2010.

[57] S. Baro, J. Hagenauer, and M. Witzke, "Iterative detection of MIMO transmission using a list-sequential (LISS) detector," in *IEEE International Conference on Communications, 2003. ICC '03*, vol. 4, May 2003, 2653–2657 vol.4.

[58] C.-H. Liao, I.-W. Lai, K. Nikitopoulos, F. Borlenghi, D. Kammler, M. Witte, D. Zhang, T.-D. Chiueh, G. Ascheid, and H. Meyr, "Combining orthogonalized partial metrics: Efficient enumeration for soft-input sphere decoder," in *Personal, Indoor and Mobile Radio Communications, 2009 IEEE 20th International Symposium on*, IEEE, 2009, pp. 1287–1291.

[59] C. Studer, A. Burg, and H. Bolcskei, "Soft-output sphere decoding: Algorithms and VLSI implementation," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 2, pp. 290–300, Feb. 2008.

[60] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct. 1974.

[61] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," *Integration, the VLSI Journal*, vol. 58, pp. 74–81, Jun. 2017.

[62] A. Chandrakasan and R. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, Apr. 1995.

[63] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, 2003.

[64] A. Burg, M. Borgmann, C. Studer, and H. Bolcskei, "Advanced receiver algorithms for MIMO wireless communications," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, Munich, Mar. 6, 2006, p. 6.

[65] F. Borlenghi, E. Witte, G. Ascheid, H. Meyr, and A. Burg, "A 772mbit/s 8.81bit/nJ 90nm CMOS soft-input soft-output sphere decoder," in *Solid State Circuits Conference (A-SSCC), 2011 IEEE Asian*, Nov. 2011, pp. 297–300.

[66] K.-J. Yang, S.-H. Tsai, R.-C. Chang, Y.-C. Chen, and G.-H. Chuang, "VLSI implementation of a low complexity 4x4 MIMO sphere decoder with table enumeration," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 2167–2170.

[67] A. Wiesel, X. Mestre, A. Pages, and J. Fonollosa, "Efficient implementation of sphere demodulation," in *4th IEEE Workshop on Signal Processing Advances in Wireless Communications, 2003. SPAWC 2003*, 2003, pp. 36–40.

[68] R. Jenkal and R. Davis, "An architecture for energy efficient sphere decoding," in *2007 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2007, pp. 244–249.

[69] Z. Guo and P. Nilsson, "Algorithm and implementation of the k-best sphere decoding for MIMO detection," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 3, pp. 491–503, 2006.

[70] M. Shabany and P. Gulak, "Scalable VLSI architecture for k-best lattice decoders," in *IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008*, 2008, pp. 940–943.

[71] ——, "A 0.13 µm CMOS 655mb/s 4x4 64-QAM k-best MIMO detector," in *Solid-State Circuits Conference - Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, 2009, 256–257,257a.

[72] D. Patel, V. Smolyakov, M. Shabany, and P. Gulak, "VLSI implementation of a WiMAX/LTE compliant low-complexity high-throughput soft-output k-best MIMO detector," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 593–596.

[73] T.-H. Kim and I.-C. Park, "Small-area and low-energy -best MIMO detector using relaxed tree expansion and early forwarding," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 10, pp. 2753–2761, 2010.

[74] L. Liu, F. Ye, X. Ma, T. Zhang, and J. Ren, "A 1.1-gb/s 115-pJ/bit configurable MIMO detector using 0.13-µm CMOS technology," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 9, pp. 701–705, 2010.

[75] N. Moezzi-Madani, T. Thorolfsson, and W. Davis, "A low-area flexible MIMO detector for WiFi/WiMAX standards," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, Mar. 2010, pp. 1633–1636.

[76] N. Moezzi-Madani and W. R. Davis, "Parallel merge algorithm for high-throughput signal processing applications," *Electronics letters*, vol. 45, no. 3, pp. 188–189, 2009.

[77] M.-Y. Huang and P.-Y. Tsai, "Toward multi-gigabit wireless: Design of high-throughput MIMO detectors with hardware-efficient architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 2, pp. 613–624, Feb. 2014.

[78] L. Liu, J. Lofgren, and P. Nilsson, "Area-efficient configurable high-throughput signal detector supporting multiple MIMO modes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 9, pp. 2085–2096, 2012.

[79] X. Chen, G. He, and J. Ma, "VLSI implementation of a high-throughput iterative fixed-complexity sphere decoder," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 5, pp. 272–276, 2013.

[80] S. Carlsson, "The deap—a double-ended heap to implement double-ended priority queues," *Information Processing Letters*, vol. 26, no. 1, pp. 33–36, Sep. 15, 1987.

[81] C.-A. Shen, A. Eltawil, K. Salama, and S. Mondal, "A best-first soft/hard decision tree searching MIMO decoder for a 4 4 64-QAM system," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 8, pp. 1537–1541, 2012.

[82] V. T. Pham, Z. Lei, and T. T. Tjhung, "Fano detection algorithm for MIMO systems," in *Information, Communications & Signal Processing, 2007 6th International Conference on*, 2007, pp. 1–5.

[83] Z. Nikolić, H. T. Nguyen, and G. Frantz, "Design and implementation of numerical linear algebra algorithms on fixed point DSPs," *EURASIP Journal on Advances in Signal Processing*, vol. 2007, no. 1, p. 087 046, 2007.

[84] P. J. Ashenden, *Digital design (verilog): An embedded systems approach using verilog*. Elsevier, 2007.

[85] P. Coussy, G. Lhairech-Lebreton, D. Heller, and E. Martin, "GAUT—a free and open source high-level synthesis tool," *IEEE Design Automation and Test in Europe-University Booth*, 2010.

[86] Synopsys Inc., *Design compiler user guide*, C-2009.06. Synopsys, 2009.

[87] ——, *Power compiler user guide*, D-2010.03-SP2. Synopsys, 2010.

[88] C. Studer, "Sphere decoding with resource constraints," Master's Thesis, 2005.

[89] M. Gamba and G. Masera, "Look-ahead sphere decoding: Algorithm and VLSI architecture," *IET Communications*, vol. 5, no. 9, pp. 1275–1285, 2011.

[90] S.-H. Kang and I.-C. Park, "High speed sphere decoding based on vertically incremental computation," in *IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007*, May 2007, pp. 665–668.

[91] L. G. Barbero and J. S. Thompson, "Rapid prototyping of the sphere decoder for MIMO systems," pp. 4–4, Jan. 1, 2005.

[92] A. Burg, M. Wenk, and W. Fichtner, "VLSI implementation of pipelined sphere decoding with early termination," in *2006 14th European Signal Processing Conference*, Sep. 2006, pp. 1–5.

[93] C. Gimmler-Dumont, F. Kienle, B. Wu, and G. Masera, "A system view on iterative MIMO detection: Dynamic sphere detection versus fixed effort list detection," *VLSI Design*, vol. 2012, p. 2, 2012.

[94] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30-May 2, 1968, spring joint computer conference*, ACM, 1968, pp. 307–314.

[95] O. Astrachan, "Bubble sort: An archaeological algorithmic analysis," in *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '03, New York, NY, USA: ACM, 2003, pp. 1–5.

[96] S. Mondal, A. Eltawil, C.-A. Shen, and K. Salama, "Design and implementation of a sort-free k-best sphere decoder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 10, pp. 1497–1501, 2010.

[97] M. Mahdavi and M. Shabany, "Novel MIMO detection algorithm for high-order constellations in the complex domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 5, pp. 834–847, May 2013.

[98] N. Moezzi-Madani, T. Thorolfsson, P. Chiang, and W. R. Davis, "Area-efficient antenna-scalable MIMO detector for k-best sphere decoding," *Journal of Signal Processing Systems*, vol. 68, no. 2, pp. 171–182, Aug. 2012.

[99] S. Chen, T. Zhang, and Y. Xin, "Relaxed k-best MIMO signal detector design and VLSI implementation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 3, pp. 328–337, 2007.

[100] B. R. Rau and J. A. Fisher, "Instruction-level parallel processing: History, overview, and perspective," *J Supercomput*, vol. 7, no. 1, pp. 9–50,

[101] C. Ju, J. Ma, C. Tian, and G. He, "VLSI implementation of an 855 mbps high performance soft-output k-best MIMO detector," in *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2012, pp. 2849–2852.

[102] M. Mahdavi and M. Shabany, "A 13 gbps, 0.13 μm CMOS, multiplication-free MIMO detector," *J Sign Process Syst*, pp. 1–13, Jun. 6, 2016.

[103] E. Perahia and M. X. Gong, "Gigabit wireless LANs: An overview of IEEE 802.11 ac and 802.11 ad," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 15, no. 3, pp. 23–33, 2011.

[104] IEEE Computer Society LAN MAN Standards Committee, *"IEEE std 802.11ac-2013 (amendment to IEEE std 802.11-2012)", IEEE standard for information technology-telecommunications and information exchange between systems local and metropolitan area networks-specific requirements-part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications-amendment 4: Enhancements for very high throughput for operation in bands below 6 GHz*. The Institute of Electrical and Electronics Engineers, 2013.

[105] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, "Closest point search in lattices," *IEEE Transactions on Information Theory*, vol. 48, no. 8, pp. 2201–2214, 2002.

[106] B. Hassibi, "An efficient square-root algorithm for BLAST," in *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, vol. 2, 2000, pp. II737–II740.

[107] Y. Liu, T. Zhang, and J. Hu, "Design of voltage overscaled low-power trellis decoders in presence of process variations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 3, pp. 439–443, Mar. 2009.

[108] Y. Lin, M. Zwolinski, and B. Halak, "A low-cost radiation hardened flip-flop," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2014, pp. 1–6.

[109] Å. Björck, "Numerics of gram-schmidt orthogonalization," *Linear Algebra and its Applications*, vol. 197, pp. 297–316, Jan. 1, 1994.