Algorithms and architectures for the detection of MIMO signals

Micaela Troglia Gamba

N° d'ordre: 2011telb0000

Sous le sceau de l'Université européenne de Bretagne

Télécom Bretagne

En accréditation conjointe avec l'Ecole Doctorale Sicma

Co-tutelle avec le Politecnico di Torino (Italie)

Algorithms and architectures for the detection of MIMO signals

---

Thèse de Doctorat

Mention: ***STIC (Sciences et Technologies de l'Information et de la Communication)***

Présentée par **Micaela Troglia Gamba**

Département: Electronique
Laboratoire: Lab-STICC Pôle: CACS

Soutenue le 13 Avril 2011

Composition du Jury :

M. Valentino Liberali, Professeur à l'Université de Milan	Rapporteur
M. Christophe Jego, Professeur à l'ENSEIRB-MATMECA	Rapporteur
M. Leonardo Reyneri, Professeur à Politecnico di Torino	Président
M. Guido Masera, Professeur à Politecnico di Torino	Examinateur
M. Amer Baghdadi, Maître de conférences à Télécom Bretagne	Examinateur
M. Michel Jézéquel, Professeur à Télécom Bretagne	Directeur

Résumé

Les systèmes multi-antennes (Multiple-Input Multiple-Output: MIMO) représentent incontestablement une technologie clé pour le déploiement de systèmes de communication sans fil de haute performance. Cependant, la complexité des détecteurs MIMO de haut débit pose un problème sérieux de mise en œuvre. Parmi les détecteurs MIMO existants, l'algorithme nommé détecteur à sphère (Sphere Decoder Algorithm: SDA) a vu le jour pour réduire la complexité de traitement par rapport à la technique de détection originale basée sur le maximum de vraisemblance (ML). En outre, il a été démontré que SDA atteint des performances optimales pour les systèmes non codés. Toutefois, pour les systèmes codés, d'autres simplifications dans l'algorithme de détection peuvent être utilisées sans altérer les performances en taux d'erreur si un processus itératif de détection et de décodage canal est adopté dans le récepteur. Un tel traitement itératif avec un décodeur canal offre une amélioration significative de la performance en taux d'erreur pour des faibles rapports signal sur bruit. Dans ce contexte, le SDA peut être simplifié davantage et modifié afin de prévoir une détection basée sur des informations pondérées. LSD (List-Sphere Decoder) a été introduit comme une version à entrées/sorties pondérées de la version SDA originale.
Ce travail de thèse traite les aspects algorithmique, architectural et de mise en œuvre de la détection MIMO basée sur SDA et LSD. Le principal objectif des travaux menés est de proposer des solutions de mise en œuvre de faible complexité, tout en considérant les exigences des systèmes avancés de communication numérique en termes de débit, de flexibilité et de taux d'erreurs.

En particulier, la première contribution est représentée par une amélioration de SDA, ce qui permet une augmentation significative du débit avec une complexité supplémentaire très limitée et sans dégradation en termes de performance en taux derreurs binaires. La méthode de détection proposée, appelée LASDA (Look-Ahead SDA), est basée sur des transformations formelles de l'algorithme, à savoir Look-Ahead, recalage et pipeline et sur une stratégie modifiée de recherche arborescente. Une conception efficace de type VLSI du détecteur LASDA supportant un system MIMO 4×4 avec une modulation MAQ-16 est proposée. Ciblant une technologie CMOS 130 nm, les résultats de synthèse montrent que la solution proposée atteint un débit moyen de 380 Mbps à un rapport signal sur bruit de 22 dB, avec une surface occupée de $0.18mm^2$ de silicium. Des comparaisons avec un certain nombre de mises en œuvre précédentes sont également fournies.

La deuxième contribution concerne une étude détaillée sur la flexibilité et la convergence de la détection itérative avec un décodage canal. À cet égard, deux détecteurs à entrées/sorties pondérées (Soft-Input Soft-Output: SISO) sont considérés. Le premier est basé sur LSD et le deuxième est basé sur un filtrage linéaire de faible complexité (Linear Minimum-Mean-Square-Error-Interference-Canceller (MMSE-IC)). Pour les deux techniques, l'impact des différents paramètres de configuration du système en termes d'ordre de modulation et du nombre d'antennes est illustré et discuté. Des diagrammes de transfert d'information mutuelle (EXIT charts) sont développés afin d'analyser en détail le comportement et la convergence du processus itératif. Cette analyse est orientée pour obtenir les compromis possibles entre complexité et performance pour une implémentation matérielle flexible.

La dernière contribution est liée à la proposition et la conception d'un processeur à jeu d'instructions dédié à l'application (Application-Specific Instruction-set Processor (ASIP)) pour un détecteur SISO LSD. L'ASIP proposé supporte différentes configurations MIMO ($2 \times$

2, 3×3, 4×4) et différents ordres de modulation (QPSK, MAQ-16, MAQ-64) en plus d'une taille de liste flexible (1 à 64 éléments). Les résultats de synthèse sur une technologie CMOS 130 nm sont détaillés en termes de débit et surface de silicium occupée, et comparés avec d'autres implémentations dans ce domaine.

Abstract

Multiple Input Multiple Output (MIMO) systems are recognized as a key enabling technology in high performance wireless communications. However the complexity of high throughput MIMO detectors poses a serious implementation issue. Among known MIMO detectors, Sphere Decoder Algorithm (SDA) has emerged to reduce the processing complexity, with respect to the original Maximum Likelihood (ML) detection. Moreover, it has been demonstrated that the SDA achieves optimal performance for uncoded systems.

However, for coded systems, further simplifications in the detection algorithm can be used without altering the error rate performance if iterative detection and channel decoding is adopted in the receiver. Such an iterative processing with a channel decoder offers significant improvement in error-rate performance for a reduced signal-to-noise ratio. In this context, the SDA can be further simplified and modified in order to provide soft detection: "List Sphere Decoder" (LSD) has been introduced as a soft version of the original SDA.

This research thesis focuses on algorithmic, architectural and implementation aspects of the " Sphere Decoder Algorithm" and the "List Sphere Decoder". The main objective of the conducted work is to propose area-efficient implementation solutions while considering throughput, flexibility, and error rate performance requirements of advanced digital communication systems.

In particular, the first contribution is represented by an improved SDA, which enables significant throughput increase at a very limited additional complexity and with no degradation in terms of Bit Error Rate performance. The proposed detection method, called LASDA (Look–Ahead SDA) is based on formal algorithm transformations,

namely look–ahead, retiming and pipelining, besides a modified tree search strategy. An efficient VLSI design of LASDA detector supporting a 4×4 MIMO channel with 16 QAM modulation is proposed. Targeting 130 nm CMOS standard cell technology, synthesis results show that the proposed solution achieves an average throughput of 380 Mbps at a signal to noise ratio of 22 dB, with an occupied Silicon area of 0.18 mm^2. Comparisons with a number of previous implementations are also provided.

The second contribution concerns a detailed study on flexibility and convergence of iterative detection and channel decoding. In this regard, two Soft-Input Soft-Output detectors are considered. The first one is based on List Sphere Decoding and the second is based on a low complexity linear filtering (Linear Minimum-Mean-Square-Error-Interference-Canceller (MMSE-IC)). For both techniques, impact of the different system configuration parameters in terms of modulation order and number of antennas is illustrated and discussed. Extrinsic Information Transfer (EXIT) charts are developed in order to thoroughly analyze the behavior and the convergence of the iterative process. This analysis is oriented to obtain possible performance-complexity trade-offs for a flexible hardware implementation.

The last contribution is related to the proposal and design of an Application-Specific-Instruction set-Processor (ASIP) for SISO List Sphere Decoding. The proposed ASIP supports different MIMO system configurations (2×2, 3×3, 4×4) and modulation orders (QPSK, 16QAM, 64QAM) besides a flexible list size (from 1 to 64 elements). Synthesis results for a 130 nm technology are detailed in terms of throughput and occupied Silicon area, and compared with other related implementations.

To my parents

Contents

List of Figures

List of Tables

Listings

Quand je vous aimerai? Ma foi, je ne sais pas, Peut-être jamais, peut-être demain. Mais pas aujourd'hui, c'est certain.
L'amour est un oiseau rebelle Que nul ne peut apprivoiser, Et c'est bien en vain qu'on l'appelle, S'il lui convient de refuser. Rien n'y fait, menace ou prière, L'un parle bien, l'autre se tait: Et c'est l'autre que je préfère, Il n'a rien dit mais il me plaît. L'amour! L'amour! L'amour! L'amour!
L'amour est enfant de Bohème, Il n'a jamais, jamais connu de loi; Si tu ne m'aimes pas, je t'aime: Si je t'aime, prends garde à toi! Si tu ne m'aimes pas, Si tu ne m'aimes pas, je t'aime! Mais, si je t'aime, Si je t'aime, prends garde à toi! Si tu ne m'aimes pas, Si tu ne m'aimes pas, je t'aime! Mais, si je t'aime, Si je t'aime, prends garde à toi!
L'oiseau que tu croyais surprendre Battit de l'aile et s'envola... L'amour est loin, tu peux l'attendre; Tu ne l'attends plus, il est là! Tout autour de toi, vite, vite, Il vient, s'en va, puis il revient... Tu crois le tenir, il t'évite, Tu crois l'éviter, il te tient. L'amour! L'amour! L'amour! L'amour! L'amour est enfant de Bohème, Il n'a jamais, jamais connu de loi; Si tu ne m'aimes pas, je t'aime: Si je t'aime, prends garde à toi!

FROM "CARMEN" OF GEORGE BIZET

1

Introduction

MIMO technology improves performance of wireless links, at the cost of high computational complexity both at transmitter and receiver sides. In particular, at the receiver end, the detector has to recover from the received signal vector the original data stream corresponding to each transmitting antenna. The optimal receiver for uncoded MIMO systems is the Maximum Likelihood detector (ML). However the direct implementation of ML methods has exponential computational complexity, which makes this approach unattractive, except for low order MIMO systems [14]. Sphere Decoder Algorithm (SDA) has been proposed as an alternative method, able to guarantee ML performance, with polynomial processing complexity [14]. SDA reduces the average computational effort by only considering points that lie inside a hypersphere.

SDA can be formulated as a tree search algorithm, which adopts a *depth–first* search strategy and results in ML performance and variable throughput (see for example [1] [3]). Alternative search methods, such as *breadth–first* search strategies, normally achieve fixed throughput at the cost of sub–optimal detection performance (see for example *K–best* algorithms [15] [16] [17]).

All the works mentioned above have the common characteristic of showing a decline in performance with respect to the optimal receiver.

While most of these solutions achieve performance that is close to ML, this research work faces the problem of a high throughput full ML implementation.

The main contributions of this work are the following:

1. The traditional SDA has been modified at two levels. At the digital signal

processing level, *look–ahead*, *pipelining* and *retiming* techniques have been exploited to enable higher clock frequency with respect to traditional SDA implementations. At the control level, a modified search strategy has been adopted. These modifications lead to a new SDA implementation scheme, called LASDA (Look–Ahead SDA).

2. To show the potential of the proposed solution and to estimate its area overhead, ASIC design has been completed for the above mentioned LASDA; compared to direct SDA implementations, the designed detector shows higher throughput with a small increase of occupied Silicon area. Moreover the faster detector maintains ML Bit Error Rate (BER) performance.

More recently, the interest of researchers has focused on MIMO detection algorithms providing soft–output information. Particularly, it has been shown that near–capacity can be achieved on a multiple–antenna channel by extending the maximum a posteriori (MAP) processing techniques developed for modern channel codes to the iterative detection and decoding of a linear space–time mapper combined with an outer channel code [18]. Proper extensions of the SDA have been proposed in this context [18] [19] [4] [20]: the List Sphere Decoder (LSD) was first introduced in [18] to enable joint detection and decoding; single tree search is exploited in [19] [4] as an alternative to repeated tree search technique to improve computational efficiency; a smart management of the tree search is proposed in [20] to achieve low complexity. [21] reports synthesis results for soft–in soft–out single tree search MIMO detectors implemented with a 90 nm technology. In [12] a soft–output detector is proposed for MIMO-OFDM systems like IEEE 802.11n WLANs; the work exploits Layered ORthogonal Lattice Detection (LORD) to simplify processing and achieves maxLogAPP performance. In [22] near ML performance with fixed throughput is achieved for soft MIMO detection thanks to a graph based greedy algorithm.

The high throughput ML solution given in this work is also applicable for most soft–output detection schemes, since both hard and soft–output sphere decoders share several key processing units.

The LASDA solution represents the first contribution of this work. The second one is related to the List Sphere Decoder, first proposed in [23] as soft version of the original SD: it finds a list of possible candidates and computes soft information according to this list. The choice of the list size is critical as it affects both computational complexity and Bit-Error-Rate (BER) performance. This is particularly true in iterative Soft-Input Soft-Output (SISO) detection and channel decoding. In such systems, short lists tend to reduce the reliability of soft outputs and this can cause the problem of divergence [24][10]. This problem can, however, be limited by constraining the *a priori* information coming from the channel decoder [10]. The main contributions are the following:

1. To illustrate the problem of divergence, adopting the Extrinsic Information Transfer (EXIT) charts [25].
2. To compare in terms of flexibility and performance the LSD with the Minimum-Mean-Squared-Error Interference-Canceller (MMSE-IC), which is an attractive alternative in MIMO detection mainly due to its reduced complexity [26].

The ambitious aim of this analysis is to the design a multi-mode flexible MIMO detector supporting both MMSE-IC and LSD with variable list size, according to channel conditions and targeted performance.

The previous analysis on possible complexity-performance trade-offs is the preamble of the third contribution of this thesis. Concerning the flexibility, one of the most flexible kind of implementation, exploited in literature, is certainly the Application-Specific-Instruction set-Processor (ASIP): it is a programmable microprocessor, where hardware and instruction set are designed together for one special application. It currently represents the best performance-flexibility trade-off, being more flexible than an Application-Specific-Integrated-Circuit (ASIC) and having better performance than a General-Purpose-Processor (GPP).
An ASIP implementation of the List Sphere Decoder is presented. In particular, the main contributions are the following:

1. To discuss which are the most critical parts, from an architectural point of view.

2. To propose at the best of our knowledge, the first ASIP of LSD,supporting multiple modulations, number of antennas and list sizes
3. To propose some modifications to the original design to increase the throughput, and to compare performance results with other existing implementations

This thesis is divided in two parts: the first part is dedicated to hard MIMO detection and the second part to the soft MIMO detection. In particular, the part I is composed of two chapters:

Chapter 2 is dedicated to present MIMO systems and MIMO functions. Some basic concepts and the mathematical model are introduced. Then, an overview of the most popular MIMO detection algorithms is reported and the Sphere Decoder Algorithm is detailed, with particular emphasis for the depth–first Schnorr-Euchner SDA, which is the starting point of this research thesis. For completeness, the chapter is concluded mentioning other versions of SDA, exploited in literature.

Chapter 3 presents a modified version of the SDA, which guarantees optimal performance and achieves an increase of throughput, with a limited additional complexity. All modifications applied to the SDA with the purpose of enabling a higher throughput implementation are detailed. In particular, some formal methods, such as look–ahead, pipelining and retiming are employed and described. Performance of this solution, called Look–Ahead SDA (LASDA) is estimated in terms of both achieved BER and required average number of clock cycles, needed to detect one symbol vector. The VLSI architecture design and the obtained implementation results are presented and compared with other existing solutions.

The part II of this thesis is composed of four chapters:

Chapter 4 represents a link between first and second parts. It explains the difference between hard and soft detection and the advantages achieved in a channel coded scheme, with respect to an uncoded one. Some hints on the most popular

detection-decoding schemes are done. A particular emphasis is given to the List Sphere Decoder (LSD), which is one of the soft versions of SDA, and a linear detector, such as the Minimum-Mean- Squared-Error (MMSE). Finally, to complete the scenario, a complexity evaluation of a suboptimal soft-output MIMO detector implementation is reported.

Chapter 5 is dedicated to analyze the existing hard and soft MIMO detector implementations. Such an analysis has been conducted to explain and justify the idea of a flexible multi-algorithm MIMO detector, supporting both LSD and MMSE.

In Chapter 6, a detailed analysis of flexibility parameters and divergence issue of iterative LSD is reported. The Extrinsic-Information-Transfer (EXIT) Charts are employed to study the iterative behavior of LSD and, in particular, to have a more precise answer to the problem of divergence, taking into account also the effect of different parameters, such as the size of the list. Finally, LSD and MMSE-IC are compared in terms of performance and complexity, justifying again the idea of a flexible multi-mode detector.

Chapter 7 details a proposed ASIP implementation for LSD. The chapter starts with an introduction to the ASIP methodology, and continues with a discussion on the architectural choices and flexibility parameters. A first suboptimal release of the ASIP is presented, with synthesis results. Then an improved version in terms of clock frequency is proposed. Performance, synthesis results and comparison with state of the art implementations are also reported. In order to increase the throughput, a more efficient usage of the pipeline has been developed and detailed. The chapter ends with a discussion of the results, accompanied by further possible improvements.

Part I

Hard MIMO detection

2

MIMO systems

A continuous increase in required data rate, system capacity and quality of service characterizes wireless applications. Multiple-Input-Multiple-Output (MIMO) systems are recognized as a key enabling technology for increasing capacity and spectral efficiency, because with multiple antennas at both sides of the wireless link they are able to improve communication reliability [27]. MIMO technology has been proposed to be incorporated into the fourth generation (4G) mobile communication systems to enhance voice and data transmission. Moreover, several current and future wireless communication standards, such as IEEE 802.11n wireless LAN (Local Area Network), IEEE 802.16e WiMax and 3GPP-LTE [28] already include MIMO techniques, often in combination with OFDM (Orthogonal Frequency-Division Multiplexing).

2.1 MIMO functions

MIMO systems are characterized by three main functions: *precoding*, *spatial multiplexing* and *diversity coding*, which are respectively associated with *array gain*, *spatial gain* and *diversity gain*.

1. **Precoding**: it is the multi-stream beamforming[1] technique, in the narrowest definition, and in a more general sense, the spatial processing done at the transmitter side. In the single layer beamforming, the same signal is emitted from each transmitting antennas, with a certain phase and gain , in order to maximize the signal power at the receiver side. The obtained advantages are:

to increase the received signal gain, thanks to the constructive interference, and to reduce the multi-path fading[2]. But when the receiver has multiple antennas, the beamforming cannot simultaneously maximize the signal level for all of the receive antennas, so that the precoding is employed. It is important to underline that this technique requires knowledge of the *channel state information* (CSI) at the transmitter.

2. **Spatial multiplexing**: an high rate signal is split into multiple lower rate streams and each stream is transmitted from a different transmit antenna in the same frequency channel. If these signals reach the receiver antenna with sufficiently different spatial signatures, they are equivalent to almost parallel channels and it is possible to separate them. The spatial multiplexing is a very powerful technique, able to improve the channel capacity at high signal-to-noise (SNR) ratio values. The maximum number of spatial streams is limited by the minimum between the number of transmitters and receivers. Spatial multiplexing can be used with or without CSI knowledge.
3. **Diversity coding**: it is employed when there is no channel knowledge at the transmitter. A single stream (unlike multiple streams in spatial multiplexing) is transmitted, but the signal is coded using techniques called space-time coding, which exploits full or near-full orthogonality. There are three main types of diversity: time diversity, frequency diversity and spatial diversity. The spatial diversity is related to the employ of spatially different links, affected by different fading. The time diversity ie related to total or partial retransmission of the information in different moments. The frequency diversity is due to frequency redundancy. The first one is very powerful, because it does not require an increase of time and bandwidth, and it is often employed together with the space-time coding.

The spatial multiplexing can be combined with the precoding, in presence of channel knowledge, or with diversity coding when a better decoding reliability is desired. Moreover, the spatial multiplexing makes the receiver very complex, and in general it is combined with the Orthogonal Frequency-Division Multiplexing (OFDM)[3] or with the Orthogonal Frequency-Division Multiple Access

(OFDMA)[4], where the multi-path is is well managed. [1] [2] [3] [4]

2.2 MIMO system model

Considering a Rayleigh fading channel and a complex–valued MIMO system with M_t transmit and M_r receive antennas, the channel is represented by the $M_r \times M_t$ matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,M_t} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,M_t} \\ \vdots & \vdots & \ddots & \vdots \\ h_{M_r,1} & h_{M_r,2} & \cdots & h_{M_r,M_t} \end{bmatrix} \tag{2.1}$$

where $h_{ij} \sim N_c(0,1)$ is the fading coefficient between the j-th transmitter and the i-th receiver The M_r-dimensional received signal $\mathbf{y}$ is given by:

$$\mathbf{y} = \mathbf{H}\mathbf{s}_t + \mathbf{n} \tag{2.2}$$

where $\mathbf{n} \sim N_c(0, N_0)$ denotes the M_r-dimensional additive i.i.d. (independent and identically distributed) white Gaussian noise vector, and $\mathbf{s}_t$ stands for the M_t-dimensional transmit signal vector, $\mathbf{s}_t = [s_1 s_2 \dots s_{M_t}]^T$. Entries of $\mathbf{s}_t$ are independently chosen from a complex constellation $\mathcal{O}$, whose cardinality is $|\mathcal{O}| = 2^Q$, with Q equal to the number of bits per symbol. The transmission rate is defined as $R = M_t Q$ bits per channel use (bpcu).

For QAM modulations, according to [3], the M_t-dimensional complex signal model in (2.2) can be decomposed into an $M = 2M_t$-dimensional real signal

[1] It consists in controlling the direction and the sensitiveness of the signal, in presence of multiple transmitters and receivers. At the receiver side, the beamforming increases the sensitiveness in the direction of the signal and reduces that in the direction of the noise. At the transmitter side, instead, it increases the power of the signal to be emitted.

[2] It indicates the effects of propagation of an electromagnetic wave, due to reflection, diffraction and scattering of signals, which reach the receiver as a sum of copies of the same.

[3] It is a multi-carrier modulation scheme, which makes use of a great number of orthogonal sub-carriers. Each of these sub-carriers is modulated with a conventional modulation scheme, such as amplitude or quadrature amplitude modulation, at low symbol rate. The total transmission rate is comparable with that of a single-carrier modulation schemes, keeping the bandwidth constant. The signals are generated and detected with the Fast Fourier Transform.

[4] It is a multi-user version of the OFDM. The multiple access is assured granting a subset of sub-carriers for each user.

model:

$$\begin{bmatrix} \Re\{\mathbf{y}\} \\ \Im\{\mathbf{y}\} \end{bmatrix} = \begin{bmatrix} \Re\{\mathbf{H}\} & -\Im\{\mathbf{H}\} \\ \Im\{\mathbf{H}\} & \Re\{\mathbf{H}\} \end{bmatrix} \begin{bmatrix} \Re\{\mathbf{s}_t\} \\ \Im\{\mathbf{s}_t\} \end{bmatrix} + \begin{bmatrix} \Re\{\mathbf{n}\} \\ \Im\{\mathbf{n}\} \end{bmatrix} \tag{2.3}$$

where $\Re\{*\}$ and $\Im\{*\}$ respectively denote real and imaginary parts of $*$. In the real–valued model, the set of valid real–valued symbols is indicated as Ω, where $|\Omega| = 2^{Q/2}$. In the rest of the thesis, a real–valued model is assumed.

2.3 MIMO detection algorithms

Assuming perfect channel knowledge at the receiver and considering systems with space-time coding and spatial multiplexing, the algorithms to separate the data streams, exploited in literature, can be divided into four main categories:

1. **Linear Detection Methods**: they simply invert the channel matrix. In this family, there are:
   - **Zero Forcing (ZF)**: it multiplies the inverse of the channel matrix by the received signal vector. It does not take into account the noise, and for this reason, it has not good results in terms of performance.
   - **Minimum Mean Square Error (MMSE)**: it minimizes the square error between the received and the transmitted vectors, taking into account the noise and operating on each component, separately. This method is not able to reach the maximum diversity order $M_r M_t$, but it is limited to $M_r - M_t + 1$. This causes a low Bit Error Rate (BER) at high SNRs.
2. **Interference Cancellation method (IC)**: it first detects the stronger stream, among all received signal vectors, by means of ZF or MMSE, and removes it before detecting the next stronger streams. Two main techniques are exploited:
   - **SUccessive Cancellation (SUC)**: it randomly chooses a component of the received vector and assumes the others as interference. It removes the contribution of it from the vector: this procedure is repeated for each symbol in the vector. This method reaches a diversity order, which is comparable with that of MMSE.

- **Ordered SUccessive Cancellation (OSUC)**: similarly as SUC, it chooses the component with the highest Signal to Interference Noise Ratio (SINR). It achieves an higher diversity order than SUC, but it has limited performance, because ZF or MMSE is employed for the initial choice.

3. **Maximum Likelihood (ML) detector**: it minimizes the difference between the received signal vector and the vector of transmitted symbols distorted by the channel. The problem of deriving the ML estimation $\tilde{\mathbf{s}}$ for the transmitted signal vector is formally stated as

$$\tilde{\mathbf{s}} = \arg \min_{\mathbf{s} \in \Omega^{M_t}} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2 \tag{2.4}$$

where $\mathcal{O}^{M_t}$ is the set of possible values for transmitted symbol vectors $\mathbf{s}$.This method is optimal, since it minimizes the error probability and reaches the maximum diversity order, but a direct implementation of ML detection is the exhaustive search that explicitly computes (2.4) for all candidate symbols $\mathbf{s} \in \mathcal{O}^{M_t}$. The detector has to examine 2^R hypothesis for each received symbol vector. If for low rates, such as $R \leq 8bpcu$, this is feasible, for large M_t and constellations it is practically unfeasible. As an example, for a 4×4 with a 16QAM modulation, corresponding to $R \leq 16bpcu$, the number of candidates to be examined is 65536.

4. **Sphere Decoder Algorithm (SDA)**: it has been introduced as a viable alternative to the ML detector, able to guarantee optimal performance, but only with a polynomial complexity [29]. It consists in limiting the search to only those points, which lie inside an hypersphere, with a certain radius, around the received vector. A detailed description of SDA is presented in the following paragraph.

2.4 Sphere Decoder Algorithm

SDA achieves ML performance, showing a polynomial average complexity [30]. It reduces the number of candidate symbols to be considered, without excluding the ML solution.

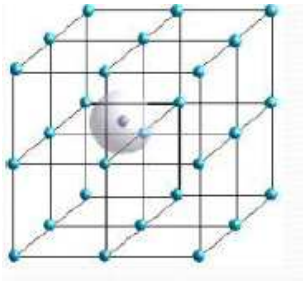


Figure 2.1: Graphical example of an hypersphere

Anyway, the term Sphere Decoder was born to indicate a family of algorithms: one differs from each other, essentially for the way to visit the tree. The following paragraph presents a complete description of a depth–first Schnorr–Euchner SDA, which is the starting point of this research. For completeness, the section 2.5 reports other versions of the SDA, presented in literature.
The Sphere Decoder Algorithm is composed of three main steps: *radius constraint*, *tree construction* and *tree exploration*.

- **Radius Constraint (RC)**
  The calculation of (2.4) is formulated as a tree visit problem, constrained to only those **Hs** points that lie inside a hypersphere with radius r, around the received point **y**. This means that:

$$\|\mathbf{y} - \mathbf{Hs}\|^2 \leq C \tag{2.5}$$

  where $C = r^2$.

- **Tree construction**
  Before computing the distance between received and transmitted vectors, **H** can be triangularized using proper techniques, such as Cholesky decomposition: $\mathbf{H} = \mathbf{QR}$, where the $M \times M$ matrix **Q** has orthonormal columns ($\mathbf{Q^H Q} = \mathbf{I}_M$) and the $M \times M$ matrix **R** is upper triangular. (2.4) can be written as:

$$\tilde{\mathbf{s}} = \arg \min_{\mathbf{s} \in \Omega^M} \|\mathbf{y} - \mathbf{QRs}\|^2 = \arg \min_{\mathbf{s} \in \Omega^M} \|\tilde{\mathbf{y}} - \mathbf{Rs}\|^2 \tag{2.6}$$

where $\tilde{\mathbf{y}} = \mathbf{Q}^T\mathbf{y}$ is the Zero–Forcing (ZF) solution [1] [1]. Thus, (2.5) becomes:

$$\|\tilde{\mathbf{y}} - \mathbf{R}\mathbf{s}\|^2 \leq C \tag{2.7}$$

A tree can be constructed [1], where each node has $|\Omega|$ sons and there are $M = 2M_t$ levels, in a real–valued model. As an example, a 4×4 MIMO system with 16 QAM modulation has $M = 8$ levels and $|\Omega| = 4$ sons ($\Omega = \{-3, -1, 1, 3\}$ is a real–valued constellation). At the tree root, the possible values of entry s_M of vector $\mathbf{s}$ are associated to $|\Omega|$ sons, while tree leaves correspond to all possible vectors $\mathbf{s}$. A node placed at intermediate tree level l will corresponds to a partial symbol vector $[s_l \; s_{l+1} \ldots s_M]^T$.

The square distance $d^2(s) = \|\tilde{\mathbf{y}} - \mathbf{R}\mathbf{s}\|^2$ can be iteratively computed following the tree from the root to a leaf:

$$T_l = \begin{cases} 0 & l = M+1 \\ T_{l+1} + |\tilde{y}_l - \sum_{j=l}^{M} R_{l,j}s_j|^2 & l = M, \ldots, 1 \end{cases} \tag{2.8}$$

where l is the current level of the tree, T_l is the (squared) *Partial Euclidean Distance* (PED), calculated as the squared distance between entries M to l of the received vector and partial symbol vector $[s_l \; s_{l+1} \ldots s_M]^T$ (distorted by the channel). Therefore $d^2(s) = T_1$ is the (squared) *Euclidean Distance* (ED). As example, figure 2.2 shows the obtained tree for a 2×2-MIMO system and a QPSK modulation.

At each level, if obtained PED is larger than the specified radius, the corresponding node and the whole underlying sub–tree are excluded from the search (tree pruning). The amount $e_l = |\tilde{y}_l - \sum_{j=l}^{M} R_{l,j}s_j|$ is called *Distance Increment* (DI) and can be written as:

$$e_l = |\tilde{y}_l - \sum_{j=l+1}^{M} R_{l,j}s_j - R_{l,l}s_l| \tag{2.9}$$

Symbol $\psi[l]$ is introduced to indicate the first two terms in (2.9), which represents the l-th received signal reduced by the interference resulting from

[1] Alternatively to this method, MMSE preprocessing [31] might be used on an extended channel matrix for the Cholesky decomposition.

the already estimated symbols, s_M to s_{l+1}:

$$\psi[l] = \tilde{y}_l - \sum_{j=l+1}^{M} R_{l,j} s_j \tag{2.10}$$

The expression above is unfolded as

$$\psi[l] = |\,(\cdots((\tilde{y}_l - R_{l,M}s_M) - R_{l,M-1}s_{M-1}) - \cdots - R_{l,l+1}s_{l+1})\,| \tag{2.11}$$

The computation in (2.11) can be arranged in $M - l + 1$ recursive steps:

$$\begin{array}{rll} 1) & \psi^{M+1}[l] & = \tilde{y}_l \\ 2) & \psi^{M}[l] & = \tilde{y}_l - R_{l,M}s_M = \psi^{M+1}[l] - R_{l,M}s_M \\ 3) & \psi^{M-1}[l] & = \tilde{y}_l - R_{l,M}s_M - R_{l,M-1}s_{M-1} = \psi^{M}[l] - R_{l,M-1}s_{M-1} \\ \cdots & & \\ M-l+1) & \psi^{l+1}[l] & = \psi^{l+2}[l] - R_{l,l+1}s_{l+1} = \psi[l] \end{array} \tag{2.12}$$

where the upper index of ψ ranges between $M + 1$ and $l + 1$. At the last recursion step, $\psi^{l+1}[l]$ is used to compute DI:

$$e_l = |\tilde{y}_l - \sum_{j=l+1}^{M} R_{l,j}s_j - R_{l,l}s_l| = |\psi^{l+1}[l] - R_{l,l}s_l| \tag{2.13}$$

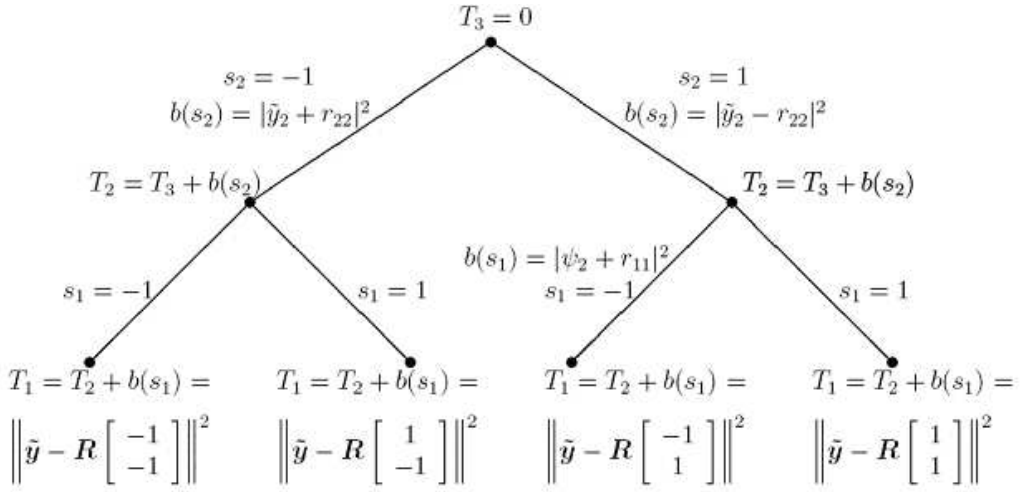


Figure 2.2: Tree for a 2×2-MIMO system and a QPSK modulation.

- **Tree exploration**

  The tree exploration in SDA is depth–first: when a tree node lies inside the hypersphere, the algorithm goes one level down (*forward processing*). When a tree leaf is reached and when part of the tree is pruned, an alternative node has to be selected going back to an upper level in the tree: this part of the algorithm is known as *backward processing* [1]. In SDA, forward processing

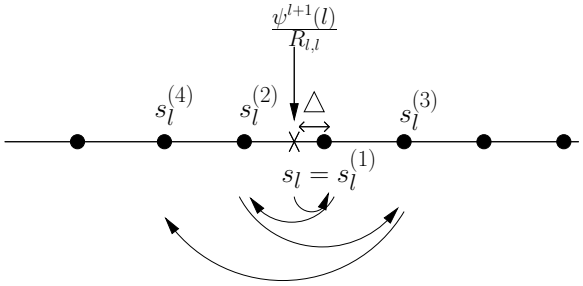


Figure 2.3: Schnorr–Euchner enumeration around Babai Point.

proceeds along tree levels: starting from the tree root ($l = M$), the nearest point to the received signal (Babai Point) [14] can be calculated by means of a rounded division, as reported in [3] [1] [32]:

$$s_l = s_l^{(1)} = \left\lfloor 0.5 \frac{\psi^{l+1}[l]}{R_{l,l}} + 1 \right\rfloor \tag{2.14}$$

where $\lfloor 0.5x + 1 \rfloor$ rounds argument x to the nearest odd integer value (this is equivalent to select the nearest point in a PAM constellation). In the rest of the thesis, the Babai Point $s_l^{(1)}$ is indicated as s_l for simplicity. Metric normalization with respect to $R_{l,l}$ elements along the diagonal of the channel matrix can significantly simplify this operation. However additional computational effort will be required in this case for normalization, especially in the presence of fast fading channel. Alternatively the Babai point can also be determined by calculating:

$$s_l = s_l^{(1)} = arg \min_s |\psi^{l+1}[l] - R_{l,l}s| \tag{2.15}$$

In this work, (2.14) is adopted.

ψ and PED metrics are then updated according to (2.12) and (2.8). Forward processing continues at the lower tree level until either a tree leaf is reached

or the current PED metric goes outside the hypersphere ($T_l > C$). In these cases, backward processing is started to return back to the closer tree level where not all sibling nodes have already been explored and to select a new symbol. In the Schnorr–Euchner (SE) enumeration, the first chosen symbol at each level is always the Babai Point, while following choices at the same tree level are made according to a zig–zag path around it; this enumeration technique follows the PED ascending order. Fig. 2.3 shows the points selected at tree level l, starting from the Babai Point s_l and following the zig–zag sequence for subsequent choices, $s_l^{(2)}$, $s_l^{(3)}$, $s_l^{(4)}$. When a tree leaf is reached, radius may be updated if $T_1 < C$: if this is the case, T_1 becomes the new radius for the subsequent tree exploration. The tree leaf with the minimum T_1 represents the ML solution.

2.5 Other versions of Sphere Decoder Algorithm

The different versions of SDA can be classified, based on three aspects:

1. **System model: complex–valued versus real–valued**
   The main difference between these two categories is the tree construction and the average number of visited nodes, directly related to the throughput. In fact, in a real–valued system, the tree has a double number of levels than in a complex–valued one, and a number of sons, which is equal to the root of the cardinality of the corresponding complex–valued constellation. Considering the previous example, shown in the section 2.4, of a 4×4 MIMO system with 16QAM modulation, the tree has $M_t = 4$ levels and $|\mathcal{O}| = 16$ sons ($\mathcal{O}$ is a complex–valued constellation), in a complex-valued model. In a real–valued one, instead, it has $2M_t = 8$ levels and $\sqrt{|\mathcal{O}|} = 4$ sons. Clearly, the number of leaves is the same for both choices.
   The actual number of tree nodes that are visited is directly related to the throughput, which may also be affected by the use of a real or complex–valued channel model. While several authors proved that complex–valued model results in a lower number of visited nodes with respect to real–valued one [1], it was shown in [33] [34] that the latter offers two advantages: (i) it involves a lower global number of elementary real–valued operations and

(ii) allows for simple enumeration techniques. For this reason real–valued model is more suitable for practical implementation.

2. **Direction of the tree exploration: breath–first versus depth–first** While the depth–first strategy (see the tree exploration in 2.4) first chooses one symbol at each level and goes down, coming back only later for examining the other alternative points at that same level, the breath–first concurrently selects more than one point at each level. The main exponent of this family is the **K-best**: it expands the K more promising points at each level. A big value of K avoids a great loss of BER, but it requires to expand an high number of paths in parallel, employing a huge amount of hardware resources. On the other hand, a small K causes a BER degradation. The algorithm finishes when it reaches the leaf level, and the ML solution is represented by the leaf with the minimum *ED*, among all K leaves. In this way, the *K-best*, and, more in general the breath–first strategy, is characterized by a fixed number of visited nodes, and therefore a fixed throughput [35]. In the depth–first, instead, the actual number of tree nodes that are visited in the detection of a received symbol vector is a stochastic value and its average depends on signal to noise ratio. This implies that detection throughput is not constant.

3. **Enumeration: Fincke Pohst (FP) versus Schnorr-Euchner (SE)** The FP method is the fist introduced from an historical point of view [36]. It consists in considering the nodes of a level with the *natural* order. For example, in the case of a 4PAM, where sons are $\{-3,-1,1,3\}$, the nodes are visited exactly with that order. Only later, a more efficient method has been introduced, such as the SE strategy ((see the tree exploration in 2.4)).

Most algorithms are a combination of this three aspects. A short description of the main important follows:

Statistical pruning : it is a modified version of the SDA, in particular, the pruning is based on statistical criteria. It establishes a schedule of the radius $r_1 \leq r_2 \leq \ldots \leq r_{M_t}$ and solves $T_1 \leq r_1^2$,$T_1+T_2 \leq r_2^2$, ...,$T_1+T_2+\ldots+T_{M_t} \leq r_{M_t}^2$. Since the radius is different at each level of the tree, the region $\mathcal{D}$, which limits the search, is not an hypersphere. This means that the detection is

not exactly ML, but the performance can arbitrarily be close to ML, thanks to the choice of the probability ε that the transmitted vector does not belong to $\mathcal{D}$, because the error probability upper bound is $P_e \leq P_e^{ML} + \varepsilon$. This solution achieves a complexity reduction of order M_t^2 [37].

Automatic Sphere Decoder (ASD) : it is a very efficient algorithm for a software implementation. The nodes are stored into an ordered list, based on their PEDs. The expanded node is the first element in the list, i.e. the node with the smallest PED. The peculiarity of this method is that the above node can be at each level of the tree, but since it requires a sorting and a visiting of a list, it is not feasible for an hardware implementation.

Fixed-complexity Sphere Decoder (FSD) : it is a combination of K-best and SDA. Independently of the noise level and the initial radius, it only visits a fixed number of nodes for each level, based on the χ^2 distribution of the noise matrix. At the uppermost level many candidates are considered, because of the inter-level interference. At the lowest levels, instead, the number is reduced. This produces a performance-complexity trade-off: bigger the number of candidates, better the performance, but higher the computational complexity. This makes FSD suitable for reconfigurable architectures, able to adaptively change the number of candidates, according to channel conditions [38].

Conditioned Ordered Successive Interference Cancellation (COSIC) : it is a particular case of FSD. The idea is to simplify the search thanks to successive interference cancellation for all nodes at the uppermost level. The throughput can be constant or not, depending on the implementation, and it has slightly worse performance with respect to SDA, but it reduces the computational complexity and occupied silicon area [39].

Two techniques, which can be easily applied to all previous algorithms and all variants of SDA in order to reduce the processing complexity, are the Early Termination (ET) and the simplification of the mathematic norm in (2.4).
The Early Termination consists in constraining the maximum number of visited nodes. When this limit is reached, the detection ends, returning the best found

solution, which is not guaranteed to be ML. It can have a deep BER degradation. A variant of this method is to change the constraint, according to channel conditions [40].
Another aspect of the SDA, which can be used in any methods, is the possibility of employing a simplified norm. This allows to reduce the complexity, from an implementation and algorithmic points of view, at a price of a degradation of BER.
The radius constraint can be rewritten as:

$$d_2(s) = \sqrt{T_1^2 + T_2^2 + \dots T_{M_t}^2} \leq r \tag{2.16}$$

where T_i represents the PED for the i-th level of the tree. It is possible to approximate the Euclidean-norm with one-norm:

$$d_2(s) \approx d_1(s) = |T_1| + |T_2| + \dots + |T_{M_t}| \tag{2.17}$$

or with the infinity-norm:

$$d_2(s) \approx d_\infty(s) = \max\{|T_1|, |T_2|, \dots, |T_{M_t}|\} \tag{2.18}$$

3

Look-ahead Sphere Decoder Algorithm

3.1 State of the Art

A relatively small number of SDA architectures have been implemented with real ML performance (e.g. the 1^{st} implementation in [1] and [3]), mainly because of three limitations associated with the ML approach: the high computational complexity, resulting in a large occupied area, the variable number of visited tree nodes, and the limited average throughput.
Some works are focused on the reduction of the implementation area: in [1] the proposed ASIC (Application–Specific Integrated Circuit) adopts a approximated norm (see section 2.5), which allows for small area; in [39] further techniques are explored to even reduce Silicon area; in [41], area cost is limited resorting to efficient searching. The second problem, the variable number of visited nodes, has been widely investigated: in [42], a breath–first algorithm is implemented to achieve a fixed throughput, independently of noise level; the concept of layered orthogonal lattice detector is introduced in [43] to guarantee constant throughput; the parallel architecture proposed in [44] achieves a small variance in the number of visited nodes, resulting in matching run–time constraints; in [45] the number of visited nodes is limited by means of a new search strategy. High throughput is also the aim of the ASIC II implementation in [1] that offers a throughput up to 169 Mbps at 20 dB on a 4×4 16QAM MIMO system, while the parallel and pipelined VLSI architecture in [17] also guarantees a fixed decoding throughput

up to 53 Mbps for the same system. Some recent VLSI implementations for MIMO detectors with reduced complexity are also available in [46] and [47]. All the works mentioned above have the common characteristic of showing a decline in performance with respect to the optimal receiver.
While most of these solutions achieve performance that is close to ML, this research work faces the problem of a high throughput full ML implementation.

Some formal transformations can be applied to processing algorithms to enable high throughput implementation. In this work, look–ahead, pipelining and retiming transformations [48] are employed in order to shorten critical path delay and to increase throughput with respect to SDA. These techniques assume a graphical representation of the initial algorithm, in particular the well-known Data-Flow-Graph representation (DFG) is chosen [48].

3.2 Look-ahead methodology

This section gives an overview of formal transformation procedures that can be applied to processing algorithms to enable high throughput implementation. In order to clarify the employed methodology, a simple example is proposed: an IIR (Infinite Impulse Response) filter. Eq. (3.1) describes the filter and Fig. 3.1 gives the corresponding DFG representation:

$$y[k] = x[k] + ay[k-1] \tag{3.1}$$

where node with symbol + represents the addition and the node with the symbol X the multiplication; a is a constant coefficient, $x[n]$ and $y[n]$ input and output samples at time n. The edge going from addition to multiplication is associated to a inter–iteration data dependency and for this reason it contains one unit-delay (register), labeled with D; on the contrary, the other edge going from multiplication to addition has no delay, as it represents an intra–iteration data dependency. With the architecture of Fig. 3.1 the critical path delay is given by the following equation:

$$t_{cp} = T_{add} + T_{mul} \tag{3.2}$$

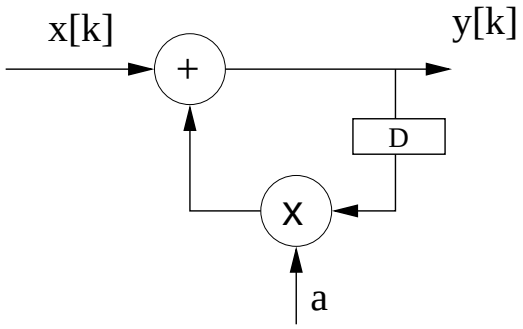


Figure 3.1: DFG representation of the computation $y[k] = x[k] + ay[k-1]$.

The critical path delay determines the maximum clock frequency, achievable by that architecture. In fact, f_{max} is exactly the inverse of t_{cp}:

$$f_{max} = \frac{1}{t_{cp}} \tag{3.3}$$

Therefore also the processing throughput, which is directly proportional to the clock frequency, is superior bounded to:

$$th <= \frac{1}{T_{add} + T_{mul}} \tag{3.4}$$

In the *Look–ahead* transformation a linear recursion is iterated few times to create additional concurrency. Applying look-ahead to the iteration (3.1), the filter equation becomes:

$$y[k] = x[k] + ay[k-1] = x[k] + ax[k-1] + a^2 y[k-2] \tag{3.5}$$

where two loop steps are concatenated. Fig. 3.2 shows the DFG, obtained after look-ahead transformation. Now, the number of resources is higher with respect to the original IIR and also the critical path is increased:

$$t_{cp}^{look} = 2T_{add} + T_{mul} \tag{3.6}$$

But what is important is the presence of two registers in the recursive part of the graph, allowing to apply retiming technique. It consists in changing the location of delay elements in the datapath, without affecting the input/output behavior of the circuit. Moreover, it is also possible to apply pipelining to the

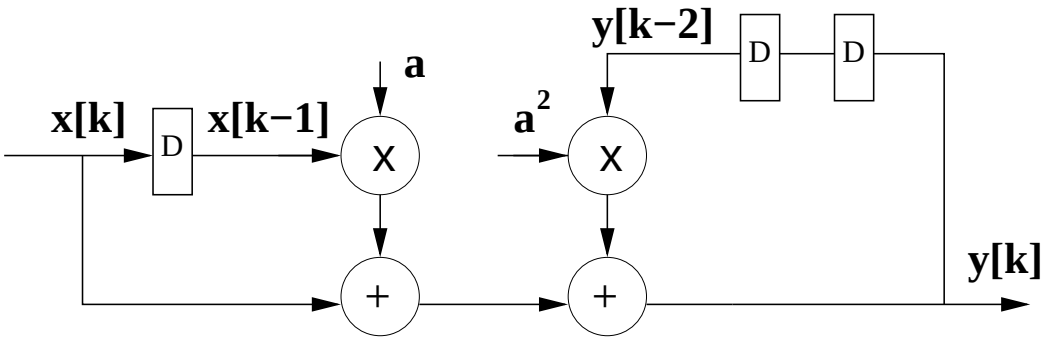


Figure 3.2: The two steps look-ahead DFG of computation $y[k] = x[k] + ay[k-1]$.

non-recursive part of the DFG, introducing registers along the datapath to reduce the critical path delay. Applying pipelining and retiming to the DFG in Fig. 3.2, the new DFG in Fig. 3.3 is obtained, where unit-delays shown as D_p are pipelining registers and unit-delays labeled with D_r are retiming registers. The critical path

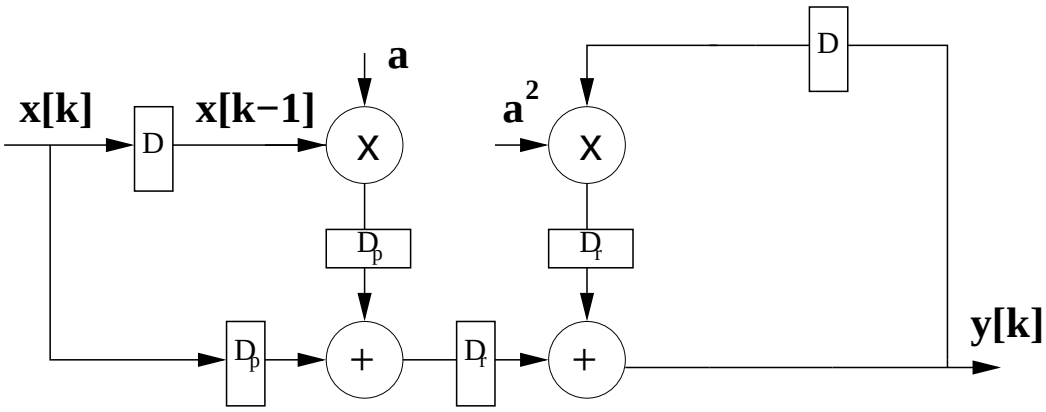


Figure 3.3: The two steps look-ahead DFG with pipelining and retiming of computation $y[k] = x[k] + ay[k-1]$.

delay is significantly reduced and the achievable throughput becomes:

$$th_{max} = \frac{1}{max(T_{add}, T_{mul})} \tag{3.7}$$

In general, look-ahead transformation, accompanied by pipelining and retiming, provides an increment of throughput, and an increase of complexity. The most important limitation of look-ahead transformation is that it is feasible only for linear loops.

The presented methodology is applied to the SDA algorithm, described in section 2.4, as detailed in the following section.

3.3 Look–ahead optimization of SDA

3.3.1 DFG representation

Starting from equations (2.8) to (2.14), the obtained DFG is represented in Fig. 3.4 for the forward processing of SDA.

To make more readable the DFG in Fig. 3.4, main operations are grouped in blocks: the S block implements (2.14), obtaining the Babai Point s_l from the received $\psi^{l+1}[l]$. DI block implements (2.13): the ED e_l is generated from symbol s_l and $\psi^{l+1}[l]$. Block PED in the DFG implements (2.8) to update PED metrics. Operations performed within each block are detailed in dotted boxes.

On the left side of the DFG, computational blocks for the processing of previously defined ψ amounts are indicated. As shown in (2.12), the computation of $\psi^{l+1}[l]$ (i.e. the received symbol reduced by interference resulting from the already estimated symbols) can be distributed along $M-l+1$ steps: at step $k=1$, $\psi^{M+1}[l]=\tilde{y}_l$ is calculated, at step $k=2$, $\psi^{M}[l]$ is obtained from $\psi^{M+1}[l]$ by subtracting the $R_{l,M}s_M$ contribution, and at each subsequent step k a new $\psi^{M+2-k}[l]$ is generated. In particular, step $k=1$ only needs the ZF term $\tilde{y}_l$ and can be performed for all l before starting tree exploration. Step $k=2$ makes use of the first estimated symbol s_M: as a consequence, all $\psi^{M}[l]$ can be calculated when exploration is at top of the tree (level M). Similarly, the following steps $k=3,\ldots,M-l+1$ are executed at tree levels $M-1,\ldots,l+1$.

The DFG in Fig. 3.4 shows the forward processing at a specific tree level (l). $\psi^{l+1}[l]$, calculated at the previous level $l+1$, is used to compute e_l according to (2.13). At the same time, $\psi^{l+1}[l-1]$, also computed at level $l+1$, is necessary to generate $\psi^{l}[l-1]$, which will be used at the lower level to obtain e_{l-1}:

$$\psi^{l}[l-1]=\psi^{l+1}[l-1]-R_{l-1,l}s_l \tag{3.8}$$

A ψ-type block consisting of one multiplier, one adder and one register is needed to execute (3.8). Additional ψ-type blocks are necessary to concurrently generate

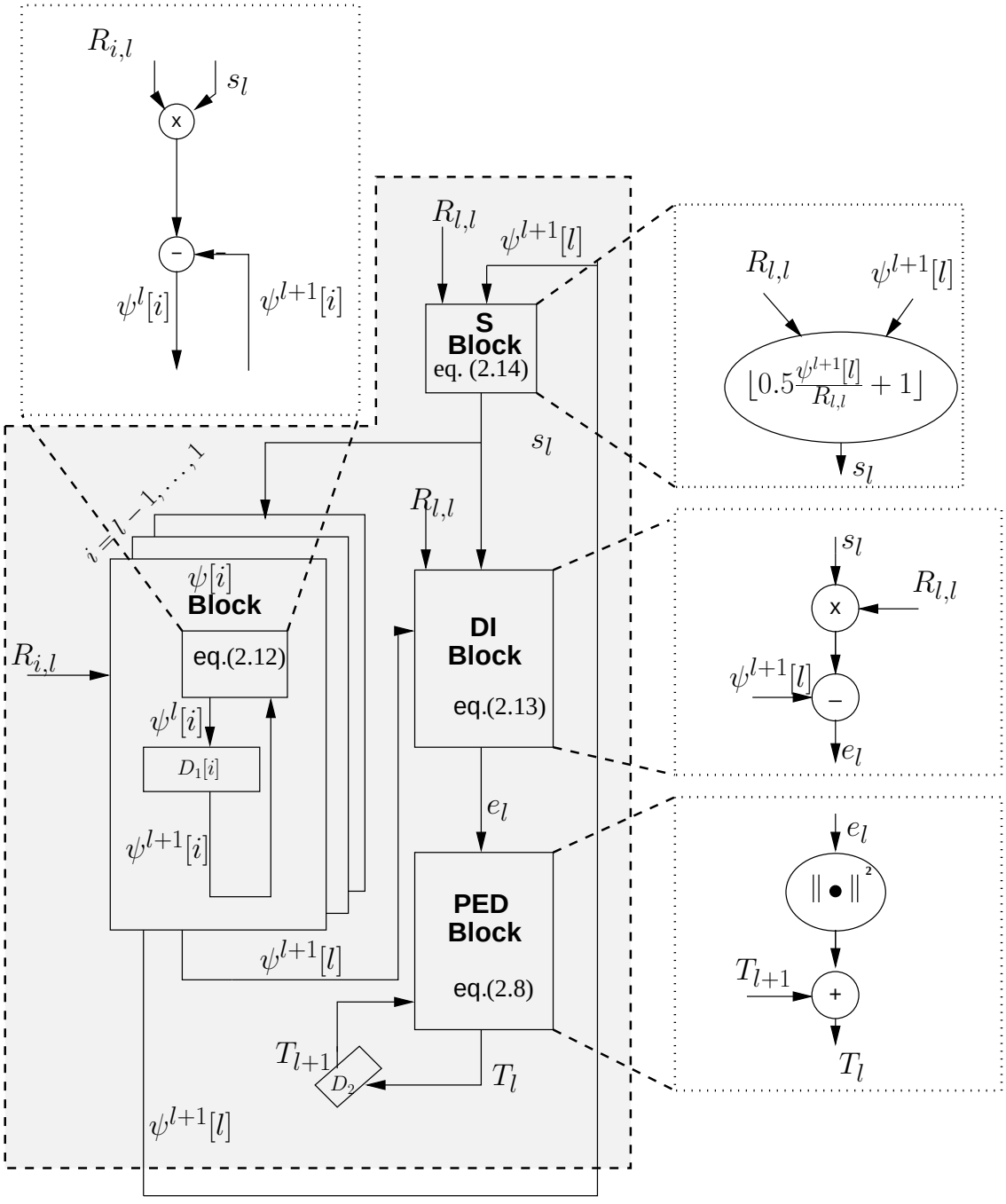


Figure 3.4: The DFG representation of SDA (Forward processing).

the $\psi^l[i]$ amounts that will be required at the lower tree levels, $i < l-1$. Therefore

Table 3.1: Combinational delay ($t_{pd_{min}}$) of computational blocks in forward processing. Rows 2 to 5 refer to processing blocks required for SDA; row 6 refers to the additional component needed to support LASDA

Block	$t_{pd_{min}}$[ns]
S block	2.05
$\psi[i]$ block	1.31
DI block	1.31
PED block	1.48
$\widehat{\psi[i]}$ block	1.65

$l-1$ ψ-type blocks are indicated in the DFG (Fig. 3.4), each one performing the following update operation:

$$\psi^l[i] = \psi^{l+1}[i] - R_{i,l}s_l \quad i = l-1, \ldots 1 \tag{3.9}$$

$\psi[i]$ blocks also include $l-1$ registers, $D_1[i]$, with $i = l-1, \cdots, 1$, used to accumulate $\psi^l[i]$ amounts. These registers are initially loaded with the corresponding $\tilde{y}_i$. In the adopted notation for $\psi^l[i]$, the l index corresponds to the current tree level and therefore during *forward processing* it decreases by one at each cycle. As an example, input of register $D_1[i]$ is indicated as $\psi^l[i]$, while the output from the same register is $\psi^{l+1}[i]$.

Now, the next useful step in this analysis is to identify where the critical path is located, along the forward processing. In particular, it is interesting to find out the delay of each block, in order to understand which is the slowest. A VHDL description of each one has been developed and processing delays have been derived from synthesis using Synopsys Design Compiler version Z-2007.03-SP1. A 0.13 μm CMOS Standard Cell technology is adopted (Table 3.3, rows 2 to 5). The first column of Table 3.3 shows the name of the block and the second column reports the minimum delay obtained with a 0.13 μm technology. According to synthesis results, the critical path delay in the DFG is approximately [1] given by:

$$t_{c.p.}^{SDA} \cong t_{S\ block} + t_{\ DI\ block} + t_{\ PED\ block} = 4.84\text{ns} \tag{3.10}$$

[1] With automated synthesis, when multiple components are integrated, the whole critical path delay may be different from the simple sum of component delays.

where $t_{S\ block}$, $t_{\ DI\ block}$ and $t_{\ PED\ block}$ are the delays associated respectively to blocks S, DI and PED. The delay along the identified critical path can be easily

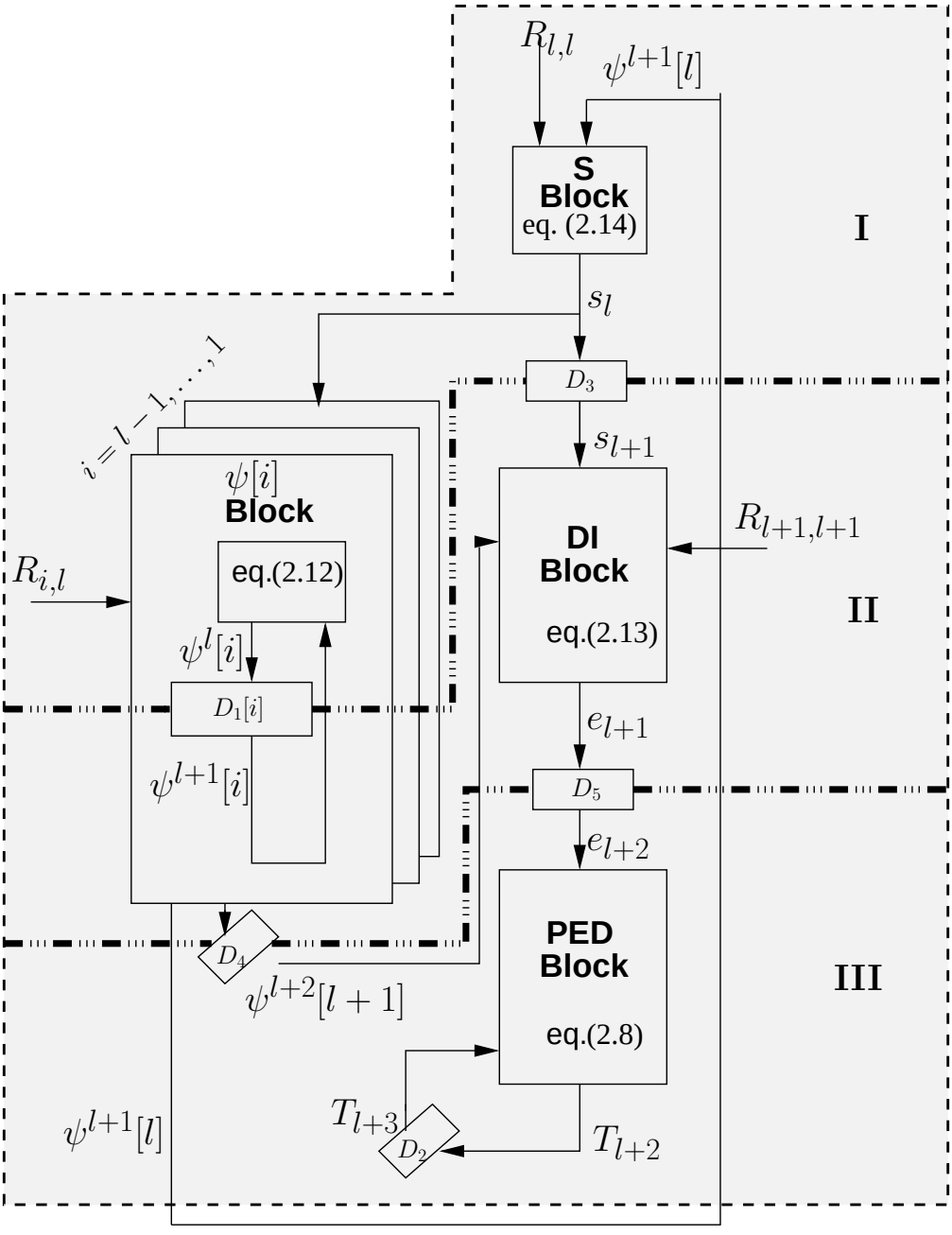


Figure 3.5: The DFG representation of SDA (Forward processing) after pipelining. Registers D_3, D_4, D_5 have been inserted.

reduced by inserting pipelining registers, because blocks S, DI and PED are not part of a loop: the pipelined DFG is shown in Fig. 3.5, where registers D_3, D_4, D_5 have been inserted. In the new DFG, the critical path is now placed along the loop including *S block* and $\psi[i]$ blocks ($loop_1 = S\ block \rightarrow \psi[i]\ block \rightarrow S\ block$): in this case the insertion of pipeline registers along the loop is not allowed [48]. The other loops in the DFG, namely $loop_2 = PED\ block \rightarrow PED\ block$ and $loop_3 = \psi[i]\ block \rightarrow \psi[i]\ block$, are not critical, as they include faster operations than $loop_1$. Therefore $loop_1$ is the actual throughput bottleneck of the SDA.

$$t_{c.p.}^{pipe\,SDA} \cong t_{S\ block} + t_{\psi\ block} = 3.36\text{ns} \tag{3.11}$$

3.3.2 Linear approximation and look–ahead transformation

One effective technique to improve processing throughput along a critical loop is look–ahead [48]. However the look–ahead technique can only be applied to linear processing, while $loop_1$ includes the non-linear operation ($\lfloor 0.5x + 1 \rfloor$). Thus a linear approximation is necessary. In the forward processing, according to (2.12) and (2.14), the calculation of $\psi^{l+1}[l]$ at level $l+1$ can be written as

$$\psi^{l+1}[l] = \psi^{l+2}[l] - R_{l,l+1}s_{l+1} = \psi^{l+2}[l] - R_{l,l+1}\left\lfloor 0.5\left(\frac{\psi^{l+2}[l+1]}{R_{l+1,l+1}}\right) + 1\right\rfloor \tag{3.12}$$

Round operation in (3.12) is now replaced with a simple division:

$$\psi^{l+1}[l] \cong \widehat{\psi}^{l+1}[l] = \psi^{l+2}[l] - R_{l,l+1}\frac{\psi^{l+2}[l+1]}{R_{l+1,l+1}} \tag{3.13}$$

The symbol $\widehat{\psi}$ is used to represent the approximate version of ψ. Amounts $\psi^{l+2}[l]$ and $\psi^{l+2}[l+1]$ in (3.12) are expressed according to (2.12) as:

$$\begin{aligned}\psi^{l+2}[l] &= \psi^{l+3}[l] - R_{l,l+2}s_{l+2}\\ \psi^{l+2}[l+1] &= \psi^{l+3}[l+1] - R_{l+1,l+2}s_{l+2}\end{aligned}$$

Eq. (3.14) are substituted in (3.13), so obtaining:

$$\widehat{\psi}^{l+1}[l] = \psi^{l+3}[l] - \frac{R_{l,l+1}}{R_{l+1,l+1}}\psi^{l+3}[l+1] - s_{l+2}\left[R_{l,l+2} - \frac{R_{l,l+1}}{R_{l+1,l+1}}R_{l+1,l+2}\right] \tag{3.14}$$

With the following pre-calculated coefficients,

$$A_l = \frac{R_{l,l+1}}{R_{l+1,l+1}} \quad B_l = R_{l,l+2} - \frac{R_{l,l+1}}{R_{l+1,l+1}} R_{l+1,l+2} \tag{3.15}$$

(3.14) becomes:

$$\widehat{\psi}^{l+1}[l] = \psi^{l+3}[l] - A_l \psi^{l+3}[l+1] - B_l s_{l+2} \tag{3.16}$$

This is the key result that allows for breaking the SDA throughput bottleneck. While in the first part of (3.12) the calculation of $\psi^{l+1}[l]$ needs s_{l+1}, (3.16) clearly shows that the metric $\widehat{\psi}^{l+1}[l]$ does not depend on s_{l+1} any more. Instead it requires s_{l+2}, which is available one cycle in advance with respect to s_{l+1}.

Since the computation of $\widehat{\psi}^{l+1}[l]$ does not need to wait for s_{l+1}, we can think to recursively used it as an input of S block, in order to accelerate the processing in the $loop_1$. This approach has the advantage of reducing the critical path. But, as a consequence, the generated output symbol might be different from the Babai Point and it must be indicated in a different way: $\widehat{s}_l$. (3.16) has then to be written in the form

$$\widehat{\psi}^{l+1}[l] = \psi^{l+3}[l] - A_l \psi^{l+3}[l+1] - B_l \widehat{s}_{l+2} \tag{3.17}$$

The effects of the possible difference, due to the substitution of the Babai Point s_l with $\widehat{s}_l$, are discussed later.

Eq. (3.17) represents now the recursion of the look–ahead SDA and a new DFG can be drawn for the modified SDA, as shown in Fig. 3.6. It is obtained from Fig. 3.5 by simply inserting the additional operations resulting from the look–ahead transformation: the $\widehat{\psi}$ block implements (3.17), while registers D_4 to D_9 introduce required delays. All other blocks perform the same operations as in Fig. 3.5. Three of the updated ψ are now directly used: $\psi^{l+2}[l+1]$ (output of register D_4) is still used by DI block, $\psi^{l+3}[l]$ and $\psi^{l+3}[l+1]$, two cycles delayed versions of $\psi^{l+1}[l]$ and $\psi^{l+1}[l+1]$ (through registers D_4, D_6, D_7 and D_8), feed the new $\widehat{\psi}$ block, which computes $\widehat{\psi}^{l+1}[l]$, according to eq. (3.17). Register D_9 is also required to generate $\widehat{s}_{l+2}$ from $\widehat{s}_{l+1}$.

As an effect of the applied look–ahead transformation, the outer feedback loop in Fig. 3.6 (S *block* $\rightarrow$ $\psi[i]$ *block* $\rightarrow$ $\widehat{\psi}$ *block* $\rightarrow$ S *block*) includes additional registers (D_4, D_6, D_7 and D_8) that can be properly shifted to cut the combinational

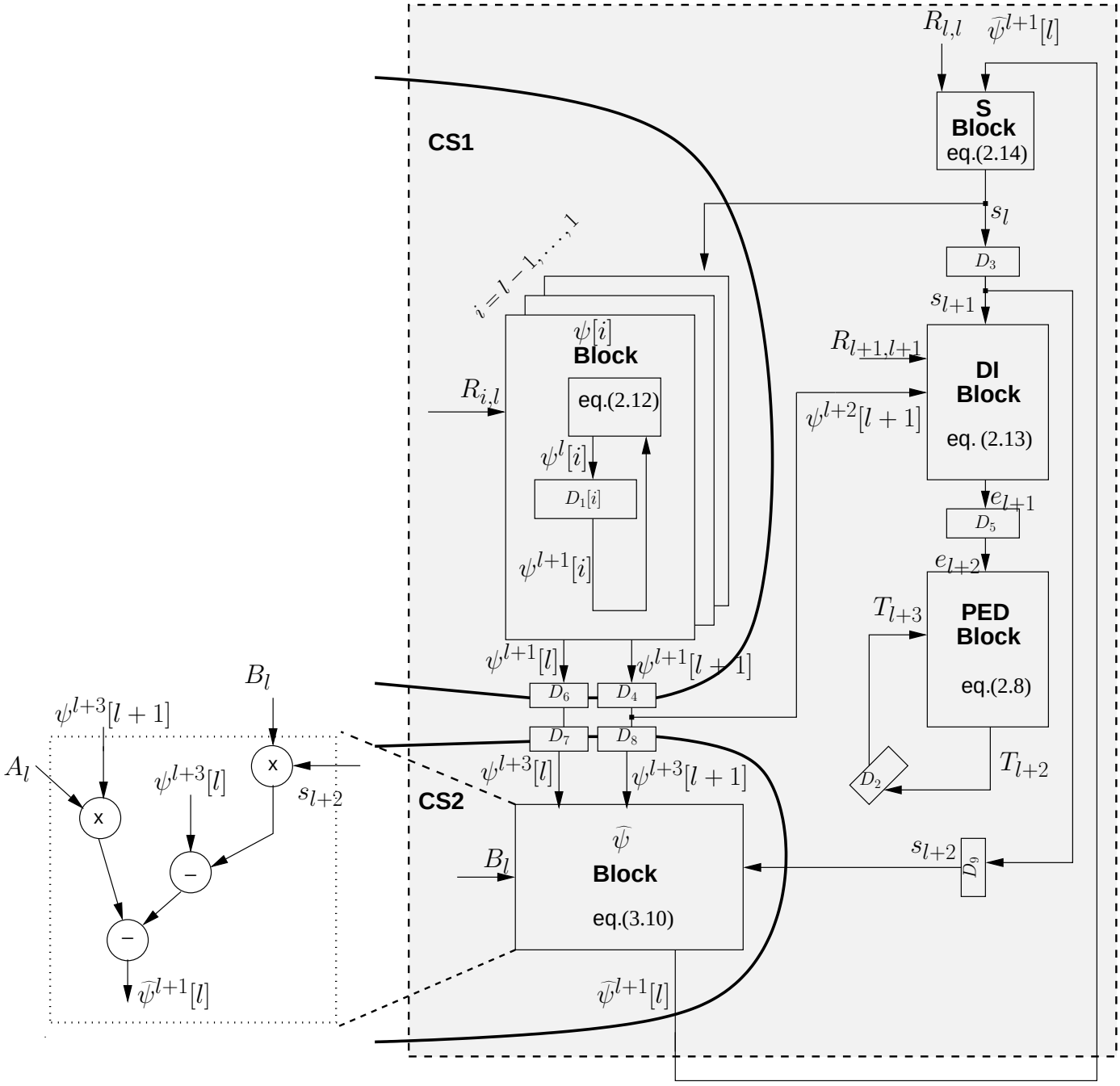


Figure 3.6: DFG representation of LASDA (Forward processing) after look–ahead. Retiming is applied on cut-sets CS_1 and CS_2.

delay (retiming). The key requirement is to split the critical path formed by S and $\psi[i]$ blocks in Fig. 3.6. In order to increase the throughput, the cut–set retiming technique [48] can be exploited to move registers in suitable positions. The one cycle delay introduced by registers D_4 and D_6 can be moved from the outputs to the input of $\psi[i]$ block by removing the two registers and feeding $\psi[i]$ block with signal $\widehat{s}_{l+1}$, which is already available at register D_3 (cut-set indicated as CS_1 in Fig. 3.6). Moreover delay introduced by registers D_7, D_8 and D_9 can be moved from the inputs to the output of $\widehat{\psi}$ block by removing D_7, D_8 and D_9 and inserting the new register D_{10} in Fig. 3.7. The $\psi[i]$ blocks now receive symbol $\widehat{s}_{l+1}$ (output of register D_3) instead of $\widehat{s}_l$: as a consequence, all signals in $\psi[i]$ blocks are shifted by one cycle (cut-set indicated as CS_2 in Fig. 3.6). Thus outputs $\psi^{l+2}[l]$ and $\psi^{l+2}[l+1]$ are directly applied to the input of $\widehat{\psi}$ and DI blocks, with no need of registers. We call this modified algorithm LASDA (Look–ahead SDA).

The $loop_1$ in the DFG of Fig. 3.5 includes S and $\psi[i]$ blocks and it is distributed along two clock cycles. The path composed by S, DI and PED blocks, instead, is distributed along three clocks, as shown in Fig. 3.5. The outer loop in the LASDA DFG of Fig. 3.7 contains three blocks (S, $\psi[i]$ and $\widehat{\psi}$), but the whole processing is distributed along three clock cycles, as indicated by bold dashed lines: this leads to a potentially large processing speed-up with respect to the SDA. In order to obtain an estimation for this speed-up, the $\widehat{\psi}$ block has been described in VHDL and synthesized with the same approach adopted for the other main blocks (see Table 3.3, last row): resulting delay is 1.65 ns. As in the new DFG, registers are present at the input and output of each processing block, the critical path corresponds to the longest block delay, which is equal to 2.05 ns (S block). The critical path delay of the LASDA DFG, $t_{c.p.}^{LASDA}$, is then shorter than $t_{c.p.}^{SDA}$.

$$t_{c.p.}^{LASDA} \cong t_{S\ block} = 2.05\text{ns} < t_{c.p.}^{SDA} = 4.84\text{ns} \tag{3.18}$$

If compared with the worst case delay estimated for SDA, this result shows a 58% improvement in terms of clock frequency, as demonstrated in (3.18).

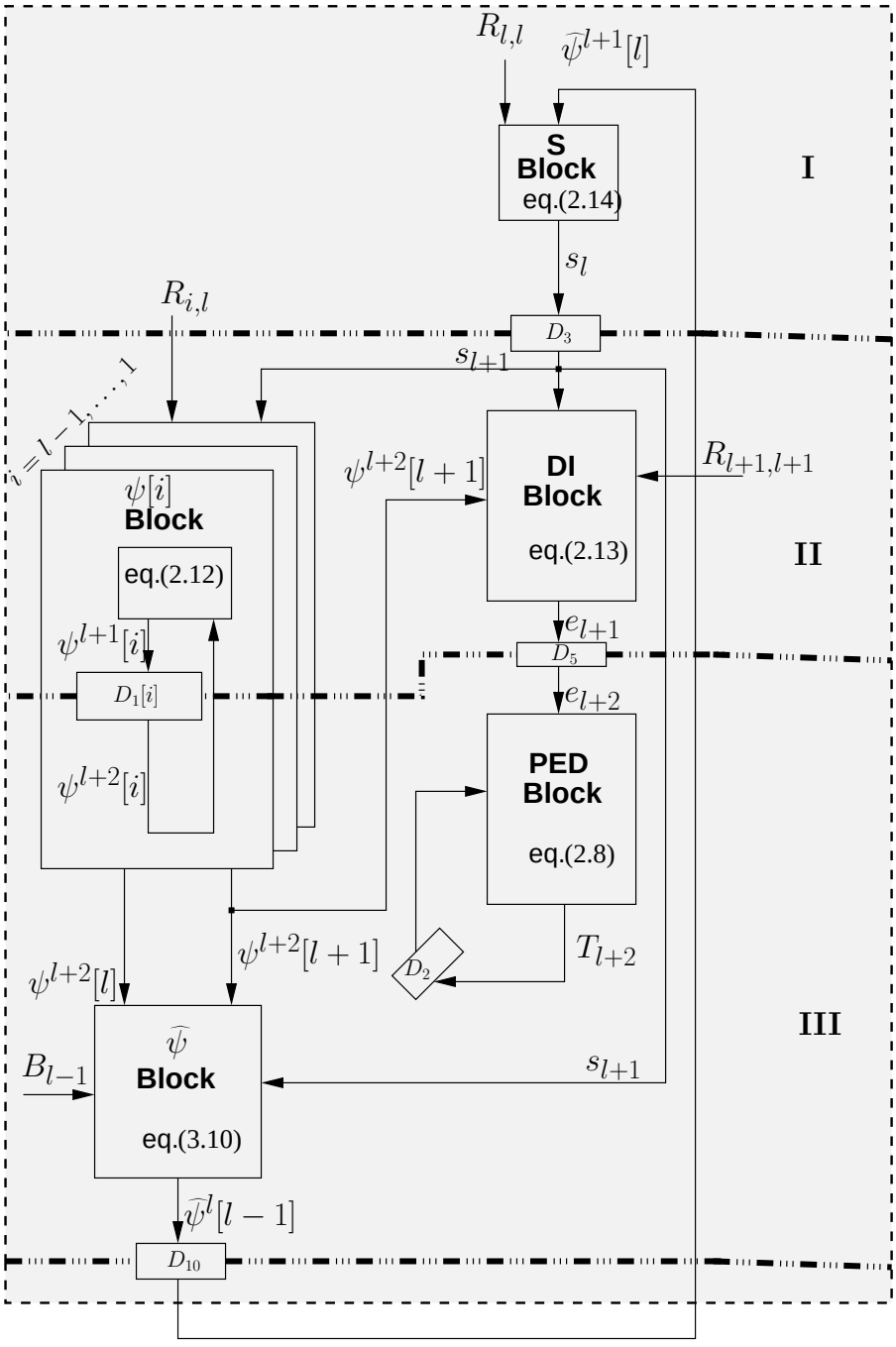


Figure 3.7: DFG representation of LASDA (Forward processing) after look–ahead, retiming and pipelining.

3.3.3 Performance evaluation of LASDA

LASDA and SDA have been compared in terms of BER performance and average number of clock cycles (which, together with clock frequency, determines the average throughput). Fig. 3.8 shows the general scheme of transmitter and receiver,

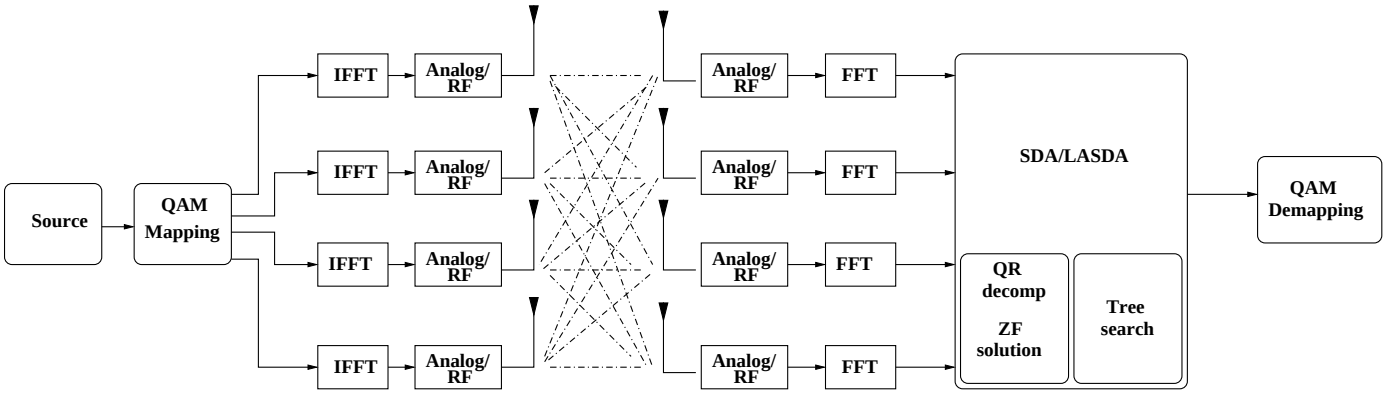


Figure 3.8: Scheme of the transmitter and the receiver.

which is described as a cycle accurate model in C language. A random source of bits is directly mapped onto a complex–valued 16 QAM constellation, using Gray mapping. A 4×4 MIMO system is assumed, so that the bit stream is divided in blocks of 16 bits (4 bits per symbols per 4 transmitters). A flat i.i.d. Rayleigh block-fading channel is assumed: the channel coefficients remain constant over a block of 16 bits. Eq. (2.2) described the MIMO system. At the receiver side, perfect CSI is assumed and unsorted QR decomposition is performed. As detector, either SDA and LASDA are employed. For both the tree search is depth–first real-valued with SE enumeration. The 16QAM is transformed into a real–valued 4 PAM. As a consequence, the tree has $M = 8$ levels, with 4 sons per node. The detector makes an hard decision on symbols, which are at last directly demapped onto bits. In the detector, a fixed-point precision of 16 bits has been chosen, because it was adopted in most of the literature for the 16-QAM modulation, so it represents a good choice for comparisons. In particular, 8 bits are adopted both for integer and fractional part [49]. The signal-to-noise ratio(SNR) is defined as $SNR = M_T E_s / N_0$, with $E_s = \mathrm{E}[|s|^2]$, $s \in \mathcal{O}$. The software model of LASDA detector is detailed in Algorithm 2, which is derived modifying the SDA, reported in Algorithm 1.

Algorithm 1 SDA

Require: $|\Omega|$ number of symbols in the real-valued constellation, M number of antennas, r initial radius, $\tilde{y}_i$ $(i = 1, ..., M)$ ZF solution

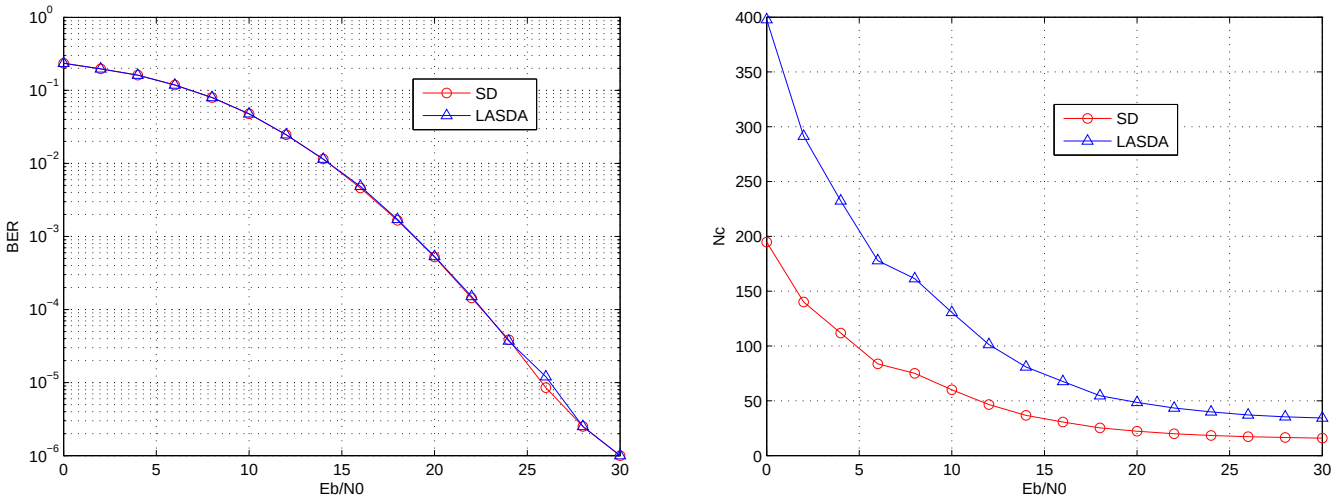
1: $l \leftarrow M$, $k_{max} \leftarrow |\Omega|$, $C \leftarrow r^2 = \infty$
2: *Forward processing*:
3: compute Babai Point, $s_l^{(1)}$, set $k_l = 1$ and $s_l = s_l^{(1)}$
4: update PD, $\psi^l(i)$ $(i = l - 1, \dots 1)$
5: update PED, T_l
6: **if** $T_l < C$ and $l > 1$ **then**
7: $\quad l \leftarrow l - 1$
8: $\quad$ **goto** 2
9: **end if**
10: *Backward processing*:
11: **if** $(l = 1)$ and $(T_l < C)$ **then**
12: $\quad C \leftarrow T_1$
13: **end if**
14: **if** $(k_l < k_{max})$ and $(T_l < C)$ **then**
15: $\quad k_l \leftarrow k_l + 1$
16: $\quad$ find $s_l^{(k_l)}$ (next best symbol choice) and set $s_l = s_l^{(k_l)}$
17: $\quad$ **goto** 4
18: **end if**
19: **if** $l < M$ **then**
20: $\quad l \leftarrow l + 1$
21: $\quad$ **goto** 14
22: **end if**

Algorithm 2 LASDA

Require: $|\Omega|$ number of symbols in the real-valued constellation, M number of antennas, r initial radius, $\tilde{y}_i$ $(i = 1, ..., M)$ ZF solution

1: $l \leftarrow M$, $k_{max} \leftarrow |\Omega|$, $C \leftarrow r^2 = \infty$, $\widehat{\psi}^{M+1}(M) \leftarrow \tilde{y}_M$
2: *Forward processing*:
3: compute $\widehat{s}_l$, first symbol choice from $\widehat{\psi}^{l+1}(l)$, and set $k_l = 1$, $s_l = \widehat{s}_l$
4: update PD, $\psi^l(i)$ $(i = l - 1, ...1)$
5: update PED, T_l
6: compute $\widehat{\psi}^l(l-1)$
7: **if** $T_l < C$ and $l > 1$ **then**
8: $\quad l \leftarrow l - 1$
9: $\quad$ **goto** 2
10: **end if**
11: *Backward processing*:
12: **if** $(l = 1)$ and $(T_l < C)$ **then**
13: $\quad C \leftarrow T_1$
14: **end if**
15: **if** $(k_l < k_{max})$ and $(T_l < C)$ **then**
16: $\quad k_l \leftarrow k_l + 1$
17: $\quad$ find $s_l^{(k_l)}$ (next best symbol choice) and set $s_l = s_l^{(k_l)}$
18: $\quad$ **goto** 12
19: **end if**
20: **if** $l < M$ **then**
21: $\quad l \leftarrow l + 1$
22: $\quad$ **goto** 23
23: **end if**

In the reported simulations, the SNR (Signal-to-Noise-Ratio) ranges between 0 and 30 dB. Two measures are derived from the C model, BER and N_C, average number of clock cycles needed to detect one symbol vector (four 16QAM signals for the considered case). In particular, N_C is derived, considering a *one-node-per-cycle* architecture, i.e. one clock is counted for each visited node. No error correction code is used in the simulated transmission scheme.



(a) (b)

Figure 3.9: BER (a) and iterations (IT)(b) of SDA and LASDA.

Simulation results obtained for SDA and LASDA are reported in Fig. 3.9. No degradation in terms of BER can be seen in Fig. 3.9(a): LASDA and SDA BER curves are exactly superimposed. ML performance of LASDA is explained by the fact that applied formal transformations do not modify the algorithm behaviour but simply its execution on a hardware architecture. Approximate $\widehat{\psi}$ is only used by the S block in Fig. 3.7: although different from the Babai Point, the computed symbol always is a valid constellation point. Moreover, once a valid symbol is generated, it is used to evaluate PED metrics with no further approximations. Since the PED calculated at a certain node is always coherent with the selected symbol,no erroneous pruning can be decided and no decline in the BER performance is caused by LASDA. On the other hand, an huge increase in the number of clock cycles is shown in Fig. 3.9(b) for LASDA over SDA. This

increase is around 118% at SNR= 20 dB (BER $5.2 \cdot 10^{-4}$). This is due to the fact that, during forward processing, LASDA evaluates approximate $\widehat{\psi}$, which is used to make the first symbol choice at a tree level l, $\widehat{s}_l$. This computation provides a processing speed-up over SDA. However the linear approximation introduced in LASDA might cause changes in the node enumeration. In particular, $\widehat{s}_l$ might be different from the true Babai Point, s_l, and this cause the increase in the average number of visited nodes. Next subsection presents the solution to this problem.

3.3.4 A modified search strategy: Test & Restart

The key idea is to allocate resources to calculate Babai Point s_l and to compare it against $\widehat{s}_l$. The calculation of the Babai Point requires a separate S block and it is carried out one clock cycle later than calculation of $\widehat{s}_l$, while LASDA is already processing the lower tree level: for this reason, this late Babai Point cannot be efficiently used to drive the tree search in the forward phase. However it can be used to detect and stop a wrong visit direction. Two symbols are then concurrently computed at each clock cycle: $\widehat{s}_l$, which is rapidly derived from $\widehat{\psi}$ metric in one single clock cycle, as illustrated in Fig. 3.7, and s_l, the Babai Point, which is derived one cycle later from $\psi^{l+1}[l]$, by means of a separate S block. The two symbols are exploited to control tree exploration with a strategy said *Test & Restart*. This strategy involves the following three steps:

1. Symbol s_l is compared to $\widehat{s}_l$.
2. $s_l = \widehat{s}_l$ means that the symbol chosen by LASDA is the Babai Point. In this case, the SE enumeration can be based on the computed $\widehat{s}_l$ and the tree visiting continues with the usual depth–first SE strategy.
3. $s_l \neq \widehat{s}_l$ means that the symbol selected by LASDA is not the Babai Point. This decision is not the best one and it is dropped: $\widehat{s}_l$ is substituted with s_l and the forward processing is started again with one cycle of delay. Also SE enumeration will start from s_l.

Fig. 3.10 reports the flux diagram of LASDA with *Test & Restart* strategy. It is worth clarifying here that step 3 does not lead to any tree pruning, which only occurs after obtaining a PED greater than current C. When $s_l \neq \widehat{s}_l$, the

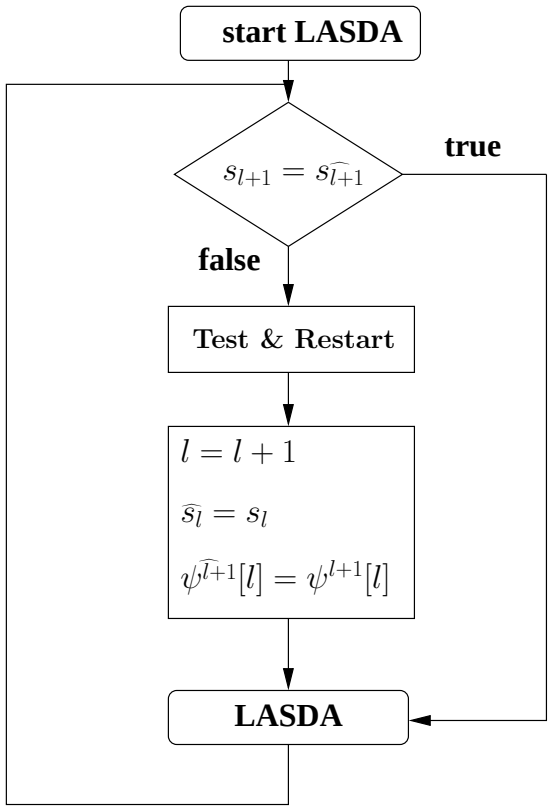


Figure 3.10: Flux diagram of the LASDA with *Test & Restart* strategy.

LASDA choice is discarded and replaced by the Babai Point; the tree exploration is then continued accordingly. However the dropped search direction could be taken again in a later visit to the same tree level.

3.3.5 Performance evaluation of LASDA with Test & Restart

Fig. 3.11 reports an example of tree search with the *Test&Restart* strategy: at levels l and $l-1$, the LASDA is supposed to select symbols $\widehat{s_l} = 1$ and $\widehat{s_l} = 3$ in cycles 1 and 2 respectively: this search direction is indicated with a bold dotted line. In cycle 2, symbol $s_l = -1$ is calculated: as this value is different from $\widehat{s_l}$, the current search direction is stopped and postponed to a later time. The new direction corresponding to $s_l = -1$ is taken and continued in cycle 3 with a new LASDA choice (bold solid line). The example shows that one additional

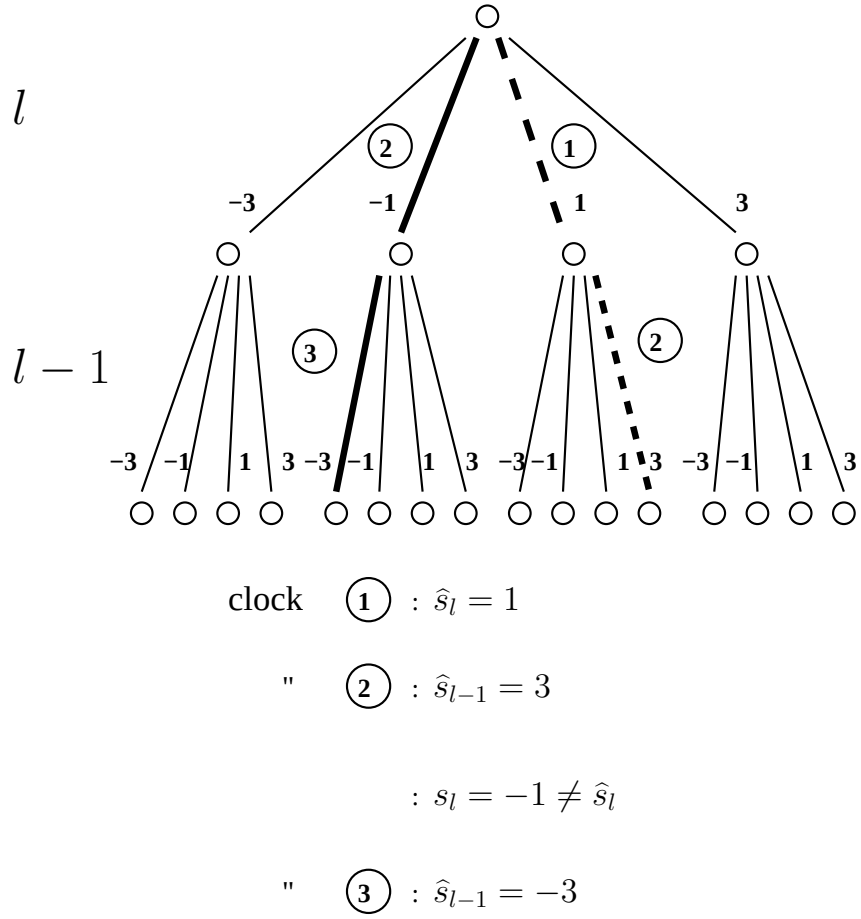
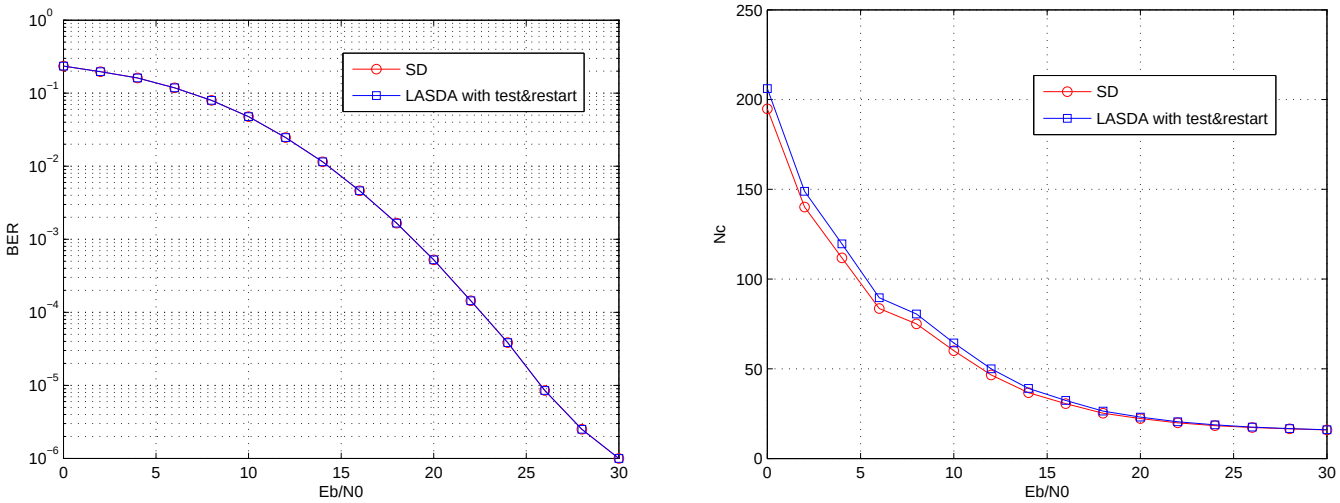


Figure 3.11: Example of the tree search with the *Test & Restart* strategy.

clock cycle is necessary when the test on $\widehat{s}_l$ fails. Thanks to the Test & Restart search strategy, LASDA and SDA visit the same sequence of tree nodes; the only difference between the two algorithms is that LASDA tries to anticipate the next node in the forward processing and returns to the best choice when a wrong direction is taken. As a consequence, ML performance is already achieved, as without the Test & Restart, as shown in Fig. 3.12-(a). But in the new search strategy, some nodes might be visited twice as an effect of failed tests; as a consequence it is expected that the average number of clock cycles needed for the



(a) (b)

Figure 3.12: BER (a) and iterations (IT)(b) of SDA and LASDA with *Test & Restart*.

detection of a received symbol vector is increased with respect to SDA, but also reduced with respect to LASDA without Test & Restart. In fact, as shown in Fig. 3.12, the described approach results into a very small number of additional clock cycles: this increase is around 4% at SNR= 20 dB (BER $5.2 \cdot 10^{-4}$). The final LASDA throughput depends on both the average number of required clock cycles and the achievable clock frequency.

The software model of LASDA with Test & Restart is detailed in Algorithm 3.

Algorithm 3 LASDA with Test & Restart

Require: $|\Omega|$ number of symbols in the real-valued constellation, M number of antennas, r initial radius, $\tilde{y}_i$ $(i = 1, ..., M)$ ZF solution

1: $l \leftarrow M$, $k_{max} \leftarrow |\Omega|$, $C \leftarrow r^2 = \infty$, $\widehat{\psi}^{M+1}(M) \leftarrow \tilde{y}_M$
2: *Forward processing*:
3: compute $\widehat{s}_l$, first symbol choice from $\widehat{\psi}^{l+1}(l)$, and set $k_l = 1$, $s_l = \widehat{s}_l$
4: **if** $l < M$ **then**
5: *Test & Restart*:
6: compute Babai Point at previous tree level, $s_{l+1}^{(1)}$, from $\psi^{l+2}(l+1)$
7: **if** $\widehat{s_{l+1}} \neq s_{l+1}^{(1)}$ **then**
8: $l \leftarrow l + 1$
9: $s_l \leftarrow s_l^{(1)}$
10: $\widehat{\psi}^{l+1}(l) \leftarrow \psi^{l+1}(l)$
11: **end if**
12: **end if**
13: update PD, $\psi^l(i)$ $(i = l - 1, ...1)$
14: update PED, T_l
15: compute $\widehat{\psi}^l(l-1)$
16: **if** $T_l < C$ and $l > 1$ **then**
17: $l \leftarrow l - 1$
18: **goto** 2
19: **end if**
20: *Backward processing*:
21: **if** $(l = 1)$ and $(T_l < C)$ **then**
22: $C \leftarrow T_1$
23: **end if**
24: **if** $(k_l < k_{max})$ and $(T_l < C)$ **then**
25: $k_l \leftarrow k_l + 1$
26: find $s_l^{(k_l)}$ (next best symbol choice) and set $s_l = s_l^{(k_l)}$
27: **goto** 12
28: **end if**
29: **if** $l < M$ **then**
30: $l \leftarrow l + 1$
31: **goto** 23
32: **end if**

3.4 Architecture design

The S block is a key component in the proposed LASDA architecture. The use of this processing block in an hardware sphere decoder was already introduced in [3]. The behaviour of the component is described in the following sub-section.

3.4.1 S block

Before seeing the overall high level architecture, a particular attention is given to S block, which is certainly the most critical block, in terms of time. Functions of remaining units are clearly defined and close to HW implementation: they process signals using adders, subtractors and multipliers. Less clear, instead, is the direct implementation of this unit.
It receives the approximate version of the PD metric and selects the Babai Point, according to eq. (2.14); additionally it derives the direction $step_l$ of the first successive element in the Schnorr-Euchner enumeration. The processing functions of S block unit is formally defined as:

$$\begin{cases} \widehat{s_l} = \left\lfloor 0.5\frac{\widehat{\psi}^{l+1}[l]}{R_{l,l}} + 1 \right\rfloor \\ \widehat{step_l} = sign\left(\triangle\right) = sign\left(\widehat{s_l} - \frac{\widehat{\psi}^{l+1}[l]}{R_{l,l}}\right) \end{cases} \tag{3.19}$$

Fig. 2.3 shows the sequence of selected symbols at a given tree level around the first choice. At the first visit to a level, the Babai-Point corresponds to the first symbol to be chosen ($\widehat{s_l}^{(1)}$); the $\widehat{step_l}$ information is set to 0 if the algorithm will have to consider the left symbol ($\widehat{s_l}^{(2)}$) at the following visit of the node, and to 1 if the next choice is on the right ($\widehat{s_l}^{(3)}$). The S block simply choses the starting direction of the SE enumeration, in the forward processing. At each new visit of the node, i.e in the backward processing, the direction will be simply switched, so following the Schnorr-Euchner enumeration.
The general purpose hardware implementation of numerical division is not straightforward and it usually leads to expensive components. However in this specific case, a general hardware division can be avoided and an ad-hoc solution, customized on 16QAM case, can be adopted in order to limit hardware cost and achieve high throughput. The result returned by the S block is a signal point.

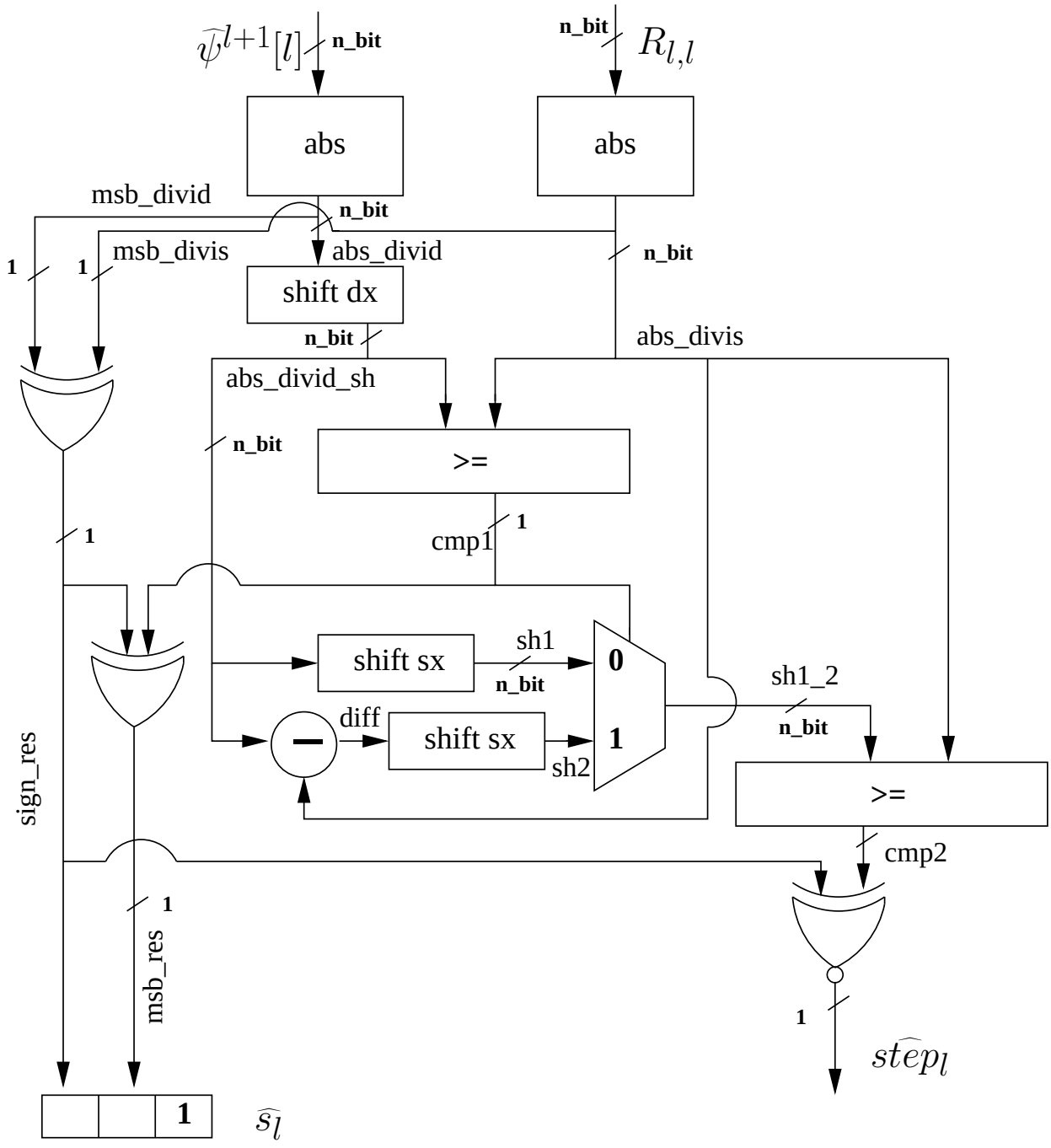


Figure 3.13: Scheme of the S block.

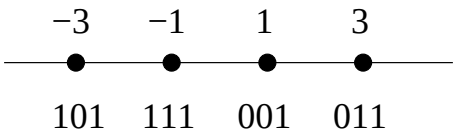


Figure 3.14: Points of a 4-PAM constellation.

The 2' complement representation of the points of a 4-PAM constellation is given in Fig. 3.14 as an example. It is worth noticing that:

1. the Least Significant Bit (LSB) is equal to 1 for all points
2. the Most Significant Bit (MSB) is the sign.

Thus only the second bit must be determined [3]. Fig. 3.15 shows the implementation architecture for S block. It is composed of elementary units: left and right shifters, a 2-to-1 demultiplexer, comparators, xor and xnor gates, and blocks which generate the absolute value.

3.4.2 High level architecture

Fig. 3.15 shows the implementation architecture for the whole LASDA based detector. The datapath architecture includes three fundamental components:

- Forward Unit (FU), which is the core of the detector and performs the forward visit of the tree. This unit, detailed in Fig. 3.7, includes S, $\psi[i]$, $\widehat{\psi}$, DI and PED blocks described in Section 3.3. Additionally multiplexers are used to enable replacement of $\widehat{s}_l$ with the symbols provided by either backward processing in SE enumeration ($s_l^{(k)}$ with $k > 1$) or Test & Restart (late Babai point s_l). It is clear that if $\widehat{s}_l$ is substituted with s_l, also corresponding $\widehat{step_l}$ must be replaced by $step_l$, computed by the additional S block in the Test & Restart Unit, and sent to the Backward Unit. The additional multiplexer, devoted to this aim, is not indicated only to not complicate Fig. 3.15. This is the same also for $\widehat{\psi}^{l+1}[l]$: the register at the output of the $\widehat{\psi}$ block should be anticipated by another 2-to-1 multiplexer, where the alternative input is $\psi^{l+1}[l]$, stored into the Ψ Mem.
- Backward Unit (BU), which concurrently selects the subsequent node to be expanded, according to SE enumeration, and updates the corresponding ψ

metrics:

$$\psi^l[i] = \psi^{l+1}[i] - R_{i,l}s_l^k \;\; i = l-1, \ldots 1 \;\; k = 2, 3, \ldots \tag{3.20}$$

In order to ensure that a new node is expanded at each clock cycle, a new, alternative metric must be available also after a pruning operation has taken place. Thus, when a given node p is reached, two candidate nodes are concurrently computed: the first one is a direct son of p and it is processed by the FU; the second one is the first sibling node of p according to the SE enumeration and it is concurrently computed by the BU. Both units generate ψ terms (indicated as $\underline{\psi}_{FU}$ and $\underline{\psi}_{BU}$ in Fig. 3.15). $\psi[i]$ blocks in FU and BU share the same architecture. S block in FU and Alt block in BU also have similar structures, however the latter does not need to perform the division included in the S block: the following best choices for s_l ($s_l^{(k)}$, with $k > 1$) are generated according to the SE policy as shown in Fig. 2.3 and described in [3].

- Test & Restart Unit (TRU), which implements the modified tree visit strategy described in Section 3.3. TRU computes the Babai Point s_l by means of an additional S block and compares it (CMP block) with the symbol $\widehat{s}_l$, provided by the FU unit. The CMP block is a simple comparator.

The architecture is completed by three memories for the storage of channel coefficients (R Mem), ψ amounts (Ψ Mem) and PED metrics (PED Mem). R memory stores the upper triangular matrix $\mathbf{R}$, obtained after Cholesky decomposition (2.6). $1/2M \times M$ locations are required in the Ψ memory, due to the triangular structure of the $\mathbf{R}$ matrix; however an $M \times M$ memory is allocated to simplify the generation of addresses. The PED memory only has M locations, so it can be implemented as a set of registers rather than using a RAM generator.

A control unit (CU) properly synchronizes the processing components. It is implemented as a Finite State Machine (FSM). The processing flow is basically affected by two control operations. The first one is related to tree pruning and backward processing. PED metrics are sent to the control unit to verify radius constraint (2.7) and signal f/b is used to control the first multiplexer in FU: the alternative symbol $s_l^{(k)}$ is selected if the test fails, while depth–first search is

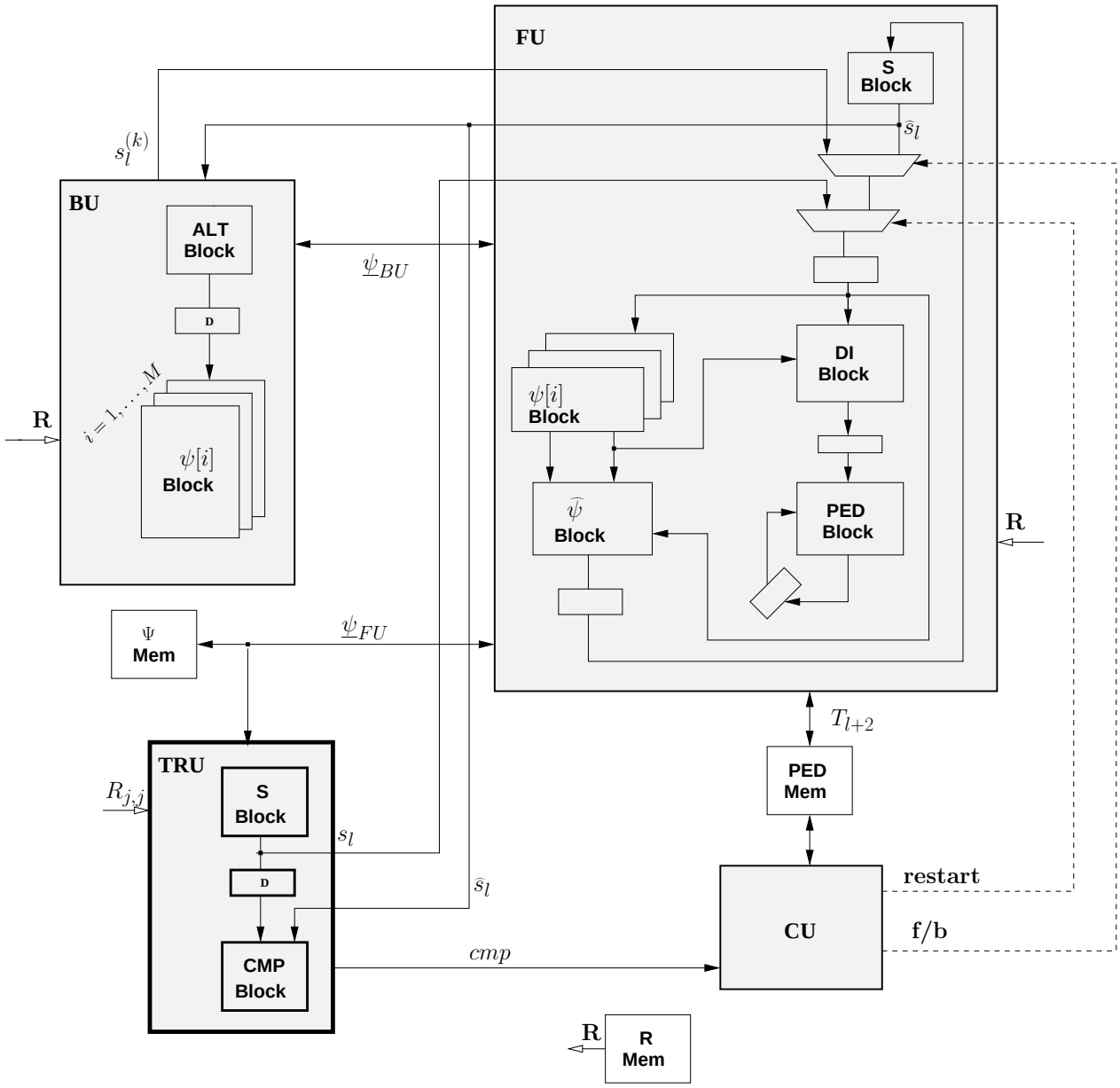


Figure 3.15: High level implementation architectures for LASDA scheme.

continued in the opposite case. The second control operation makes use of the *restart* signal: signal *cmp* informs the CU about the comparison between the $\widehat{s}_l$ symbol, calculated by the FU, and the Babai point s_l, and if the two symbols are different, the CU stops the current processing flow and starts it again from the Babai point.

Table 3.2: Combinational delay ($t_{pd_{min}}$) for additional architecture blocks in Fig. 3.15

Block	$t_{pd_{min}}$[ns]
ALT block	1.39
CMP block	0.34
R Mem	2.05
Ψ Mem	2.05

3.5 Synthesis results

To assess the achievable throughput and occupied area, the datapath of the proposed architecture (tailored to a 4×4 MIMO channel with 16 QAM constellation) has been synthesized on the same technology and with the same tool mentioned in Section 3.3. The internal precision has been set to 16 bits, which guarantees almost the same performance as the floating point model [50].

Besides running the synthesis of the whole detector, key blocks have been also separately synthesized. Tables 3.3 reports the minimum combinational delay for additional blocks in the whole architecture of LASDA, providing results for the CMP block used to manage the Test & Restart search strategy, the ALT block in the TRU and R and Ψ memories. Memories are obtained by means of a RAM generator.

Area complexity given in Table 3.3 was derived with a target clock period of 2.05 ns, corresponding to the delay of the S block. Given results reveal that most of area complexity in the detector is due to the $\psi[i]$ blocks (M blocks need to be allocated in FU and BU), while the additional components needed in the LASDA, namely $\widehat{\psi}$ block, the second S block and the CMP block in the TRU have a limited cost, lower than 25000 μm^2. It is also worth noticing here that area cost for $\psi[i]$ and DI blocks increases linearly with the number of levels in the tree, and so with the number of transmit antenna. On the contrary, the area complexity of the all other blocks reported does not depend on the assumed signal constellation: higher order modulations result into a larger number of sons to be considered at each tree level, but this does not affect the forward processing.

Table 3.3: Synthesis results of computational blocks in the whole architecture: occupied area at the maximum clock frequency $f_{ck} = 487$ MHz. Rows 2 to 8 refer to processing blocks required for SDA; rows 9 and 10 refer to the additional component needed to support LASDA

Block	$Area\ [\mu m^2]$ @ $f_{ck} = 487$ MHz
S block	3124
$\psi[i]$ block	39947
DI block	6566
PED block	12331
ALT block	403
R Mem	36269
Ψ Mem	36269
$\widehat{\psi[i]}$ block	21493
CMP block	38

3.6 Comparisons with the state of the art

To facilitate comparisons, Table 3.4 reports the main features of a number of sphere decoder implementations: four different k–best SD implementations proposed in [46] [47] [16] [15] (columns 2 to 5), the ASIC II in [1] (column 6), the solution in [2] (available for both 0.25 *um* and 0.13 *um* technologies), which resorts to "pipeline interleaving" to improve throughput (columns 7 and 8), and an ML serial detector implementation based on [3] (column 9). Column 10 reports the achieved results for the LASDA implementation. The "pipeline interleaving", described in [2], consists in cutting the combinational delay and processing multiple independent streams of incoming symbols. Look–ahead and pipeline interleaving provide the same kind of advantage, that is the achievement of higher clock frequency with respect to the original SDA. However, the two approaches achieve different cost - throughput trade-offs, as shown later in this discussion.
Technology and algorithm differences must be taken into account in the comparison: while architectures proposed in [3] and in this work offer full ML performance, the other compared implementations give close to ML performance; moreover solutions in columns 4 to 7 are related to an older Silicon technology, and finally implementation in column 2 addresses a 64QAM modulation

Table 3.4: Comparisons with other works

Reference	[46]	[47]	[16]	[15]	[1]	[2]		[3]	This work
MIMO system	4×4								
Modulation	64 QAM	16 QAM							
BER Perf.	Close to ML							ML	
Technology $[\mu m]$	0.065	0.13	0.25	0.35	0.25	0.25	0.13	0.13	0.13
Max. f_{CK} [MHz]	158	150	100	100	71	180	333	250	488
Area $[EG]$	1760K	438K	669(1)	91K	50K	73K	90K	30K	34K
Area $[mm^2]$	2.38(1)	2.23	13.4	5.76	1.0 (1)	1.46 (1)	0.49 (1)	0.16	0.18
Av. Throughput [Mbps] @SNR=22dB	463	2,400	95	53.3	193	488	903	196	381
TAR [Mbps/KEG]	0.26	5.48	0.14	0.58	3.86	6.68	10.03	6.53	11.21

(1) Estimated value, obtained assuming an area occupation, of a two input NAND gate, of 20 μm^2, 5.4 μm^2 and 1.35 μm^2 respectively for 0.25 μm, 0.13 μm and 0.065 μm CMOS standard cell technologies

and uses a 65 nm technology. Table 3.4 gives for each implementation the corresponding technology, maximum clock frequency (Max. f_{CK}), area complexity (Area), expressed in both mm^2 and equivalent gate count (EG), and average throughput (Av. Throughput) at SNR=22 dB. To evaluate the VLSI efficiency of the compared solutions, the last row in Table 3.4 gives the Throughput to Area Ratio (TAR), derived as the ratio between average throughput expressed in Mbps and occupied area evaluated as KEG. Looking, in particular, at the achievable throughput, reported results show that pipelining [47], pipeline interleaving [2] and the proposed look–ahead method are effective techniques to break the throughput bottleneck in the forward processing of SDA and to achieve high throughput. However different penalties in terms of complexity are paid for this advantage. The TAR figure of merit can be used to compare these approaches: pipelining in [47] requires a huge amount of area, which significantly affects the

TAR; pipeline interleaving proposed in [2] also results into a very high throughput, but the corresponding increment in the implementation complexity is larger than that required by look–ahead. The larger area cost of the pipeline interleaving approach basically comes from the need of replicating internal memories, which retain data over multiple iterations. On the contrary, look–ahead only needs a few extra resources to implement Test & Restart strategy. Therefore the two approaches provide two different trade-offs between area and throughput.

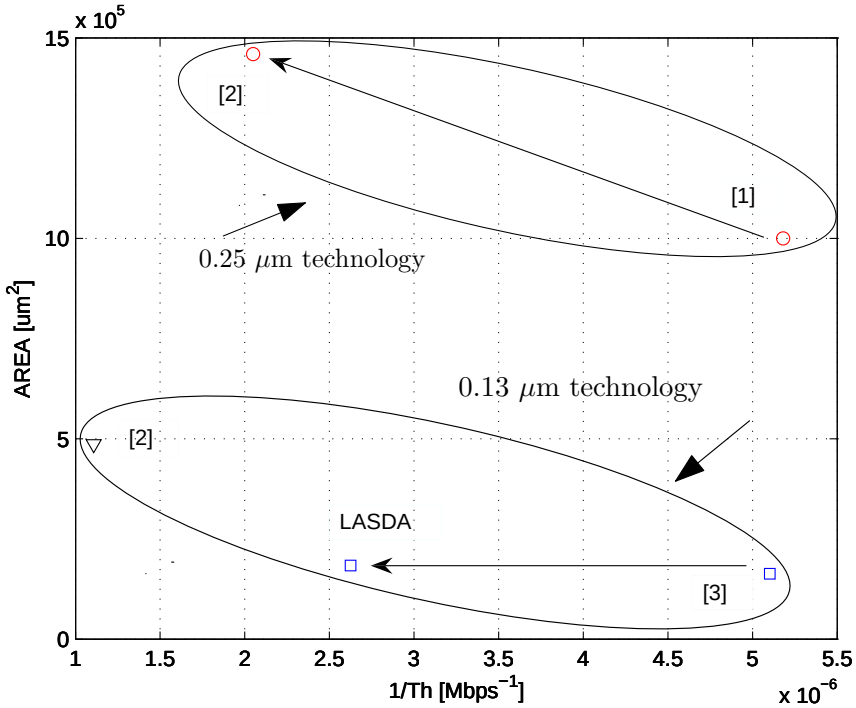


Figure 3.16: Area-delay comparison between solutions in [1], [2], [3] and this work, at SNR=22dB. 1/Th indicates the inverse of the average throughput.

The area-delay plot in Fig. 3.16 shows how pipeline interleaving and look–ahead span the design space. The global detector area is reported along the vertical axis, while the per unit time processed symbols are on the x axis (inverse of the throughput, 1/Th). Two points in the upper part of the plot indicate the 0.25 μm implementations in [1] and [2]: the slope of the arrow that links the two points shows that pipeline interleaving provides a huge throughput advantage at the cost of a significant area increase. Unfortunately the same comparisons

between the two mentioned implementations cannot be done for the 0.13 μm technology, as synthesis results are available only for the pipeline interleaved detector in [2], which is shown as an isolated point in the left part of the plot.

Finally, in the lower part of the plot, two 0.13 μm architectures are reported: a serial detector based on [3] and the look-ahead solution proposed in this work. The arrow here proves the potential of the look–ahead method, which provides large throughput gain at the cost of an almost negligible additional area.

3.7 Discussion of the results

A modified Sphere Decoder Algorithm for high throughput implementation is proposed (LASDA). Look–ahead, pipelining and retiming techniques together with a modified search strategy (Test & Restart) are applied to the original SDA to increase throughput, without penalty in BER performance. Synthesis results of the above architecture show a throughput speedup of 92% at $SNR = 20$ dB (BER $5.2 \cdot 10^{-4}$) with respect to a serial unpipelined SDA solution. From the implementation point of view, the extra resources required by LASDA with respect to SDA introduce an overhead of 24% in terms of occupied area. The proposed solution achieves the best throughput to area ratio among compared SD implementations.

Moreover, this high throughput ML solution is also applicable for most soft–output detection schemes. As shown in [18] [32], both hard and soft–output sphere decoders share several key processing units, particularly those necessary for tree search. As a consequence, not only hard–output, but also soft–output sphere decoding can take advantage of the proposed approach, which achieves a throughput gain in tree search.

Part II

Soft MIMO detection

4

Towards Soft Detection

In order to improve data rate and quality of service of wireless communications, the information is protected, thanks to the Forward Error Correction (FEC) (also called channel coding) at the transmitter side. It is a system of error control for data transmission, whereby the sender adds systematically generated redundant data to its messages, also known as an error-correcting code. The American mathematician Richard Hamming pioneered this field in the 1940's and invented the first FEC code, the Hamming (7,4) code, in 1950. The carefully designed redundancy allows the receiver to detect and correct a limited number of errors occurring anywhere in the message without the need to ask the sender for additional data. FEC is therefore applied in situations where retransmissions are relatively costly, or impossible such as broadcasting to multiple receivers. Error-correcting codes are usually distinguished between convolutional codes and block codes. The convolutional codes are processed on a bit-by-bit basis and they are particularly suitable for hardware implementation. The Viterbi decoder allows optimal decoding for such a king of codes. The block codes are processed on a block-by-block basis. Early examples of block codes are repetition codes, Hamming codes and multidimensional parity-check codes. They were followed by a number of efficient codes, like the Reed-Solomon codes, which were the most notable due to their current widespread use. Turbo codes and Low-Density Parity-Check codes (LDPC) are relatively new constructions that can provide almost optimal efficiency.
In MIMO scenario, at the receiver side a MIMO detector is combined with a

channel decoder: the so called Turbo principle [51] [52]. In fact, the success of iterative decoding for Turbo codes [53] [54] suggests that a new way to devise high-performance wireless communications could be found by considering a probabilistic approach where soft information, presenting the probabilities of different alternatives, is used in detection and decoding processes. Many detection, decoding and estimation problems can be reduced to the estimation of certain probabilities. There are two possible schemes of receiver:

- Scheme with no feedback: as shown in Figure 4.1, the MIMO detector receives the channel observations and the received signal vector and produces the soft information, which becomes input of the decoder. It is also called *Soft-Output* detection-decoding, and it is able to improve performance of coded systems up to 3 dB of SNR (see Figure 4.2), with respect to the hard-output case.

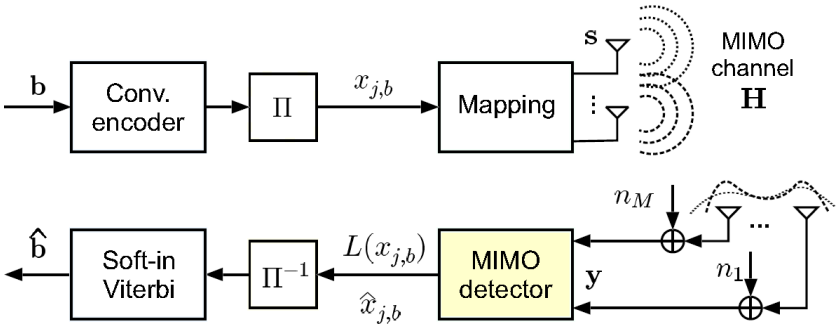


Figure 4.1: General scheme of a transmitter and a receiver with no feedback.

- Scheme with feedback: the detector and the decoder exchanges soft information in an iterative way, as shown in Figure 4.3. It is also known as *Soft-In Soft-Out (SISO)* detection-decoding, and achieves significantly better performance than hard-output or soft-output only.

As MIMO detector, in literature many methods are exploited. There are linear detectors, such as Minimum-Mean-Square-Error-Interference-Canceller (MMSE-IC) and other more complex ones, such as the family of detectors, directly derived from the SDA, called Soft Sphere Decoders. In fact, the Sphere Decoding Algorithm (see section 2.4 in the part I) [55] achieves optimum error-rate performance

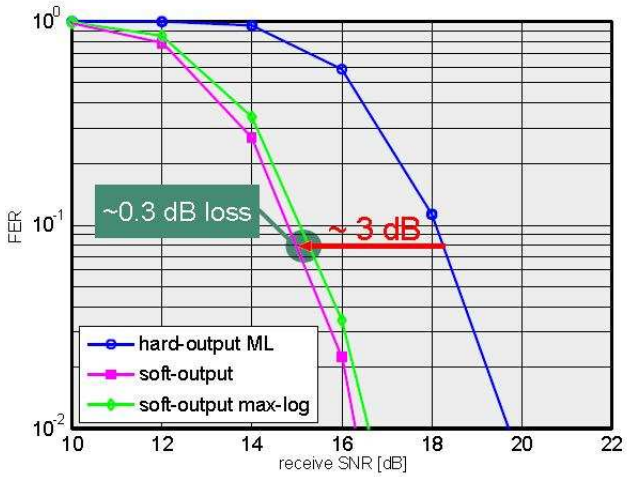


Figure 4.2: FER of a traditional Hard-Output ML Decoder and the Soft-Output Sphere Decoder, implemented in [4](MIMO-OFDM System,4 × 4MIMO, 16-QAM, 64 tones, R = 1/2 convolutional code (K = 7, [133o, 171o]), random interleaver, 1024 bits/codeblock, TGn Type C channel model, BCJR decoder).

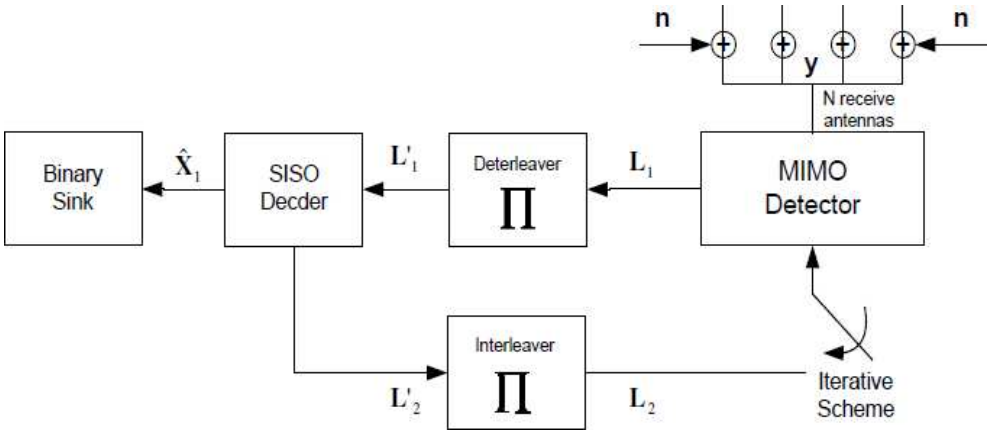


Figure 4.3: General scheme of the receiver with feedback.

for systems without error-correcting code, but for the more relevant case of channel coded systems, it can be used in a modified version in order to provide soft information.

Two main approaches are exploited, in order to modify the SDA:

1. Repeated (RTS) or Single Tree Search (STS): It restarts the SD after the ML solution has been found, forcing the SD onto paths for which the bit is the

opposite of the same bit in the ML solution, the so called *counter-hypothesis*. In particular, the Repeated Tree Search first finds the ML solution, and restarts again the SDA in order to find the counter-hypothesis. In the Single Tree Search approach [19] [4], instead, the ML solution and all other counter-hypothesis are concurrently found.

2. List Sphere Decoder (LSD)[18]: it first reduces the number of candidate vector symbols to those points which are closer to the received vector.These points are kept into a list $\mathcal{L}$ with a certain cardinality and only candidate vector symbols in $\mathcal{L}$ are considered for the computation of soft information.

Next section shows a complexity evaluation, related to an hardware implementation of a suboptimal soft-output MIMO detector, which completes the scenario of soft MIMO detection algorithms.

4.1 Complexity evaluation of a soft-output MIMO detector

The aim of this work, which is based on [56], is the study of feasibility of a sub-optimum soft-output MIMO detector, called Elementary Signal Estimator (ESE), which is the Suboptimal Front End mentioned in Deliverable D4.1, paragraph 2.1.7.1 of FP7 WiMagic Project. This algorithm guarantees the best performance among detectors with polynomial complexity. In [56] different digital architectures are explored, in order to analyze possible performance-complexity trade-offs. This report, which is the Chapter 4, titled Soft Output MIMO detector, in deliverable D7.2 "Prototype design" of P7 WiMagic Project (December 2009), has only the aim of summarize that work, highlighting implementation aspects and synthesis results.

4.1.1 Description of the system

The considered detector includes an outer LDPC channel decoder and a soft-output inner detector for MIMO systems. Figure 4.4 shows the scheme of the transmitter and the iterative structure of the receiver of an Interleave Division Multiplexing-Space Time (IDSM-ST) system, with N_T transmitters N_R receivers.

The role of the receiver is to estimate transmitted bits $\mathbf{d}$, with respect to the received vector $\mathbf{y}$. In particular, the receiver is composed by a sub-optimal iterative detector, called ESE, and an A Posteriori Probability Decoder (APP DEC) [57], which exchange information, as shown in Fig. 4.4.

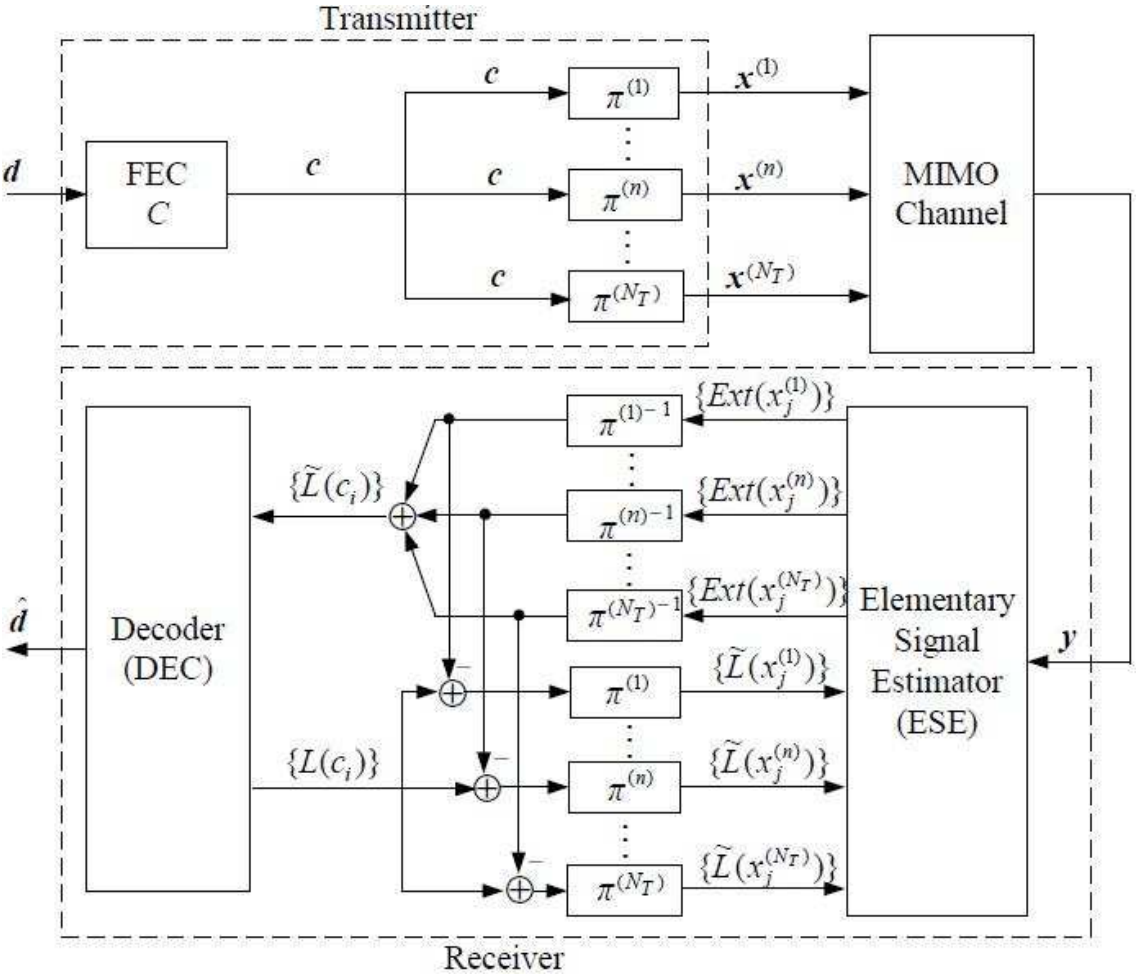


Figure 4.4: Scheme of transmitter and receiver of an IDSM-ST system with B-PSK modulation

Initially the ESE computes coarse estimations $Ext\left(x_j^{(n)}\right)$ of transmitting signals $x_j^{(n)}$, by means of their a priori LLR (Log-Likelihood Ratio) $\overrightarrow{L}_x \equiv \overrightarrow{L}\left(x_j^{(n)}\right)$ (initially set to zero) and by means of received signals $\overrightarrow{y}$, where $Ext\left(x_j^{(n)}\right)$ are

defined as:

$$Ext\left(x_j^{(n)}\right) \equiv \log\left(\frac{\Pr\left(x_j^{(n)} = d|\overrightarrow{y}, \overrightarrow{L}_x\right)}{\Pr\left(x_j^{(n)} = d_0|\overrightarrow{y}, \overrightarrow{L}_x\right)}\right) \tag{4.1}$$

The outputs of ESE are then used by APP DEC, in order to compute a posteriori LLRs with a standard APP decoding [53], so that the ESE can refine the estimations in the next iteration. This mechanism is repeated a predefined number of iterations, here chosen equal to 10. In the last iteration, the APP DEC produces hard decisions $\widehat{\mathbf{d}}$ on $\mathbf{d}$.

4.1.1.1 The algorithm of the Elementary Signal Estimator

Assuming $\left\{\alpha_m^{(n)}, n = 1, \cdots, N_T, m = 1, \cdots, N_R\right\}$ are the fading coefficients between the n-th transmitter and the m-th receiver, modeled as random complex Gaussian variables with zero mean, the received vector at time j can be represented as:

$$\begin{bmatrix} y_j^{(1)} \\ \vdots \\ y_j^{(m)} \\ \vdots \\ y_j^{(N_R)} \end{bmatrix} = \begin{bmatrix} \alpha_1^{(1)} & \cdots & \alpha_1^{(n)} & \cdots & \alpha_1^{(N_T)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \alpha_m^{(1)} & \cdots & \alpha_m^{(n)} & \cdots & \alpha_m^{(N_T)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \alpha_{N_R}^{(1)} & \cdots & \alpha_{N_R}^{(n)} & \cdots & \alpha_{N_R}^{(N_T)} \end{bmatrix} \begin{bmatrix} x_j^{(1)} \\ \vdots \\ x_j^{(1)} \\ \vdots \\ x_j^{(N_T)} \end{bmatrix} + \begin{bmatrix} n_j^{(1)} \\ \vdots \\ n_j^{(m)} \\ \vdots \\ n_j^{(N_R)} \end{bmatrix} \tag{4.2}$$

where $n_j^{(m)}$ denotes a sample of an N_R -dimensional additive i.i.d. (independent and identically distributed) white Gaussian noise vector. Eq. (4.2) can be written in matricial notation, as:

$$\overrightarrow{y}_j = [\alpha]_j \overrightarrow{x}_j + \overrightarrow{n}_j \tag{4.3}$$

The a priori LLR of $x_j^{(n)}$ is defined as:

$$\log\left(\frac{\Pr\left\{x_j^{(n)} = d\right\}}{\Pr\left\{x_j^{(n)} = d_0\right\}}\right) = \left(\overrightarrow{L}_x\right)_d \qquad \text{with} \qquad d = d_1 \div d_{2^{(\frac{m}{2}-1)}} \tag{4.4}$$

where m is the modulation efficiency and $\Pr\left\{x_j^{(n)} = d\right\}$ denotes the a priori probability that $x_j^{(n)}$ is equal to d and it can be computed by inverting (4.4):

$$\frac{\Pr\left\{x_j^{(n)} = d\right\}}{\Pr\left\{x_j^{(n)} = d_0\right\}} = \exp\left(\left(\overrightarrow{L}_x\right)_d\right) \tag{4.5}$$

Keeping into account the contribution of each symbol of the constellation $\mathcal{O}$ to the sum of exponentials, it can be written:

$$\sum_{d' \in \mathcal{O}} \exp\left(\left(\overrightarrow{L}_x\right)_{d'}\right) = \frac{\sum_{d' \in \mathcal{O}} \Pr\left\{x_j^{(n)} = d'\right\}}{\Pr\left\{x_j^{(n)} = d_0\right\}} = \frac{1}{\Pr\left\{x_j^{(n)} = d_0\right\}} \tag{4.6}$$

Now, substituting (4.6) in (4.5), the following equation is obtained:

$$\Pr\left\{x_j^{(n)} = d\right\} = \frac{\exp\left(\left(\overrightarrow{L}_x\right)_d\right)}{\sum_{d' \in \mathcal{O}} \exp\left(\left(\overrightarrow{L}_x\right)_{d'}\right)} \tag{4.7}$$

Let $x_j^{(n)}$ be a random variable, the mean $\mathrm{E}(\cdot)$ and the variance $\mathrm{Var}(\cdot)$ of $x_j^{(n)}$ can be computed starting from (4.7):

$$\mathrm{E}\left(x_j^{(n)}\right) = \sum_{d \in \mathcal{O}} d \Pr\left\{x_j^{(n)} = d\right\} = \frac{\sum_{d \in \mathcal{O}} d \exp\left(\left(\overrightarrow{L}_x\right)_d\right)}{\sum_{d' \in \mathcal{O}} \exp\left(\left(\overrightarrow{L}_x\right)_{d'}\right)} \tag{4.8a}$$

$$\mathrm{Var}\left(x_j^{(n)}\right) = 1 - \left(\mathrm{E}\left(x_j^{(n)}\right)\right)^2 \tag{4.8b}$$

In order to avoid excessive growing of the exponential function, it is convenient to replace the exponent with the difference between the actual LLR value and the maximum LLR, as shown below:

$$\mathrm{E}\left(x_j^{(n)}\right) = \frac{\sum_{d} d \exp\left(\left(\overrightarrow{L}_x\right)_d - \overrightarrow{L}_{max}\right)}{\sum_{d'} \exp\left(\left(\overrightarrow{L}_x\right)_{d'} - \overrightarrow{L}_{max}\right)} \tag{4.9}$$

The received signal at time j has mean

$$\mathrm{E}\left(\overrightarrow{y}\right) = \sum_{n'=1}^{N_T} \overrightarrow{\alpha}_j^{(n)} \mathrm{E}\left(x_j^{(n)}\right) \tag{4.10}$$

and covariance

$$[R]_j \equiv \mathrm{Cov}\left(\overrightarrow{y}_j\right) = \mathrm{E}\left(\overrightarrow{y}_j \overrightarrow{y}_j^T\right) - \mathrm{E}\left(\overrightarrow{y}_j\right)\mathrm{E}\left(\overrightarrow{y}_j^T\right) = \sum_{n=1}^{N_T} \mathrm{Var}\left(x_j^{(n)}\right) \overrightarrow{\alpha}_j^{(n)} \left(\overrightarrow{\alpha}_j^{(n)}\right)^T + \sigma^2 [I] \tag{4.11}$$

where ^{T} indicates the transposed vector.

For *single-path* channels, where each transmitted signal $x_j^{(n)}$ is considered only in one received signal, the soft estimation of $x_j^{(n)}$ in (4.1) can be computed from (4.2), as in [58]:

$$\begin{aligned} Ext\left(x_j^{(n)}\right) = (d_n - d_0) \frac{\overrightarrow{\alpha}_j^T [R]_j^{-1}\left(\overrightarrow{y} - \mathrm{E}\left(\overrightarrow{y}\right)\right) + \mathrm{E}\left(x_j^{(n)}\right) \overrightarrow{\alpha}_j^T [R]_j^{-1} \overrightarrow{\alpha}_j}{1 - \mathrm{Var}\left(x_j^{(n)}\right) \overrightarrow{\alpha}_j^T [R]_j^{-1} \overrightarrow{\alpha}_j} + \\ -\frac{1}{2}\left(d_n^2 - d_0^2\right) \frac{\overrightarrow{\alpha}_j^T [R]_j^{-1} \overrightarrow{\alpha}_j}{1 - \mathrm{Var}\left(x_j^{(n)}\right) \overrightarrow{\alpha}_j^T [R]_j^{-1} \overrightarrow{\alpha}_j} \end{aligned} \tag{4.12}$$

Looking at (4.12), it is evident that the reverse matrix $[R]_j^{-1}$ for each j is the most complex computation. A simplification of (4.12) is possible applying Cholesky-decomposition [59]. Let $[L]_j$ be the inferior triangular matrix, obtained by Cholesky-decomposition $[R]_j = [L]_j [L]_j^T$, and let us define $g_j^{(n)} \triangleq [L]_j^{-1} \overrightarrow{\alpha}_j^{(n)}$ and $f_j \triangleq [L]_j^{-1}\left(\overrightarrow{y} - \mathrm{E}\left(\overrightarrow{y}\right)\right)$, (4.12) can be written as:

$$\begin{aligned} Ext\left(x_j^{(n)}\right) = (d_n - d_0) \frac{\left(\overrightarrow{g}_j^{(n)}\right)^T \overrightarrow{f}_j + \mathrm{E}\left(x_j^{(n)}\right)\left(\overrightarrow{g}_j^{(n)}\right)^T \overrightarrow{g}_j^{(n)}}{1 - \mathrm{Var}\left(x_j^{(n)}\right)\left(\overrightarrow{g}_j^{(n)}\right)^T \overrightarrow{g}_j^{(n)}} + \\ -\frac{1}{2}\left(d_n^2 - d_0^2\right) \frac{\left(\overrightarrow{g}_j^{(n)}\right)^T \overrightarrow{g}_j^{(n)}}{1 - \mathrm{Var}\left(x_j^{(n)}\right)\left(\overrightarrow{g}_j^{(n)}\right)^T \overrightarrow{g}_j^{(n)}} \end{aligned} \tag{4.13}$$

For clarity, a short description of the iterative decoding algorithm for *single-path* channel with QAM modulation and multiple receivers is reported below:

Algorithm for *single-path* channel with multiple receiver antennas

(a) Initialization: Set $\overrightarrow{L}(x_j^{(n)} = 0) \qquad \forall n, j$

(b) Main iteration:

$$\mathrm{E}\left(x_j^{(n)}\right), \mathrm{Var}\left(x_j^{(n)}\right) = \text{Apriori\_stat}\left(\overrightarrow{L}_x\right) \tag{4.14a}$$

$$\mathrm{E}\left(\overrightarrow{y}_j\right) = \sum_{n=1}^{2N_T} \overrightarrow{\alpha}_j^{(n)} \mathrm{E}\left(x_j^{(n)}\right) \tag{4.14b}$$

$$\overrightarrow{M40b1}_j^{(n)} = \mathrm{Var}\left(x_j^{(n)}\right) \overrightarrow{\alpha}_j^{(n)} \tag{4.14c}$$

$$[S40b]_j = \sum_{n=1}^{2N_T} \overrightarrow{M40b1}_j^{(n)} \left(\overrightarrow{\alpha}_j^{(n)}\right)^T \tag{4.14d}$$

$$[\mathrm{Cov}(y_j)] = [S40b]_j + \sigma^2 [I] \tag{4.14e}$$

$$[L]_j = \text{Cholesky factorization}([\mathrm{Cov}(y_j)]) \tag{4.14f}$$

$$\left[L^{-1}\right]_j = \left([L]_j\right)^{-1} \tag{4.14g}$$

$$\overrightarrow{g}_j^{(n)} = \left[L^{-1}\right]_j \overrightarrow{\alpha}_j^{(n)} \tag{4.14h}$$

$$\overrightarrow{S14}_j = \overrightarrow{y}_j - \mathrm{E}\left(\overrightarrow{y}_j\right) \tag{4.14i}$$

$$\overrightarrow{f}_j = \left[L^{-1}\right]_j \overrightarrow{S14}_j \tag{4.14j}$$

$$M11_j^{(n)} = \left(\overrightarrow{g}_j^{(n)}\right)^T \overrightarrow{f}_j \tag{4.14k}$$

$$M12_j^{(n)} = \left(\overrightarrow{g}_j^{(n)}\right)^T \overrightarrow{g}_j^{(n)} \tag{4.14l}$$

$$M13_j^{(n)} = \mathrm{E}\left(x_j^{(n)}\right) M12_j^{(n)} \tag{4.14m}$$

$$M14_j^{(n)} = \mathrm{Var}\left(x_j^{(n)}\right) M12_j^{(n)} \tag{4.14n}$$

$$S15_j^{(n)} = M11_j^{(n)} + M13_j^{(n)} \tag{4.14o}$$

$$S16_j^{(n)} = 1 - M14_j^{(n)} \tag{4.14p}$$

$$\overrightarrow{Ext}\left(x_j^{(n)}\right) = (s_n - s_0) \frac{S15_j^{(n)}}{S16_j^{(n)}} - \frac{1}{2}\left(s_n^2 - s_0^2\right) \frac{M12_j^{(n)}}{S16_j^{(n)}} \tag{4.14q}$$

More complex operations are detailed below.

4.14a:

$max\_apr = 0$

For $m = 0, \ldots, M1 - 2$

  If $\left(\overrightarrow{L}_x\right)_m > max\_apr$

    $max\_apr = \left(\overrightarrow{L}_x\right)_m$

$var\_app1 = \exp\left(-\frac{max\_apr}{fact}\right)$

$var\_app2 = \text{Const}[0] \cdot var\_app1$

$var\_app3 = var\_app1$

$var\_app4 = (\text{Const}[0])^2 \cdot var\_app1$

For $m = 0, \ldots, M1 - 2$

  $var\_app1 = \exp\left(\frac{\left(\overrightarrow{L}_x\right)_m - max\_apr}{fact}\right)$

  $var\_app2 = var\_app2 + \text{Const}[m+1] \cdot var\_app1$

  $var\_app4 = var\_app4 + (\text{Const}[m+1])^2 \cdot var\_app1$

  $var\_app3 = var\_app3 + var\_app1$

$\text{E}\left(x_j^{(n)}\right) = \frac{var\_app2}{var\_app3}$

$\text{Var}\left(x_j^{(n)}\right) = \frac{var\_app4}{var\_app3} - \left(\text{E}\left(x_j^{(n)}\right)\right)^2$

The parameters $\text{Const}[i]$ with $i = 0, \ldots, 2^{\frac{m}{2}} - 1$ are shown in Table 4.1 for a Q-PSK and a 16-QAM.

Table 4.1: Constants used in the computation of a priori statistics

	Q-PSK	16-QAM
Const[0]	−0.70710678118654757	−0.94868329805051377
Const[1]	+0.70710678118654757	−0.31622776601683794
Const[2]		+0.31622776601683794
Const[3]		+0.94868329805051377
$(\text{Const}[0])^2$	+0.5	+0.9
$(\text{Const}[1])^2$	+0.5	+0.1
$(\text{Const}[2])^2$		+0.1
$(\text{Const}[3])^2$		+0.9

4.14f:

For $m = 1, \ldots, 2N_R$

  For $i = m, \ldots, 2N_R$

    $x = ([\mathrm{Cov}\,(y_i)])_{mi} - \sum_{r=1}^{i-1} l_{ir} l_{mr}$

    If $i = m$

      $l_{mi} = \sqrt{x}$

    else

      $l_{mi} = \frac{x}{l_{ii}}$

4.14g:

For $m = 1, \ldots 2N_R$ do

  $\left([L^{-1}]_j\right)_{mm} = \frac{1}{l_{mm}}$

For $m = 1, \ldots 2N_R - 1$ do

  For $i = m + 1, \ldots, 2N_R$ do

    $S13 = \sum_{r=m}^{i-1} l_{ir} \left([L^{-1}]_j\right)_{rm}$

    $\left([L^{-1}]_j\right)_{im} = -\left([L^{-1}]_j\right)_{ii} S13$

4.14h:

For $m = 1, \ldots, 2N_R$

  $\left(\overrightarrow{y}_j^{(n)}\right)_m = \sum_{r=1}^{2N_R} \left([L^{-1}]_j\right)_{mr} \left(\overrightarrow{\alpha}_j^{(n)}\right)_r$

4.14i:

For $m = 1, \ldots, 2N_R$

  $\left(\overrightarrow{S14}_j\right)_m = (\overrightarrow{y}_j)_m - (\mathrm{E}\,(\overrightarrow{y}_j))_m$

4.14j:

For $m = 1, \ldots, 2N_R$
$\left(\overrightarrow{f}_j\right)_m = \sum_{r=1}^{2N_R} \left([L^{-1}]_j\right)_{mr} \left(\overrightarrow{S14}_j\right)_r$

4.14k:
$M11_j^{(n)} = \sum_{r=1}^{2N_R} \left(\overrightarrow{g}_j^{(n)}\right)_r \left(\overrightarrow{f}_j\right)_r$

4.14l:
$M12_j^{(n)} = \sum_{r=1}^{2N_R} \left(\overrightarrow{g}_j^{(n)}\right)_r \left(\overrightarrow{g}_j^{(n)}\right)_r$

It can be mathematically demonstrated that the decoding complexity grows linearly with the number of transmitters and with the cube of number of receivers [60].

4.1.2 Hardware implementation

The high level block diagram of ESE is depicted in Fig. 4.5, where $L0$, $L1$ and $L2$ represent the a priori probabilities for the computation of mean E and variance Var of transmitted signals. The covariance matrix is obtained by means of $L0$, $L1$ and $L2$ and CSI and stored into a Register File (RF). After Cholesky-decomposition, the resulting upper triangular matrix replaces the original covariance matrix, then it is inverted and stored in the same locations, in order to save memory. The vector $\overrightarrow{f}_j$ (4.14j), which is common to each transmitter antenna, is used in the final computation of the a posteriori probabilities $Ext0$, $Ext1$ and $Ext2$.

The architecture is composed of six main macroblocks:

1. *Apriori_stat*: computes a priori statistics,
2. *cov*: computes the covariance,
3. *cholesky*: applies the Cholesky-decomposition,

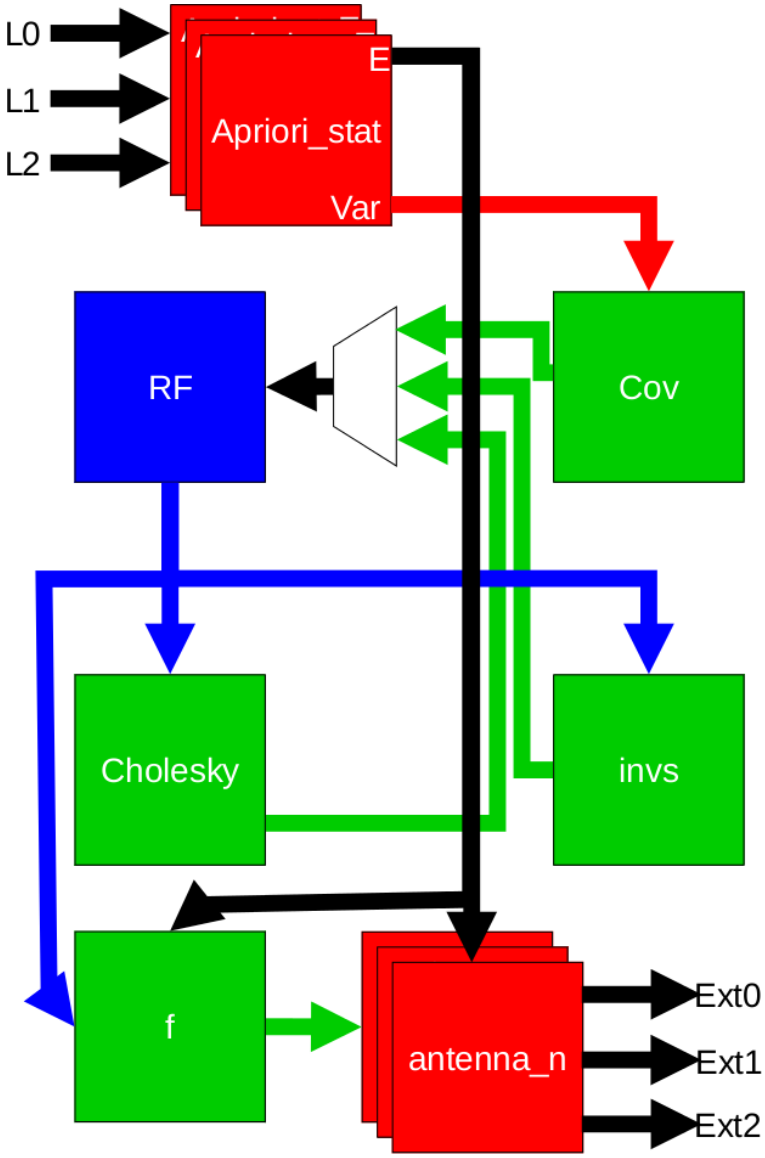


Figure 4.5: Block diagram of ESE

4. *invs*: inverts the input matrix,
5. f: computes the vector of auxiliary parameters $\overrightarrow{f}_j$,
6. *antenna_n*: computes the a posteriori probabilities $Ext0$, $Ext1$ and $Ext2$.

Next paragraphs contains the detailed description of each block of the architecture.

4.1.2.1 *Apriori_stat* block

The *Apriori_stat* block computes mean $E(x_j^{(n)})$ and variance $Var(x_j^{(n)})$ of N_T transmitted signals. In order to avoid the excessive increase of the exponential function, the values are normalized to the highest LLR, by means of comparators and multiplexers, as shown in Fig. 4.6. The highest LLR is subtracted to each LLR computed and the result is stored into the memory MEMORIA exp(-x/8), which produces *var_app*1 for one accumulator and two Multiply-Accumulate (MAC) units. This last block has the aim of computing mean and variance of signal coming from n-th antenna. It is important to underline that a division operation is avoided by means of a multiplication and a second memory MEMORIA 1/x, which stores the inverse (precomputed) of memory address. The *Apriori_stat* block can be allocated $2N_T$ times in a parallel architecture, or just once in a sequential one. In the following, the second case is assumed with the purpose of saving silicon area.

4.1.2.2 *cov* block

The most complex task of the algorithm is the computation of the covariance matrix, which is performed once per iteration. Moreover, the maximum dimension established for a MIMO system is 4×4, equivalent to a 8×8 real–valued system. Thanks to the symmetric structure of the above matrix, it is necessary to store only 36 elements. For each column and for each row, one summation of N_T elements is applied:

$$[\mathrm{Cov}\,(y)]_{ij} = \sum_{n=1}^{2N_T} \mathrm{Var}\left(x^{(n)}\right) \left(\overrightarrow{\alpha}^{(n)}\right)_i \left(\overrightarrow{\alpha}^{(n)}\right)_j^T + \sigma^2\,[I] \tag{4.15}$$

The implementation of the above equation is characterized by three counters: the first, synchronized by the system clock, for index n, the second, synchronized by the terminal count of the first one, for the row index j and the last, synchronized by the terminal count of the second counter, for the column index i. The same indexes are also used as selection signals of multiplexers, whose inputs are variances and coefficients, α and α^T, of channel. The circuit is simple. In order to avoid to lose the time (36 clock cycle) necessary to reset one MAC, two MACs

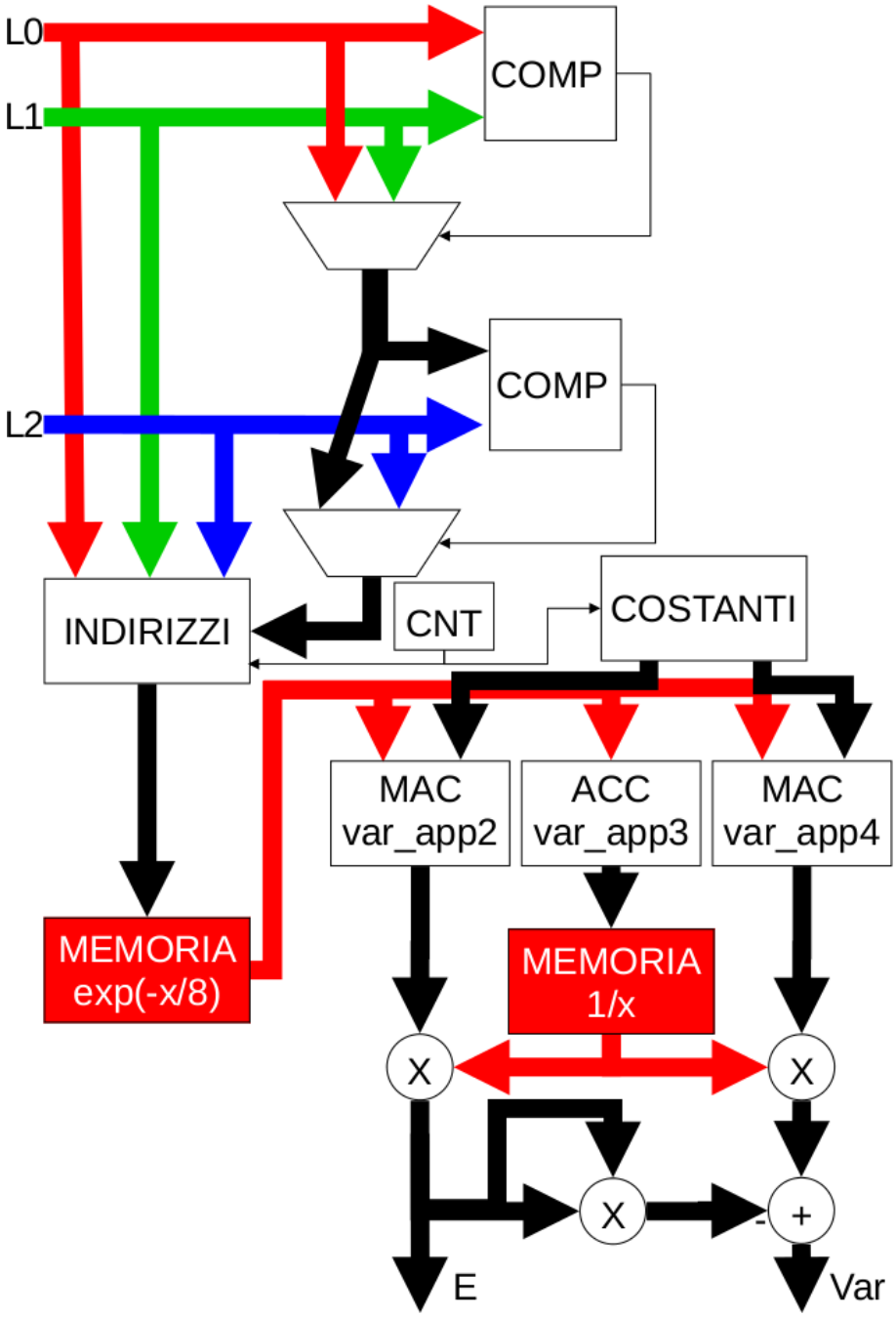


Figure 4.6: Block diagram of *Apriori_stat* block

can be alternately used: while the former is working, the latter can be reset to zero. It is aim of an another multiplexer to choose the final result, based on if it must be or not added to σ^2.

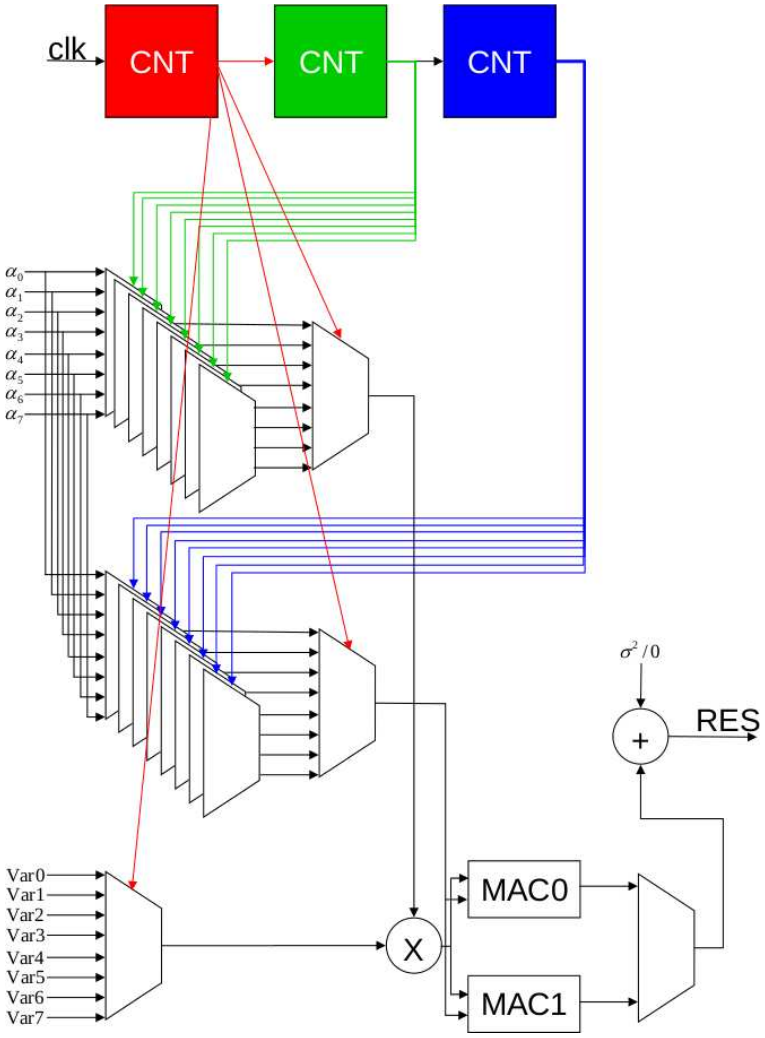


Figure 4.7: Block diagram of *cov* block

4.1.2.3 *cholesky* block

For this block, solution in [61] is chosen, because it requires few clock cycles. A matrix $[A]$, defined as positive in a real–valued system and hermitian in a complex–valued one, can be decomposed, using $[A] = [L]\,[L]^T$, where the inferior

triangular matrix $[L]$ can be computed as:

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \tag{4.16}$$

for diagonal elements and

$$l_{ij} = \frac{1}{l_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) \tag{4.17}$$

for not-diagonal elements. The new values of $[L]$ are stored in the same location occupied by values of $[A]$. The division by $\sqrt{x}$ in (4.17) can be replaced by a multiplication for $\frac{1}{\sqrt{x}}$. When required, $\sqrt{x}$ can be computed as $\sqrt{x} = x\frac{1}{\sqrt{x}}$.

The values of $\frac{1}{\sqrt{x}}$ are stored into a Look Up Table (LUT), called *rom_oos*. Since simulations demonstrate that $x < 16$, the significant bits of input data goes from s^3 to the desired precision, with respect to the chosen signal width. For example, if the architecture would initially implemented on FPGA, in particular a *Xilinx Virtex 5*, which has memories resources of 36Kbits, a 11 bits data width is necessary. So, input data goes from s^{-7} to s^3. The representation of output data is, instead, variable, because it does nott contain initial zeros, as shown in Table. 4.2.

Fig. 4.8 and 4.9 show the block diagrams respectively for the computation of diagonal and not-diagonal elements.

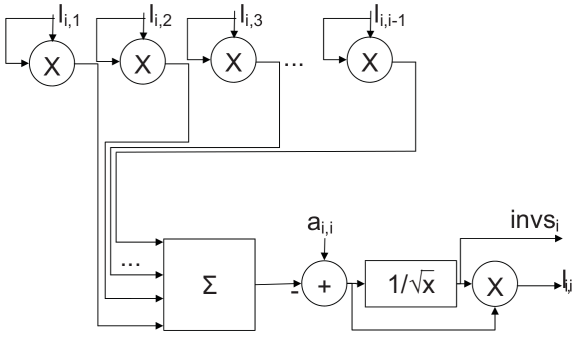


Figure 4.8: Computation of diagonal elements in *cholesky* block.

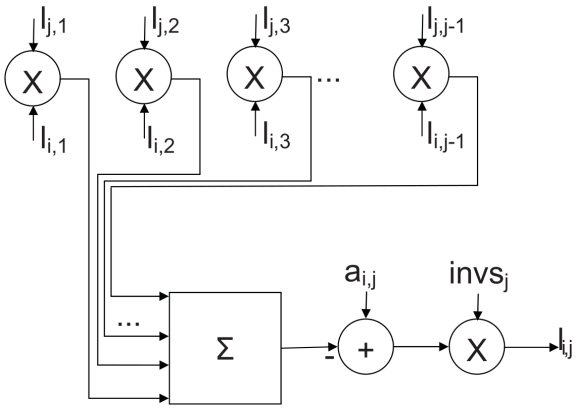


Figure 4.9: Computation of not-diagonal elements *cholesky* block.

Table 4.2: Notation of $1/\sqrt{x}$ stored into the LUT

x	$1/\sqrt{x}$	LUT notation
2^{-7}	$2^{3}\dots 2^{-7}$	$2^{2}\dots 2^{-8}$
2^{-6}	$2^{3}\dots 2^{-7}$	$2^{2}\dots 2^{-8}$
$2^{-6}+2^{-7}$	$2^{2}\dots 2^{-8}$	$2^{1}\dots 2^{-9}$
2^{-5}	$2^{2}\dots 2^{-8}$	$2^{1}\dots 2^{-9}$
2^{-4}	$2^{2}\dots 2^{-8}$	$2^{1}\dots 2^{-9}$
2^{-3}	$2^{1}\dots 2^{-9}$	$2^{0}\dots 2^{-10}$
2^{-2}	$2^{1}\dots 2^{-9}$	$2^{0}\dots 2^{-10}$
2^{-1}	$2^{0}\dots 2^{-10}$	$2^{-1}\dots 2^{-11}$
2^{0}	$2^{0}\dots 2^{-10}$	$2^{-1}\dots 2^{-11}$
2^{1}	$2^{-1}\dots 2^{-11}$	$2^{-2}\dots 2^{-12}$
2^{2}	$2^{-1}\dots 2^{-11}$	$2^{-2}\dots 2^{-12}$
2^{3}	$2^{-2}\dots 2^{-12}$	$2^{-3}\dots 2^{-13}$
$2^{4}-2^{-7}$	$2^{-2}\dots 2^{-12}$	$2^{-3}\dots 2^{-13}$

4.1.2.4 *invs* block

In the computation of $[B] = [L]^{-1}$, diagonal elements are calculated, using Cholesky-decomposition, as:

$$b_{kk} = l_{kk}^{-1} = \sqrt{a_{kk}}^{-1} = \frac{1}{\sqrt{a_{kk}}} \tag{4.18}$$

which is exactly the value stored into the LUT. In this way, *cholesky* block can be modified as shown in Fig. 4.10. As for the not-diagonal elements, *cholesky* block can be reused just changing input data. Looking at Fig. 4.11, it is evident that, for the computation of $b_{i,j}$, entire i-nth row and j-nth column are necessary at the same time. The solution is a smart Register File, composed of 36 registers, some control signals and concerning multiplexers, which choose between two possibilities: row j and row i for Cholesky decomposition or row i and column j for the matrix inversion.

4.1.2.5 *f* block

The equation for the auxiliary parameters $\overrightarrow{f}_j$ is the following:

$$\overrightarrow{f}_j = \left[L^{-1}\right]_j \left(\overrightarrow{y}_j - \sum_{n=1}^{2N_T} \overrightarrow{\alpha}_j^{(n)} \mathrm{E}\left(x_j^{(n)}\right) \right) \tag{4.19}$$

Fig. 4.12 shows the block digram of the f block: it is composed of 8 MAC units for the computation of the parallel summation of all elements of the vector $\overrightarrow{f}_j$, 8 subtractors and a unit, called g and shown in detail in Fig.4.13, for matrix-vector product $\overrightarrow{g}_j^{(n)} = [L^{-1}]_j \overrightarrow{\alpha}_j^{(n)}$.

4.1.2.6 *antenna_n* block

As for *Apriori_stat* block, also *antenna_n* block, Fig. 4.14 , is allocated once and works sequentially for $2N_T$ transmitters. This block computes the auxiliary parameter $\overrightarrow{g}_j^{(n)}$, by means of the g unit, as described before. Then last variables are calculated:

$$M11_j^{(n)} = \left(\overrightarrow{g}_j^{(n)}\right)^T \overrightarrow{f}_j \tag{4.20}$$

and

$$M12_j^{(n)} = \left(\overrightarrow{g}_j^{(n)}\right)^T \overrightarrow{g}_j^{(n)} \tag{4.21}$$

by means of 2 MAC units for $M11_j^{(n)}$ and other 2 MAC for $M12_j^{(n)}$. For each MAC couple, the two units work alternatively: while the first MAC evaluates,

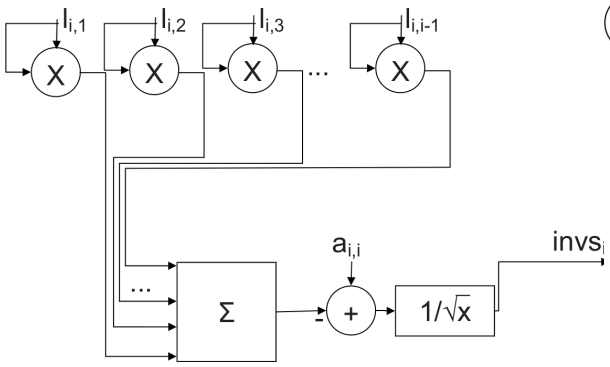


Figure 4.10: Computation of diagonal elements in *invs* block

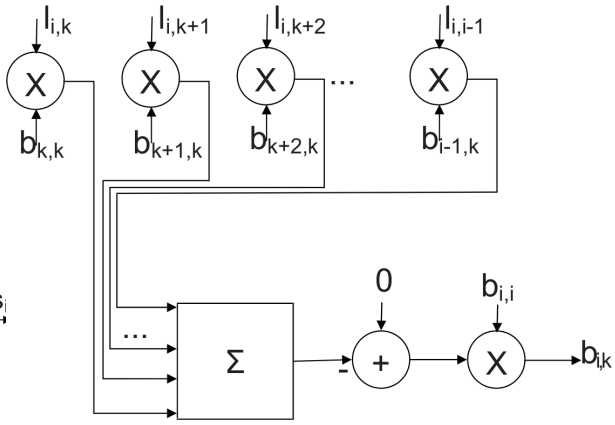


Figure 4.11: Computation of non-diagonal elements in *invs* block

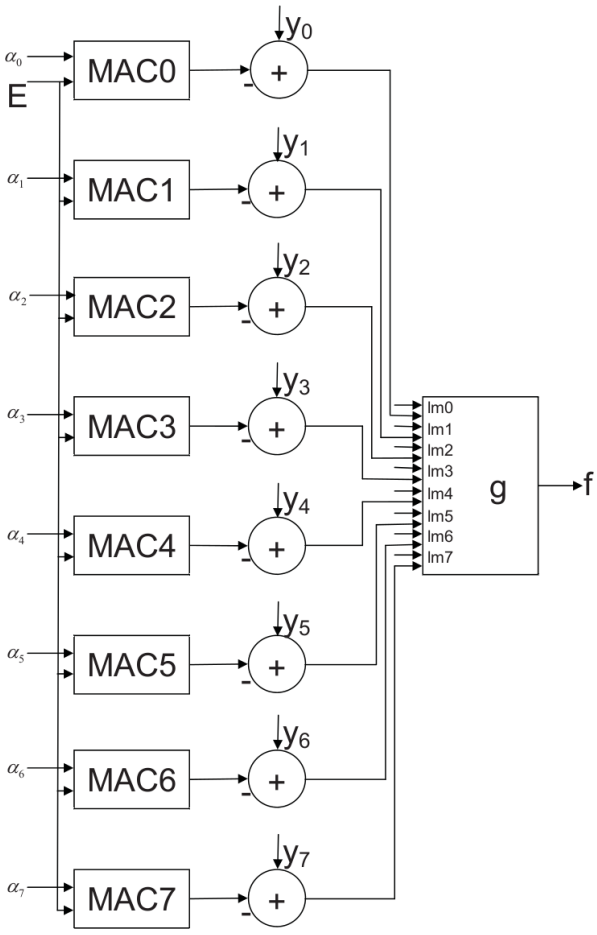


Figure 4.12: Block diagram of f block.

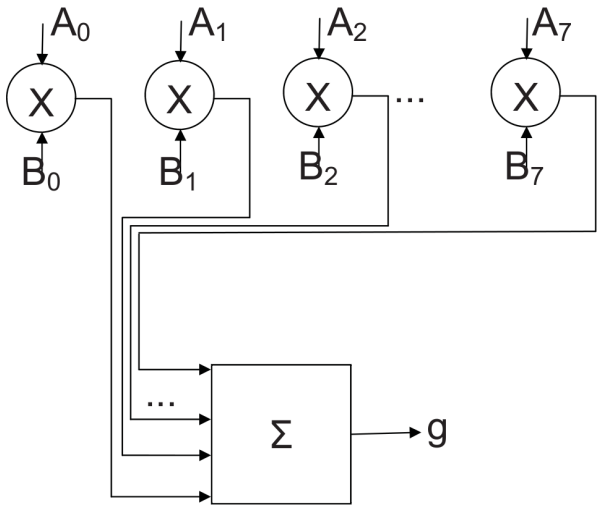


Figure 4.13: Block diagram of g block.

the second one is reset to zero. Final outputs are:

$$\overrightarrow{Ext}\left(x_j^{(n)}\right) = Cost\frac{M11_j^{(n)} + \mathrm{E}\left(x_j^{(n)}\right) M12_j^{(n)}}{1 - \mathrm{Var}\left(x_j^{(n)}\right) M12_j^{(n)}} + Cost2\frac{M12_j^{(n)}}{1 - \mathrm{Var}\left(x_j^{(n)}\right) M12_j^{(n)}} \tag{4.22}$$

Thanks to the common denominator in (4.22), one memory, that as the same structure and functionality of *rom_div* in *Apriori_stat* block, can replace the divisions.

4.1.3 Synthesis results

In order to assess the achievable throughput and occupied area, the proposed architecture, tailored to a MIMO system $\{2 \times 2,\ 3 \times 3,\ 4 \times 4\}$ and modulations {QPSK and 16QAM}, has been synthesized on a 130 nm CMOS Standard Cell technology, using Synopsis Design Compiler version Z-2007.03-SP1. The internal precision has been set to 22 bits [1], which guarantees almost the same performance as the floating point model. Tab. 4.3 shows the detailed silicon area results for each block of the architecture. The total area, reported in the last row, of $852194um^2$ is related to a sequential architecture and the maximum clock frequency of it is $f_{ck} = 100MHz$. As shown in Table 4.3, the most complex units are f and *antenna_n* blocks, because of their strong parallelism. Memories are

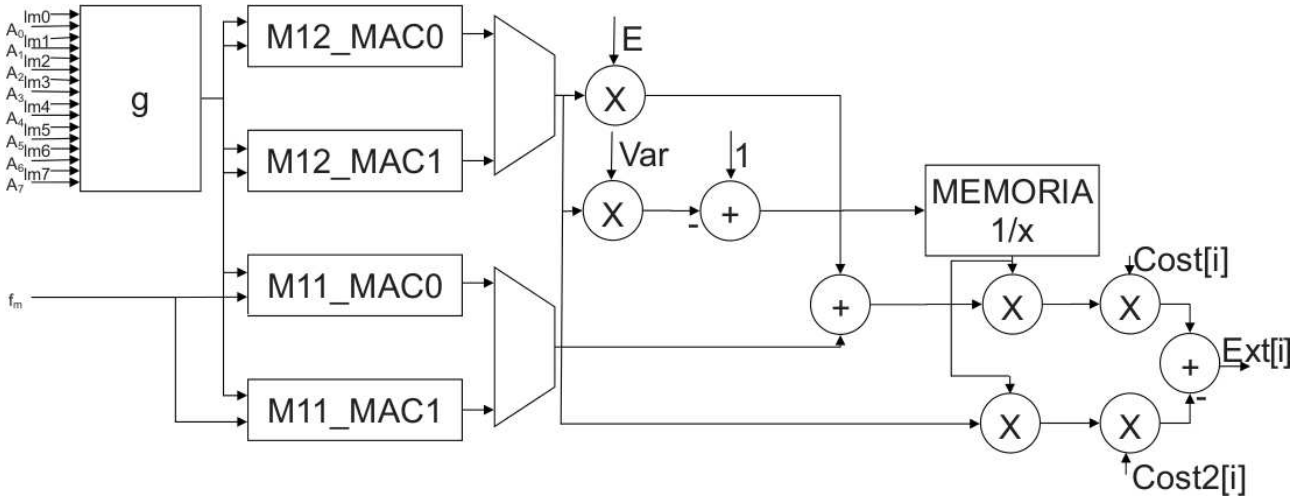


Figure 4.14: Block diagram of *antenna_n* block.

Unit	Area um^2
Apriori_stat	26052
cov	43676
cholesky	134542
invs	94814
f	190059
antenna_n	224748
Total	852194

Table 4.3: Area results of synthesis of ESE (sequential architecture)

Memory	n.	Area um^2
rom_div	2	135605
rom_exp	1	24412
rom_exp	1	99609
Total	4	395231

Table 4.4: Area results of synthesis of memories

generated using a RAM generator for a 130 nm technology, obtaining results in Tab. 4.4. An estimation of throughput can be done in term of LLRs per second, as in the following equation:

$$th = \frac{N_{LLR}}{N_{CK}} f_{CK} \tag{4.23}$$

where N_{LLR} is the number of LLRs, depending on the topology of MIMO system and modulation, N_{CK} the number of clock cycles to produce outputs and f_{CK} the clock frequency. For the case of a 3×3-MIMO system and 16-QAM, the resulting throughput for a sequential and parallel architecture is respectevely:

$$th_{seq} = 6.8 \cdot 10^6 \frac{LLR}{s} \tag{4.24}$$

$$th_{par} = 9.3 \cdot 10^6 \frac{LLR}{s} \tag{4.25}$$

In the sequential architecture each block, shown in Tab. 4.3, is allocated once and work in a sequential way, so that *Apriori_stat* and *antenna_n* must be reused a number of times equal to the double of the number of transmitters. The resulting throughput is low and silicon area is the minimum. In the parallel scheme, instead, *Apriori_stat* and *antenna_n* are allocated $2N_T$ times, obtaining an increase of area and a further increase in term of throughput. Moreover, some blocks contain functional units that are the same used in other blocks: in order to optimize the area occupation, those resources could be shared, keeping into account that the throughput decreases.

[1] In some block of the architecture the data width is set less than 22 bits

5

Soft MIMO detection: the idea of a multi-algorithm detector

After a brief overview on the introduction of soft detection in MIMO communications presented in the previous chapter, here the idea of an multi-algorithm detector is introduced. Such an idea is detailed and supported by an analysis of existing hard and soft MIMO detector implementations.

5.1 State of the Art

In MIMO scenario, linear detectors are suboptimal, in terms of error rate performance, with respect to Soft Sphere Decoders, but they have also a reduced computational complexity. Therefore, it is fair to ask if it is possible to combine in some way the high BER performance, achieved by the family of Soft Sphere Decoders, and the low computational complexity, obtained with a linear detector, such as MMSE-IC. In order to have a precise answer, this chapter reports a detailed analysis of hard and soft detectors, exploited in literature. It points out the advantages of the soft detection with respect to the hard one, and represents a feasibility study, which justifies the previous idea.
Table 5.1 shows a comparison between different solutions, presented in literature, included [13], which is the VLSI design of the Look–ahead SDA presented in chapter 3. The ASIP design in [6] is related to a Minimum-Mean-Square-Error Interference Canceller Equalizer (Soft-Input Soft-Output). The throughput results are obtained for an iterative system, with MMSE-IC equalizer combined with Turbo

decoder, and the number of iterations is equal to 5. It is implemented as ASIC on a 90 nm technology. Then two VLSI designs of near-ML Soft-Output MIMO detectors are presented in [12] and [9]. The QR-decomposition is not included in area results of the former, while it is comprised in the latter. In particular, solution in [12] is based he Layered ORthogonal Lattice. Already mentioned in chapter 1, the work in [4] proposes a near ML Soft-Output Single-Tree-Search Sphere Decoder, complex-valued and with Schnorr-Euchner enumeration. The error rate performance ranges from exact max-log soft-output to hard-output SIC. The MMSE-SQRD stage preprocessing is not included in the area results. Finally, a Soft-Output MIMO Detector, dynamically reconfigurable for QPSK, 16-QAM and 64-QAM modulation schemes for 4×4 MIMO system, is presented in [7]. The QR-decomposition is not included in the area results.

Before starting the analysis of the introduced implementations, some clarifications related to the parameters, shown in the Table 5.1 are also necessary:

- All papers, except [9], show area results in Equivalent Gate (EG) unit, as shown in the row 11 of the Table 5.1 . The correspondent values in mm^2 are obtained, according to $A_{1gate} = 320f^2$, where $f = feature\ size$, as reported in ITRS'10.
- The unit of measurement for the throughput (row 12) in [6] is MSymbol/s. These values must be multiplied by the number of bits per symbol, according to the modulation (1 bit for a BPSK, 2 bits for QPSK, 4 bits for 16QAM and 6 for 64QAM), and divided by the number of iterations (5 in this case) in order to obtain the correspondent values in Mbps.
- Solutions in [9], [13] and [4] are characterized by a variable throughput, depending on SNR value. [9] reports the worst case (not defined in the paper). In order to have a direct comparison between [13] and [4] a value of $SNR = 18dB$ has been chosen. The remaining papers have, instead, a fixed throughput and they cannot directly be compared to [9], [13] and [4].
- In order to have a correct comparison, a reference technology of $90nm$ has been chosen, and area, frequency and throughput for different technologies are scaled, according to ITRS'10. In particular, the normalized throughput (row 12) and normalized clock frequency (row 10) are computed considering

Ref.	Baghdadi *et al.* [6]	[12]	[9]	My work [13]	Burg *et al.* [4]	[7]
Standard	IEEE 802.11n, 802.16e, LTE	IEEE 802.11n				
Detection	Soft			Hard	Soft	
MIMO	2×2, 3×3, 4×4	2×2		4×4		
Modulation	BPSK ,QPSK, 16 and 64QAM	64QAM		16QAM		QPSK, 16 and 64QAM
Receiver	Turbo		Convolutional		Viterbi	
Algorithm	MMSE-IC (ASIP)	LORD	SSD	LASDA	STS-SESD	SO
Implem.	ASIC					
Tech. $[nm]$	90	65	45	130	250	45
BER Perf.	close to ML			ML	close to ML	
f_{max} $[MHz]$	546	80	312	488	71	500
Norm.90nm	546	40	124.8	976	568	200
Area $[KEG]$	84.3	236	293	34	56.8	70
Area$[mm^2]$	0.22	0.32	0.19	0.18	1.14	0.04
Norm.90nm $[mm^2]$	**0.22**	**0.61**	**0.76**	**0.09**	**0.15**	**0.18**
Th. $[Mbps]$	273MSps$_{2\times2}$ 148MSps$_{3\times3}$ 168MSps$_{4\times4}$	164$_{2\times2}$	307	295	40	1000$_{QPSK}$ 500$_{16QAM}$ 187.5$_{64QAM}$
Norm.90nm	**29.6to327.6**	**82**	**122.8**	**590**	**320**	**75to400**
Th/Area $[Mbps/mm^2]$	**135to1499**	**134**	**162**	**6556**	**2173**	**413to2205**

Table 5.1: Comparisons between works in [6], [12], [9], [13], [4], [7].

a clock frequency increase of $2x$ per technology generation until 2007 (65nm) and of $1.25x$ per technology generation after the same year. The normalized throughput in [6] and [7] is given as a range, from the minimum to the maximum values (*min to max*). The normalized area (row 11) is computed according to $A_{90} = A(\frac{f_{90}}{f})^2$, where $f = feature\ size$, $f_{90} = 90nm$, $A = area\ in\ the\ original\ technology$ and $A_{90} = normalized\ area\ in\ 90nm$.

- Architecture efficiency metric has been measured through the Throughput to Area Ratio (TAR), computed as normalized throughput divided by normalized area, already defined in subsection 3.6.

5.2 Analysis

Analyzing Table 5.1, there are four points to be discussed:

1. **Soft-Output vs Hard-Output**: a Soft-Output Detector has a gain up to $3dB$, with respect to an Hard-Output Detector, as already said. A Soft-Output detector presents an increase of area of about 60% [4], also pointed out comparing [13] and [4]. This is due to the higher number of symbols to be visited and to the additional computational complexity required to compute LLRs. The BER of solution in [13] cannot be directly compared to that in [5], also shown in Fig. 5.1, because the former is related to the output of MIMO detector and the latter, instead, is related to the output of the decoder. Moreover no channel coding is assumed in the former, while a convolutional code is employed in the latter.
   Anyway the performance improvement, with a limited increase of occupied Silicon area, justifies the recent interest towards soft-output detection in MIMO systems.
2. **Optimal Soft-Output vs Max-Log Soft-Output**: as reported in [4], the Max-Log approximation produces only up to $\sim 0.3dB$ loss of SNR, with respect to Optimal Soft-Output (see Fig. 4.2), but achieves a great reduction of computational complexity.
3. **Soft-Output MMSE vs Soft-Output and fixed-throughput MIMO Decoder**: this analysis concerns solutions in [6], [12] and [7]. About FER

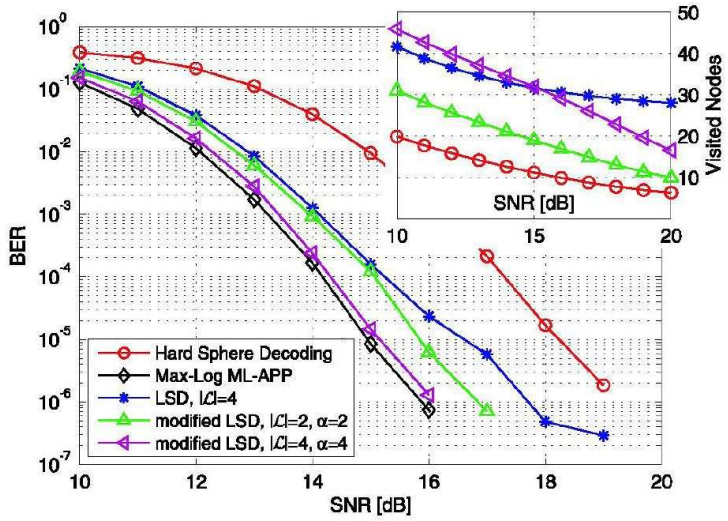


Figure 5.1: BER, shown in [5], of a traditional Hard-Output Sphere Decoder, the List Sphere Decoder and the Max-Log ML-APP. Coded 4 x 4 MIMO system, 16-QAM, Fast-fading channel, rate 1/2, length of the code-block is 1024 bits and the code has constraint length K = 7 with generator polynomials $[133_0,171_0]$.

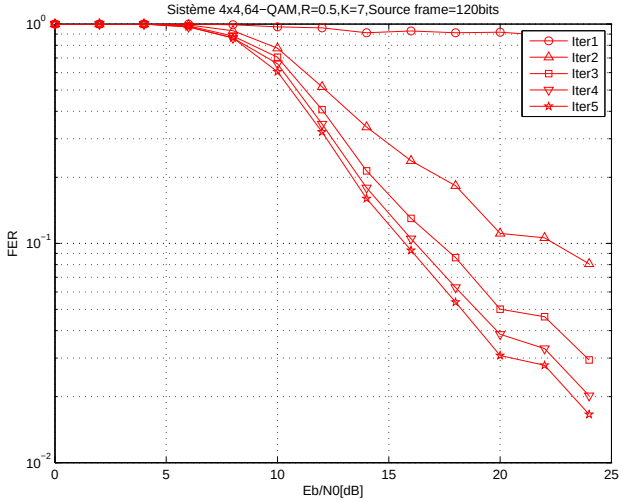


Figure 5.2: FER of [6] (4×4MIMO, 64QAM, Frame source= $120bits$, Channel matrix constant over 240bits (10 vectors of symbols), R=1/2 convolutional code).

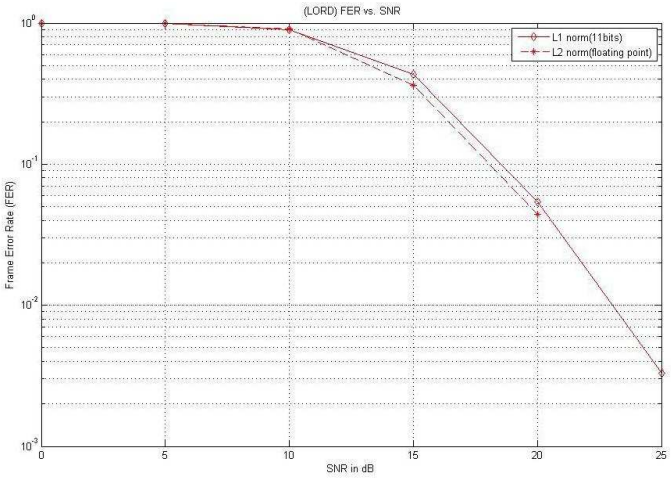


Figure 5.3: FER of different solution, reported in [7] (4×4 and 64-QAM, R = 1/2 [7 5] convolutional code, Channel matrix constant over a frame, frame size=10 vectors (24bits per symbol, 10 symbols), Viterbi decoder).

performance, [12] does not report any results, and it just states to achieve near-ML performance. So it is not comparable to the others, because of the absence of information. Instead, looking at Fig. 5.2 and Fig. 5.3, it is evident that the performance of MMSE with 3 iterations is comparable with that of LORD: in fact for a $FER = 5 * 10^{-2}$ both have $SNR \sim 20dB$. For more iterations MMSE has better performance. This improvement is due to iterations: the same equalizer can achieve a gain of some dBs if it iterates. In fact, paper [8] shows as an Iterative Full-Rate STBC equalizer gains $2dB$ of SNR for a $BER = 6 * 10^{-2}$ with 4 iterations with respect to the not-iterative one, as demonstrated in Fig. 5.4 and Fig. 5.5. This attests as a sub-optimal equalizer, like MMSE, can achieve better performances than an optimal one (optimal for not-iterative algorithms), if it becomes iterative. Analyzing the throughput over area ratio for all three papers, [6] shows the

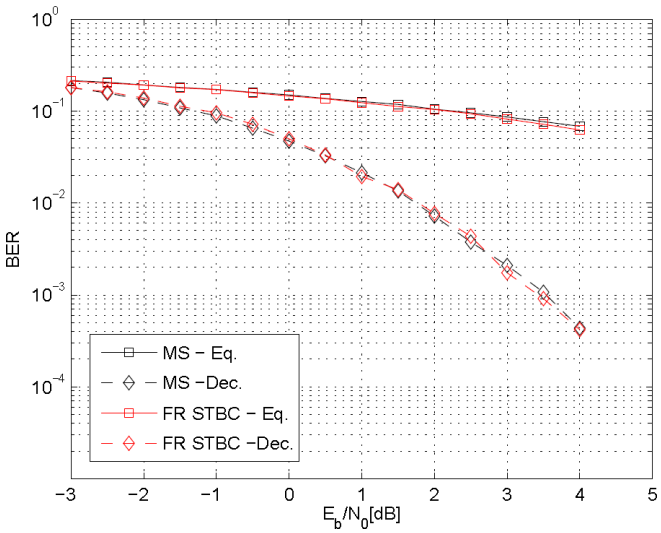


Figure 5.4: BER of a full-rate space-time block code (FR STBC) equalizer and decoder (red curves) without iterations, shown in [8].

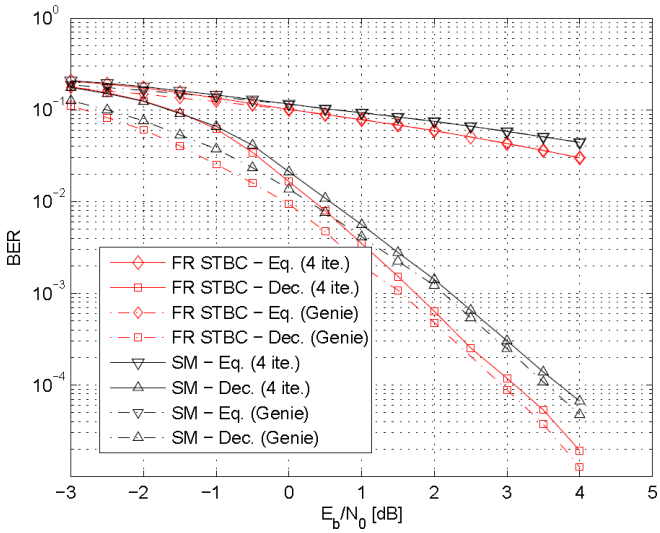


Figure 5.5: BER of a full-rate space-time block code (FR STBC) equalizer and decoder (red curves) with 4 iterations, shown in [8].

highest among all, as shown in Tables 5.3 and 5.4.

This is due to the fact that both [12] and [7] propose a K-best strategy, guaranteeing a fixed throughput. They focused on increase parallelism, visiting more points in the same time, and, clearly, requiring more area.

Ref.	[13]	[4]
Th/Area $[\frac{Mbps}{mm^2}]$	6556	2173

Table 5.2: Comparisons of architecture efficiency in [13] (Hard-Ouput Sphere Decoder) and [4](Soft-Output Sphere Decoder) for SNR=18dB.

Ref.	Baghdadi *et al.* [6]	[12]
Th/Area $[\frac{Mbps}{mm^2}]$	1499	134

Table 5.3: Comparisons of architecture efficiency in [6] and [12] for a 2×2MIMO and 64-QAM.

Ref.	[6]	[7]
Th/Area $[\frac{Mbps}{mm^2}]$	923	413

Table 5.4: Comparisons of architecture efficiency in [6] and [7] for a 4×4MIMO and 64-QAM.

4. **Soft-Output MMSE vs Soft-Output Sphere Decoder**: a Soft-Output Sphere Decoder gets a gain of $\sim 3.2dB$ with rate of 2/3 and $\sim 5.2dB$ with rate of 5/6 for a $FER = 10^{-2}$ with respect to MMSE, as reported in [9] (see Fig. 5.6). This is the further reason why to iterate is necessary, transforming a Soft-Output MMSE into a Soft-Input Soft-Output MMSE. A number of iterations from 5 to 7 is sufficient to have better performance [8]. In fact, comparing the curve of Soft-Output ML in Fig. 4.2 and MMSE (2 iterations) in Fig. 5.7, it can be seen that for a $FER = 2 * 10^{-2}$ the latter as a gain of $\sim 2.3dB$ with respect to the former.

   The reduction of area, achieved by the Soft-Output Sphere Decoder in [4] is about 32.6% with respect to MMSE-IC in [6], but the solution in [4] is related to a particular system and a particular constellation. The throughput and the TAR cannot be analyzed in this comparison, because [6] has a fixed throughput and [4] a variable one, based on channel conditions.
   Also the [9] does not specify which value of SNR or FER is associated to the reported worst case throughput, so that it is not possible to compare its results to [4] and, therefore to [6].

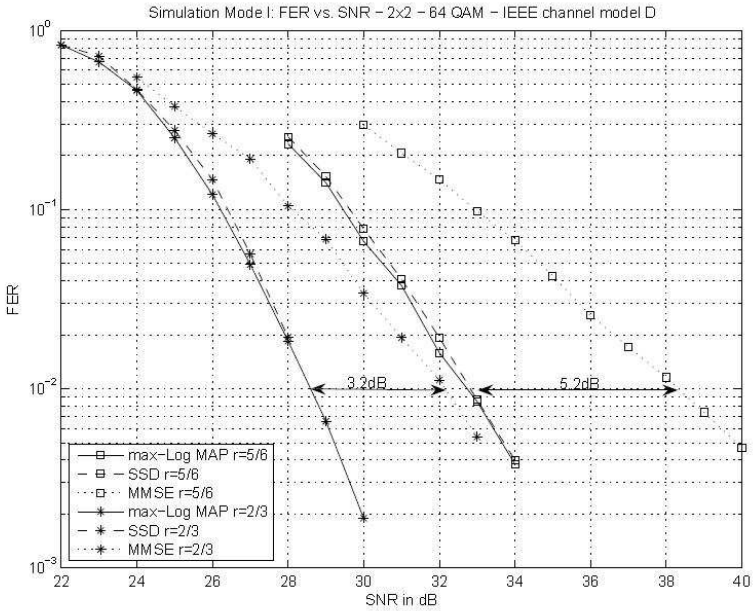


Figure 5.6: FER of different solution, reported in [9] (IEEE 802.11n 2×2MIMO and 64QAM, channel model D, 20 Mhz bandwidth and coding rates 5/6 and 2/3).

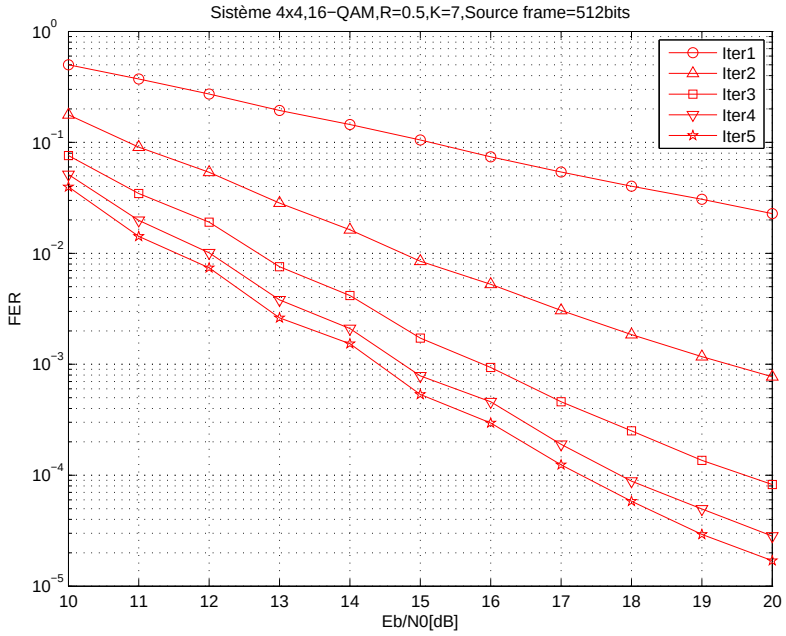


Figure 5.7: FER of [6] (4×4MIMO and 16QAM, R=1/2, Source frame=512bits).

Therefore, the comparison must be limited to [6] and [4], also because [13] is related to Hard-Output MIMO detection. Therefore, it is necessary a detailed analysis between this above papers.

(a) **Area**: in order to have a more precise comparison in term of Silicon area, it is necessary to consider that results in [6] includes a matrix inversion, which is performed through an analytical method. But, in many works in literature, such as [62], [63], [64] the matrix inversion includes a QR-decomposition and it consists of tree steps: 1) QR-decomposition, 2) a matrix inversion (R^{-1}) and 3) a matrix multiplication ($A^{-1} = R^{-1}Q$) [62]. Also the Soft-Output Sphere Decoder needs a preprocessing stage to apply QR-decomposition to channel matrix, but area results in [4] does not include it. The table 5.5 shows some implementations of QR-decomposition in different technologies.

Ref.	[65]	[66]	[67]
Matrix dimensions	4×4 complex-valued		8×8 real-valued
Technology $[nm]$	250	180	130
Frequency $[MHz]$	125	162	not shown
Area [KGE]	54	61.8	27

Table 5.5: Area results of different implementations of QR-decomposition for MIMO detection.

It is reasonable taking into account results in [67], since it is the more efficient and one of the most recent work among them. The normalized area in $90nm$ technology is $A_{QR} = 0.069984mm^2$, and by adding it to results in [4], a rough estimation of the total area required by QR-decomposition and Soft SD is obtained $A_{QR+SSD} = 0.2172mm^2$, i.e. $\sim 6\%$ less than MMSE in [6]. This percentage seems to be low, but the Soft SD replaces the MMSE equalizer and the soft demapper, because its output are estimated symbols and LLRs, as Fig. 4.1 shows. So, an estimation of occupation of demapper is useful to have a more clear idea.

Regarding this, paper [68] proposes an ASIP implementation of a universal demapper for multi-wireless standards and for one receiver. The area result must be multiplied by the number of receivers, which is 4, obtaining $A_{demapper} = 4 * 8KGE = 32KGE$. Now, adding this value to that shown in [6], a rough estimation of the total area of MMSE+demapper is obtained, $A_{MMSE+demapper} = 116.3KGE = 0.3014mm^2$. The reduction achieved by the Soft Sphere Decoder is about 27.9% with respect to MMSE. Certainly, this is the case of a 4×4 system and 16-QAM. For smaller systems and constellations, the area will be furthermore reduced. Anyway, the Soft SD has a very limited increase for higher number of antenna and higher order system, due to the more registers, necessary to store more visited points and related metrics. In fact, the search along the tree is depth-first, which means that the number of computational resources is the same, even if the number of levels and the number of sons per father node increase. So it can be expected that the area of the Soft SD is less or at most equal to MMSE with demapper for all cases (2×2, 3×3, 4×4, B-PSK, Q-PSK, 16-QAM, 64-QAM).

(b) **Throughput**: the throughput of Soft SD is variable with channel conditions, number of antennas and modulation. When channel conditions are good (high SNR), the number of visited nodes is low, and throughput high. Otherwise the algorithm needs to visit more nodes because of the presence of high level of noise. Moreover, usually the throughput decreases when number of antennas and order of constellation increase. It would seems that Soft SD is more suitable for smaller systems and modulations. In fact, for the case of (4×4 and 16-QAM) and $FER = 10^{-2}$ Soft SD guarantees $320Mbps$ at $SNR = 18dB$, which is 5% less than that of MMSE ($336Mbps$). This last value is not reported in the Table 5.1, because in this case 2 iterations are sufficient to have better performance of FER: so it must be divided by 2 and not by 5. By the way, the throughput of [4] is related to $SNR = 18dB$ and this means that the difference, between throughput of Soft SD and MMSE, reduces when SNR increases, since throughput of Soft SD becomes much more higher.

(c) **Architecture efficiency**: the discussion can be similar to the previous. Soft SD obtains the best efficiency for high SNR. The architecture efficiency must be computed, taking into account the area of QR+Soft SD for [4] and MMSE+demapper for [6]. About throughput of [6], a 4×4 system, with 16-QAM modulation, $FER = 10^{-2}$ is considered: the original throughput must be divided by 2, as in the previous case. Results are: $TAR_{MMSE+demapper} = \frac{336Mbps}{0.3014mm^2} = 1115\frac{Mbps}{mm^2}$ and $TAR_{QR+SSD} = \frac{320Mbps}{0.2172mm^2} = 1473\frac{Mbps}{mm^2}$. So, Soft SD has a value of throughput to area ratio, which is about $\sim 32\%$ more with respect to MMSE.

Therefore, Soft SD could be a good candidate as substitute of MMSE, under certain conditions. Probably for low SNRs MMSE is better, because with some iterations it achieves better performance and higher throughput than Soft SD; instead for high SNRs Soft SD has a better architecture efficiency. Anyway, this point can be furthermore investigated, also considering an iterative detection-decoding system. The key word in this research is the flexibility, not only in terms of trade-off between performance and complexity, but also meaning a detector able to fit in different numbers of antennas, modulation orders and wireless standards. As Soft-Input Soft-Output SD, the List Sphere Decoder has been chosen, since it is the easiest extension of a traditional depth–first hard SDA, and offers some parameters, such as the size of the list and the LLR clipping, which can be tuned, to obtain the desired performance-complexity trade-off.

6

Flexible Soft-Input Soft-Output detector: List Sphere Decoding and Linear MMSE Detection

While chapter 5 justifies the idea of a multi-algorithm flexible detector, this chapter illustrates a detailed analysis of a Soft-Input Soft-Output LSD in terms of iterative behavior and flexibility parameters. BER performance and computational complexity of LSD have been also explored and compared to those of linear MMSE-IC.

6.1 Description of the system

In order to analyze the iterative behavior of the LSD algorithm, a traditional scheme for data transmission and reception has been adopted, as Fig. 6.1 shows. The system model is represented by eq. (2.2), in section 2.2. A stream of bits is encoded, randomly interleaved and mapped onto a constellation. In particular, a convolutional code (rate 1/2, generator polynomials $[133_0\ 171_0]$ and constraint length 7) and an hard Gray mapper are assumed. Then the stream of symbols is mapped onto M_t transmitters, through a serial-to-parallel block. The channel model is assumed to be flat, i.e. all frequency components of the signal experience the same magnitude of fading (Rayleigh). Moreover, it can be either stationary over a frame composed by several vectors of M_t transmitted symbols (block-fading), or it can change from vector to vector (fast-fading). Since the channel

model has a great impact on performance of the detector, in this work both models are considered. At the receiver side, a MIMO detector is combined with a channel decoder. They reciprocally exchange soft information. The detector takes as inputs channel observations y and the *a priori* information coming from the decoder. The soft outputs of the detector become the *a priori* information for the decoder. The channel decoding is performed by means of a BCJR decoder, with either MAP or Max-log-MAP approximation. As for MIMO detection, two algorithms are considered: LSD and MMSE-IC. In Fig. 6.1, L^{D1} and L^{D2} represent the *a posteriori* Log-Likelihood Ratios (LLRs) respectively of the detector and of the decoder, while L^{E1}, L^{E2} and L^{A1}, L^{A2} the correspondent *extrinsic* and *a priori* LLRs. A description of LSD and MMSE-IC algorithms is reported in

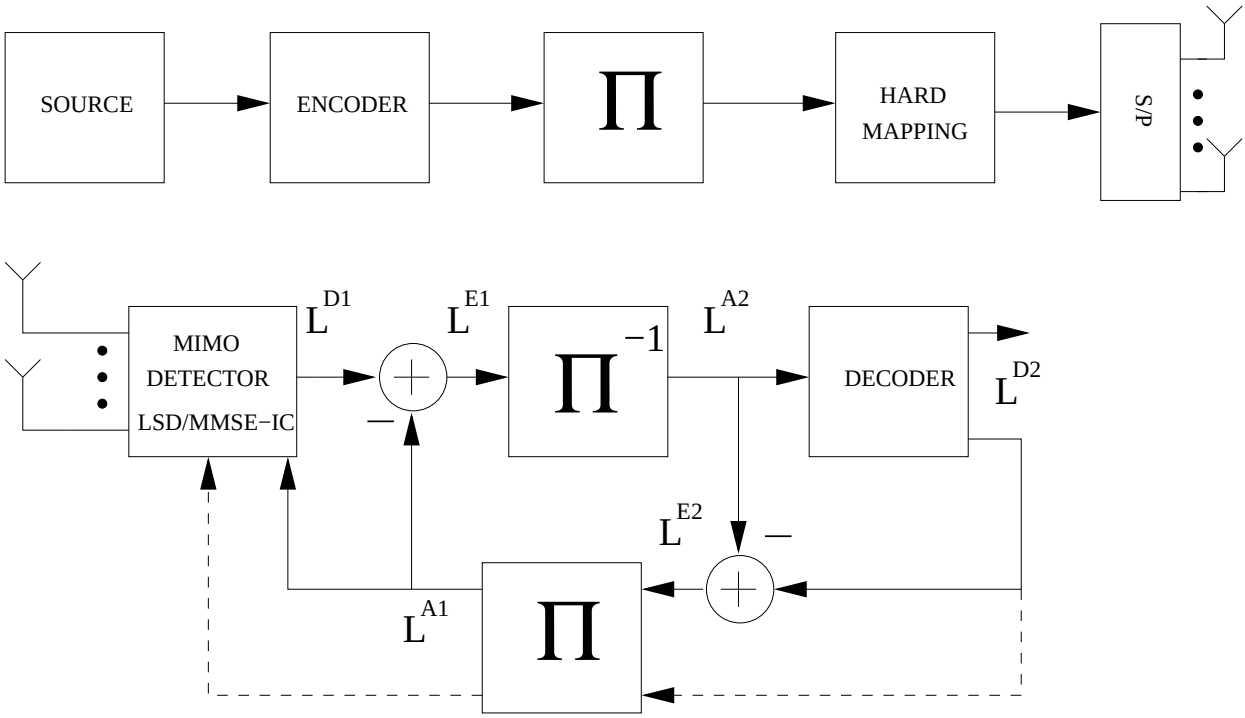


Figure 6.1: Scheme of the MIMO transmitter and receiver.

the two following subsections.

6.1.1 List Sphere Detector (LSD)

The LSD algorithm was proposed in [23], as a straightforward extension of the sphere decoder [55]. It finds not only the ML solution, but a list of $\mathcal{L}$ candidates

which have the smallest distance from the received vector.
Considering a block of bits $\mathbf{x}$, the *a posteriori* symbol probability can be expressed by means of LLR [23] as:

$$L_k^D = \log\left(\frac{P[x_k = +1]|\mathbf{y}, \mathbf{H}}{P[x_k = -1]|\mathbf{y}, \mathbf{H}}\right) \tag{6.1}$$

with $k = 1 \dots QM_t$. $x_k = -1$ and $x_k = +1$ represent respectively the logical zero and the logical one. Using Bayes theorem, the *a posteriori* L_k^D can be written as:

$$L_k^D = L_k^A + L_k^E \tag{6.2}$$

where $L_k^A = \log\left(\frac{P[x_k=+1]}{P[x_k=-1]}\right)$ represent the *a priori* LLR values coming from the decoder, and L_k^E are the *extrinsic* LLRs. According to (6.2) and with the Max-log approximation, the *extrinsic* LLRs can be directly computed by the detector as [23]:

$$L_k^E \approx \frac{1}{2}\max_{x\in\mathcal{L}_k^{+1}}\{-\frac{1}{\sigma^2}\|\mathbf{y}-\mathbf{Hs}\|^2 + \mathbf{x}_{[k]}^T\mathbf{L}_{[k]}^A\} - \frac{1}{2}\max_{x\in\mathcal{L}_k^{-1}}\{-\frac{1}{\sigma^2}\|\mathbf{y}-\mathbf{Hs}\|^2 + \mathbf{x}_{[k]}^T\mathbf{L}_{[k]}^A\} \tag{6.3}$$

where $\mathbf{s} = map(\mathbf{x})$, $\mathbf{x}_{[k]}$ denotes the subvector of $\mathbf{x}$ obtained by omitting its k-th element x_k, and $\mathbf{L}_{[k]}^A$ denotes the vector of all $\mathbf{L}^A$ values omitting L_k^A. $\mathcal{L}_k^{+1}$ and $\mathcal{L}_k^{-1}$ denote respectively the subset of candidates in the whole list with the bit $x_k = +1$ and the subset with $x_k = -1$.

6.1.2 MMSE-IC Linear Equalizer

In the context of low-complexity suboptimal algorithms the Minimum-Mean-Square-Error Interference-Canceller (MMSE-IC) was developed [69][70]. The detector receives symbols corrupted by the noise of the channel and co-antenna interference. Then it partially cancels the interference and symbols are demapped into LLRs and deinterleaved. The decoder removes the noise and produces soft information values, which are interleaved and given as feedbacks to the detector. Since it is shown in [71] that if the decoder and the detector exchange *a posteriori* information for the computation of symbol expectations, the receiver achieves faster convergence rate and better performance, we use *a posteriori* information

as feedback. The detected symbols can be expressed using time invariant approximation [72] as:

$$\tilde{s}_i = \lambda_i \mathbf{p}_i^H (\mathbf{y} - \mathbf{H}\hat{\mathbf{s}} + \hat{s}_i \mathbf{h}_i) \tag{6.4}$$

where $i = 1 \dots M_t$, $\hat{\mathbf{s}}$ is the vector of decoded symbols, $\hat{s}_i$ is the i-th element of this vector, $\mathbf{h}_i$ is the i-th column of matrix $\mathbf{H}$ and $(.)^H$ represent the Hermitian operator. The parameters λ_i and $\mathbf{p}_k$ are given by:

$$\lambda_i = \frac{\sigma_s^2}{1 + \sigma_{\hat{s}}^2 \mathbf{p}_i^H \mathbf{h}_i} \tag{6.5}$$

$$\mathbf{p}_i = \mathbf{E}^{-1} \mathbf{h}_i \tag{6.6}$$

In the previous equation the matrix $\mathbf{E}$ is:

$$\mathbf{E} = ((\sigma_s^2 - \sigma_{\hat{s}}^2)\mathbf{H}\mathbf{H}^H + \sigma_n^2 \mathbf{I}) \tag{6.7}$$

where σ_s^2, $\sigma_{\hat{s}}^2$ and σ_n^2 are variances respectively of transmitted symbols, received symbols and noise of the channel. $\mathbf{I}$ is the identity matrix.

6.2 Flexibility and divergence analysis of iterative LSD

6.2.1 Flexibility parameters

The size of the list in LSD has an impact on both performance (BER) and computational complexity. If a detector has the capability to adaptively change the list size, this can be exploited to achieve a desired trade-off between BER performance and complexity.

In the context of iterative decoding, for a low number of antennas and low modulation order, a small list is enough in order to achieve good performance. In fact the list size is directly related to the system configuration. If a short list size is applied for high number of antennas and/or high modulation order rather than having an improvement of BER with iterations, a problem of divergence [24][10] appears, so that a bigger list is necessary.

In literature two methods are mainly exploited to handle the list: the first method,

introduced in [23], consists in using the list obtained at the first iteration also for all subsequent iterations; the second method updates also the list at each iteration [73]. As showed in [24], this last method, which implies much more computations, is equivalent to generate at the first iteration a larger list and to extract from the same list different sub-sets at each following iteration; this technique avoids the divergence and achieves further improvements in terms of performance. On the other hand, from the implementation point of view, the size of the list is a parameter with a great impact in term of latency and requirement of memory. Thus list size can be used to search a proper trade-off between performance and complexity.

The second key element of soft-output LSD is the "LLR clipping" level [23]: when one of the two sets $\mathcal{L}_k^{+1}$, $\mathcal{L}_k^{-1}$ is empty, the LLR value for the bit under consideration should be set to a certain value. This predefined value has an impact on performance and convergence of the iterative process. Different choices are proposed in literature: ± 8 in [23], ± 3 in [73], ± 50 in [10], $\pm r^2$, where r is the radius of the hypersphere, in [5] or $\pm \ln \frac{1-P_b}{P_b}$, where P_b is the probability of a bit error at the detector output and it depends on the modulation, in [74].
The divergence is mainly caused by the unreliability of a smaller list [24][10], but a better understanding of this phenomenon can be obtained via EXIT-chart, as explained in the next subsection.

6.2.2 Analysis of divergence using EXIT chart

First proposed in [25] for parallel concatenated codes, and later extended to iterative demapping and decoding [75], the Extrinsic Information Transfer Chart is a powerful tool to analyze the iterative behavior of the system. The detector can be treated as a demapper. According to L^{A1}, L^{A2}, L^{E1} and L^{E2} in Fig. 6.1, I^{A1} and I^{A2} represent the *a priori* mutual input information respectively of the detector and of the decoder, and I^{E1} and I^{E2} the correspondent *extrinsic* mutual output information. The EXIT chart formulation is based on the fact that the extrinsic information L^{E1} (i.e. L^{A1}) coming back from the decoder, are almost Gaussian distributed [75]. Additionally, large interleavers keep the a priori L^{A1} fairly uncorrelated over many iterations. Hence, the a priori input L^{A1} can be

modeled, applying an independent Gaussian random variable n_{A1}, with variance σ_{A1}^2 and zero mean μ_{A1}, according to:

$$L^{A1} = \mu_{A1} x + n_{A1} \tag{6.8}$$

where $x \in \{\pm 1\}$ represent the known transmitted unmapped bits. Since L^{A1} is supposed to be Gaussian, $\mu_{A1} = \sigma_{A1}^2/2$ is fulfilled. The conditional probability density function belonging to L^{A1} value is:

$$p(L^{A1}/X = x) = \frac{1}{\sqrt{2\pi\sigma_{A1}^2}} e^{\frac{-(L^{A1} - x\frac{\sigma_{A1}^2}{2})^2}{2\sigma_{A1}^2}} \tag{6.9}$$

It can be demonstrated that the Shannon's mutual information between the equally likely X and the respective LLRs L^{A1}, for symmetric and consistent L^{A1} values, is $I^{A1} = I(L^{A1}; X)$, $0 \leq I_{A1} \leq 1$[75]:

$$I^{A1} = \frac{1}{2} \sum_{x=+1,x=-1} \int_{-\infty}^{+\infty} p(L^{A1}/X = x) log_2 \left(\frac{2p(L^{A1}/X = x)}{p(L^{A1}/X = -1) + p(L^{A1}/X = +1)} \right) dL^{A1} \tag{6.10}$$

Then, with eq. (6.9), (6.10) becomes:

$$I^{A1} = 1 - \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma_{A1}^2}} e^{\frac{-(L^{A1} - x\frac{\sigma_{A1}^2}{2})^2}{2\sigma_{A1}^2}} log_2 \left(1 + e^{-L^{A1}} \right) dL^{A1} \tag{6.11}$$

Moreover, for a generic standard deviation σ, defining $J(\sigma) := I^{A1}(\sigma_{A1} = \sigma)$:

$$\lim_{\sigma \to 0} J(\sigma) = 0 \qquad \lim_{\sigma \to \infty} J(\sigma) = 1 \qquad with \;\; \sigma > 0 \tag{6.12}$$

The function $J(\sigma)$ is directly connected to the capacity of the AWGN channel, it cannot be expressed in closed form. It is monotonically increasing and reversible [76].

$$\sigma_{A1} = J^{-1}(I^{A1}) \tag{6.13}$$

In the same way, also extrinsic mutual output information $I^{E1} = I(L^{E1}; X)$ can be expressed as:

$$I^{E1} = \frac{1}{2} \sum_{x=+1,x=-1} \int_{-\infty}^{+\infty} p(L^{E1}/X = x) log_2 \left(\frac{2p(L^{E1}/X = x)}{p(L^{E1}/X = -1) + p(L^{E1}/X = +1)} \right) dL^{E1} \tag{6.14}$$

Viewing I^{E1} as a function of I^{A1} and E_b/N_0 values, the extrinsic information transfer characteristics of the detector are defined as:

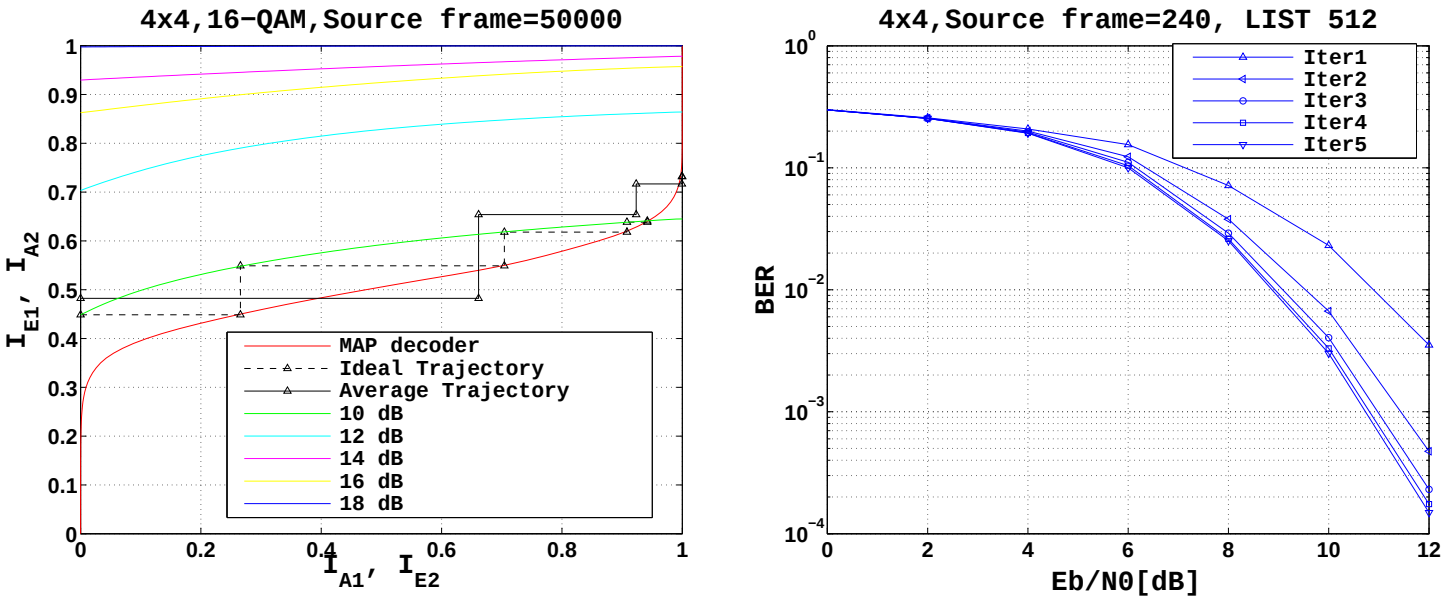
$$I^{E1} = T_1(I^{A1}, E_b/N_0) \tag{6.15}$$

which can be computed for a desired $(I^{A1}, E_b/N_0)$ combination, by means of Monte Carlo simulations. The independent Gaussian random variable of eq. (6.8), where σ_{A1} is determined through eq. (6.13), is applied as a priori input to the detector.
Similarly, the extrinsic transfer characteristic of the decoder is:

$$I^{E2} = T_2(I^{A2}) \tag{6.16}$$

It describes the input/output relationship between the a priori input L^{A2} and the extrinsic output L^{E2} values. It is independent of the channel noise, and it can be computed assuming L^{A2} to be Gaussian distributed and applying same equations used for the characteristics of the detector T_1.
The detector and the decoder characteristics are then plotted into a single diagram, called Extrinsic Information Transfer Chart. In the following graphics, the decoder characteristic is always represented by a solid red line, while other coloured lines are detector characteristics for different E_b/N_0 values. On the y-axis, the extrinsic information produced by the detector I^{E1} becomes the a priori input for the decoder I^{A2}. On the x-axis, instead, the extrinsic output of the decoder I^{E2} becomes the a priori input of the detector I^{A1}. In this way, the so called *ideal* or *snapshot trajectory* , indicated with a solid black line, approximately describes the true behavior of the iterative system. While the *ideal trajectory* is obtained from a detection-decoding of a single block of bits, the *average trajectory* (dashed black line) traces the average behavior over a certain number of blocks. The considered system is a 4×4-MIMO, with a 16-QAM modulation, code rate 0.5, a MAP decoder and Gray mapping. The model of the channel is considered as block-fading for this analysis. Each EXIT chart is obtained for an interleaver size of 10^5 bits and finite list length. The numerical method to apply EXIT chart has been derived from [77].

Since the divergence phenomenon has to be studied here, the first method of handling the list has been chosen, i.e. the list obtained at the first iteration

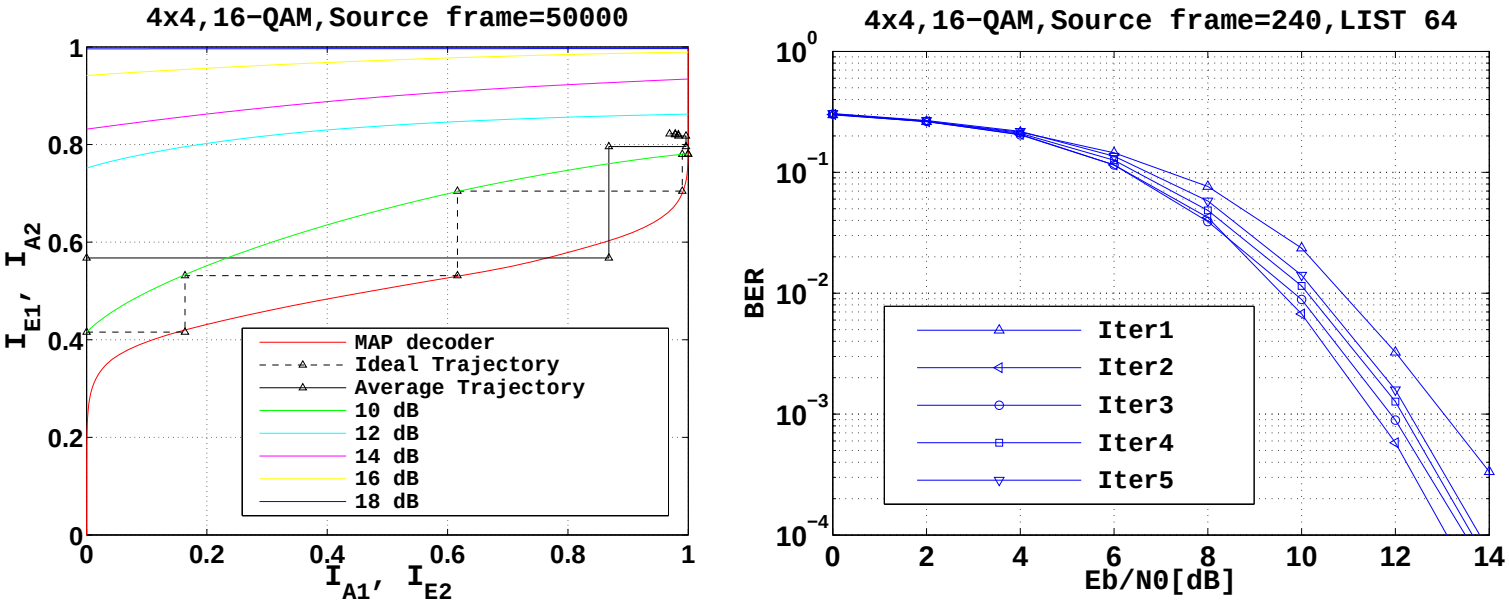


(a) (b)

Figure 6.2: EXIT charts and BER performance for LSD and MAP decoder with different SNR values and a list size of 512 (a,b).

is also used for all subsequent iterations. In particular, the original algorithm of LSD in [23], with LLR clipping value of ± 8, has been adopted, in order to analyze the effect of a reduction of the size of the list. The Fig 6.2(a) shows that for list size of 512 at $E_b/N_0 = 10dB$ the ideal trajectory gets stuck after 4 iterations. This means that the system is able to converge in 4 iterations. Fig. 6.2(b), instead, shows the BER for the same system, modulation and list size, but for an interleaver size of 480 bits (frame size of 240 bits), which is a more realistic case, applied by wireless communications standards, such as WiMax Standard. Similarly to EXIT Chart, also BER curves show that performance is not improved with more than four iterations. The *average trajectory* in Fig. 6.2(a) is obtained from a "free-running" iterative detection-decoding of 1000 different blocks of 480 bits. The difference with respect to the ideal trajectory is evident [25], due to the very smaller interleaver size (480 bits) with respect to the ideal case (10^5 bits), but this decoding also gets stuck after 4 iterations. The same conclusion can be drawn from BER curves. Then it can be concluded that a list of 512 candidates for a 16-QAM constellation is a reliable choice, which results into a decoder that

improves performance with iterations.



(a) (b)

Figure 6.3: EXIT charts and BER performance for LSD and MAP decoder with different SNR values and a list size of 64 (a,b).

Now, a reduction of the list size is applied: results obtained with a list size equal to 64 are shown in Fig. 6.3. With $E_b/N_0 = 10dB$, the effect of a short list is to open the tunnel between detector and decoder, in particular increasing the mutual information for low BERs. The ideal trajectory has about 3 iterations, however Fig. 6.3(b) shows that, with an interleaver size of 480 bits, performance gets worse after the second iteration. This phenomenon is also visible in the average trajectory of Fig. 6.3(a): after the second iteration, even if the mutual information at the output of the LSD slightly increases, the decoder is not able to improve the results, and gets worse. The reason of this behavior is mainly the unreliability of the list: smaller the list, higher the probability that one of the two sets $\mathcal{L}_k^{-1}$, $\mathcal{L}_k^{+1}$ is empty, higher the number of times you need to fix LLRs to the clipping value. When the detector sets the output LLRs to the fixed clipping level, it cannot take into account feedback soft inputs coming from the channel decoder and its behavior reduces to a hard-output detector [74]. Moreover, when the list size is reduced, the LLR distribution at the decoder output is no more

Gaussian, as seen in Fig. 6.4. This explains why the EXIT Chart does not match with BER Chart, since the hypothesis of Gaussian distribution of soft information fails.

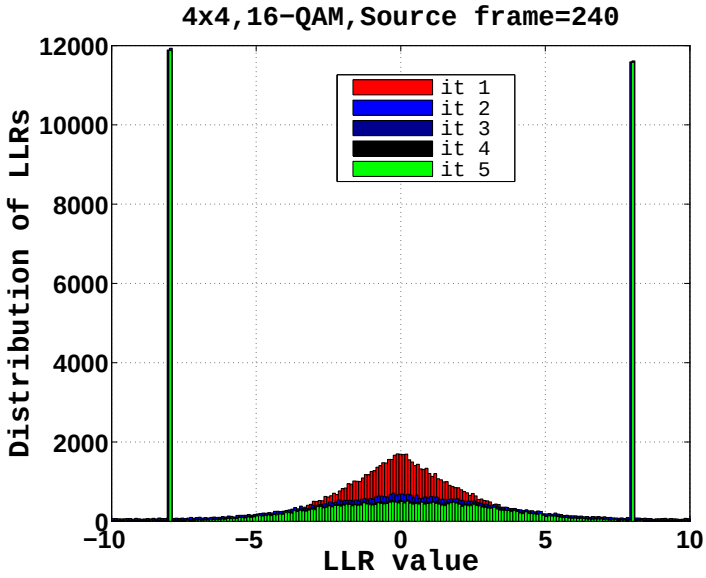


Figure 6.4: Distribution of LLRs at the output of the LSD (16-QAM, 4×4-MIMO, list size of 64, block-fading channel model and $E_b/N_0 = 10dB$).

After a wide research in literature, only [10] and [24] has been found, which highlight the problem of divergence. In particular, only solution in [10] details the employed method in order to avoid the divergence: it is based on constraining the maximum allowable value of *a priori* information $\mathbf{L}^A$, coming from the decoder, to prevent the value of $\mathbf{x}_{[k]}^T \mathbf{L}_{[k]}^A$ from growing too large. The maximum value is chosen as:

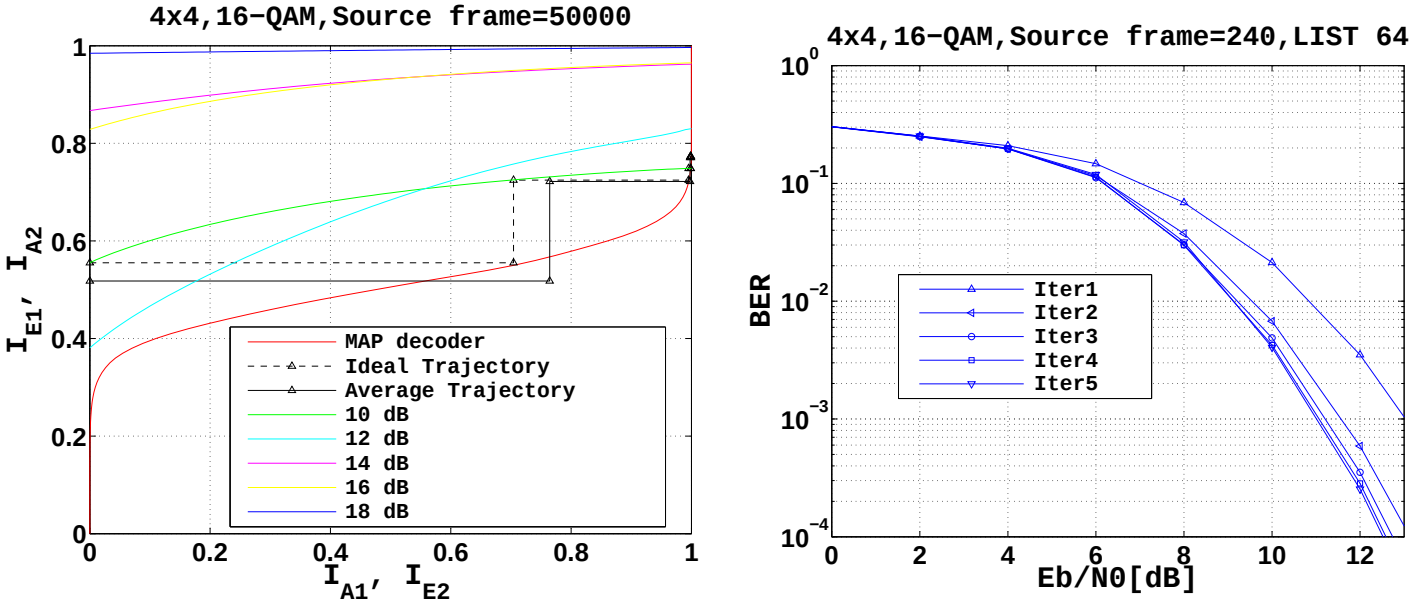
$$L_{max}^A = \frac{r}{min(1, k_{\mathcal{O}}\sigma^2)} \tag{6.17}$$

where r is the radius of the LSD and $k_{\mathcal{O}} = M_t M_r E_{avr}$. E_{avr} is the average energy of the modulation (2 for Q-PSK, 10 for 16-QAM and 42 for 64-QAM). The rule for constraining the *a priori* information is the following:

$$L^{AC} = \begin{cases} L^A & max\{L^A\} \leq L_{max}^A \\ \frac{L_{max}^A}{max\{L^A\}} L^A & max\{L^A\} > L_{max}^A \end{cases} \tag{6.18}$$

Moreover, an LLR clipping value of ±50 has been chosen [10]. Results obtained when this solution is applied to the case of 16-QAM and a list of 64 candidates,

are shown in Fig. 6.5. The effect of constraining *a priori* LLRs is, in general, a reduction and a modification of the original mutual information (see Fig.6.5(a)), and since an empirical and conditional scaling is applied, the original transfer functions of the detector is modified (the curve corresponding to $E_b/N_0 = 12dB$ is below the one corresponding to $E_b/N_0 = 10dB$ for $I_{A1} < 0.6$). No divergence appears and at $E_b/N_0 = 10dB$ both average and ideal trajectories get stuck after 3 iterations, as furthermore pointed out by the BER chart in Fig. 6.5(b).



(a) (b)

Figure 6.5: EXIT chart and BER performance for LSD and MAP decoder with different SNR values and a list size of 64 (a,b) with the proposed solution in [10].

6.3 Comparisons between LSD and MMSE-IC

Adopting the solution in [10], it is interesting to explore the performance of LSD for different modulations and systems, also compared to a low-complexity suboptimal algorithm, such as the Minimum-Mean-Square-Error Interference-Canceller (MMSE-IC) equalizer [78][79]. As mentioned in section 6.1 both fast and block flat fading channel models are considered. The setup for the following simulations is shown in Table 6.1.

MIMO system	4×4, 2×2
Modulation	Q-PSK,16-QAM, 64-QAM
Mapping	Gray
Channel model	Rayleigh fading, block and fast
List size for block-fading	64
List size for fast-fading	256, 512
Channel encoder	Convolutional (rate 0.5)
Channel decoder	Max-log MAP BCJR
Number of iterations	5
Coded frame	480 bits

Table 6.1: Setup of the conducted simulations

6.3.1 Block Fading Channel

In the case of block-fading, the channel matrix is constant over a frame, here a coded frame of 480 bits is considered. As Fig. 6.6 shows, no significant improvement is achieved after two LSD iterations. Moreover LSD is able to outperform MMSE in two iterations, also with a moderate list size, such as 64 candidates. Regarding MMSE a significant performance degradation appears for high SNRs, due to a matrix inversion issue in the eq. (6.7) [80]. This matrix has two contributes: one represented by the channel matrix and the other by the variance of the noise. For high SNRs the main contribution comes from the channel matrix $\mathbf{H}$. Thus a bad realization of $\mathbf{H}$ causes the above matrix to be singular, which results in a wrong computation of equalization coefficients. This will impact the whole frame in case of block-fading scenario.

6.3.2 Fast Fading Channel

In the case of fast-fading channel, the matrix inversion issue in MMSE, described in the previous section, has much less impact (Fig. 6.7(a)(b)) because a bad channel matrix realization affects one symbol vector rather than the complete frame. Regarding LSD, simulations have shown that the proposed solution [10] is not able to eliminate definitively the unreliability of a small list. In the fast-fading, with a list of 64, a performance degradation, compared to MMSE, happens and the divergence appears early (since the third iteration). This forces to increase

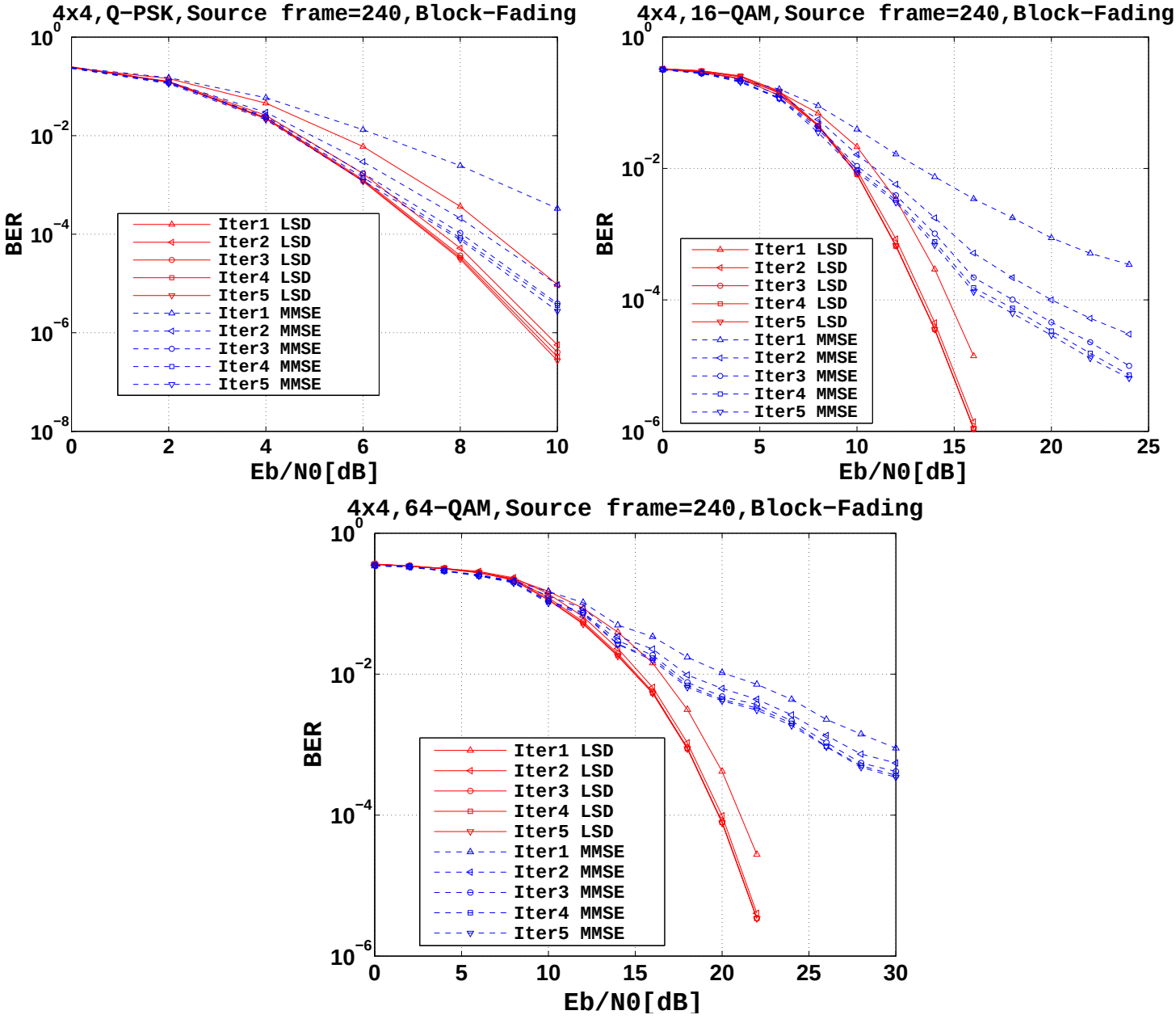


Figure 6.6: BER comparison between LSD (list size 64) and MMSE for different modulations and block-fading channel.

the size of the list, in order to improve BER performance and avoid the divergence with iterations. So, for the QPSK the size of the list has been increased to 256, and for 16QAM and 64QAM to 512.

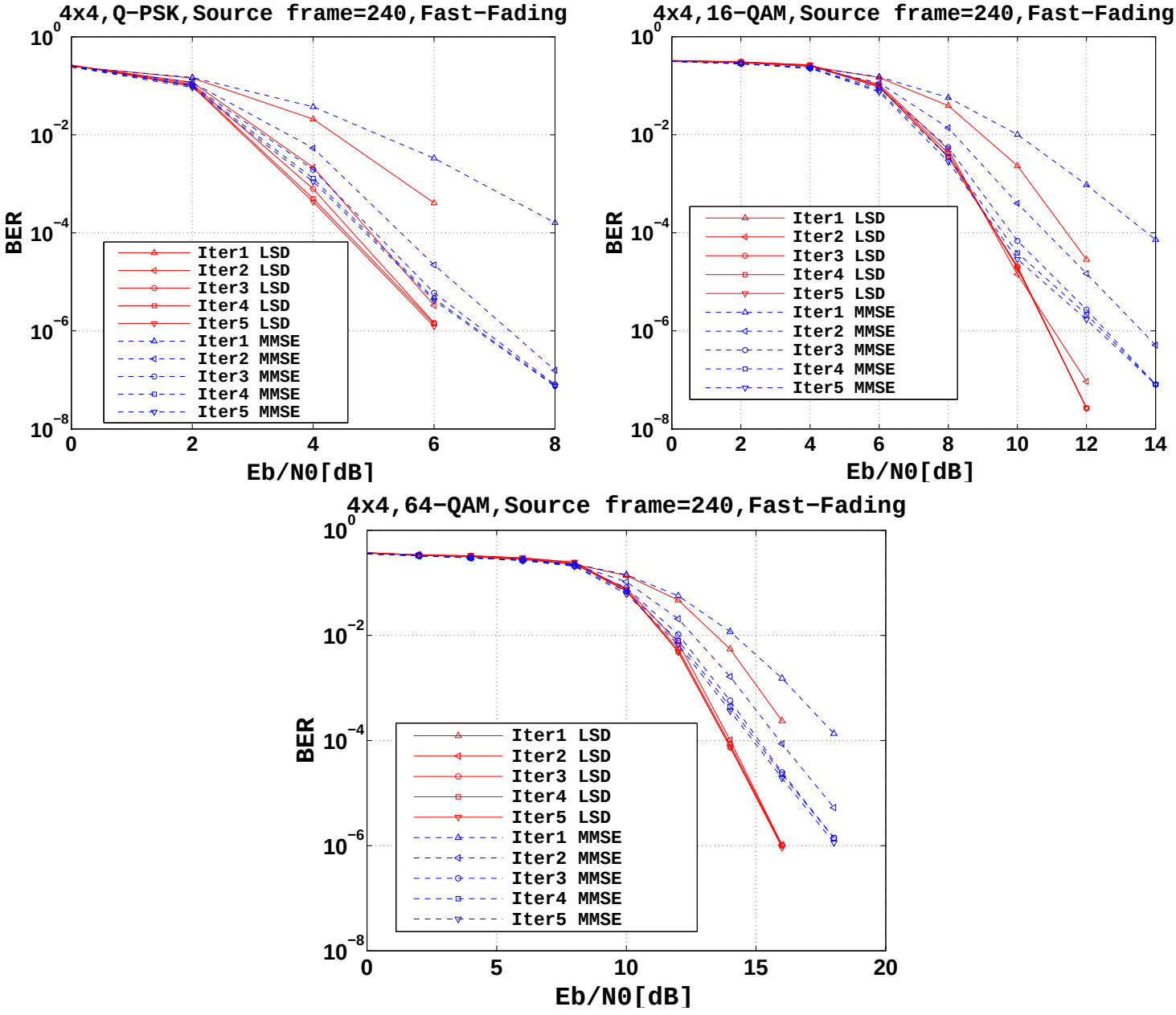
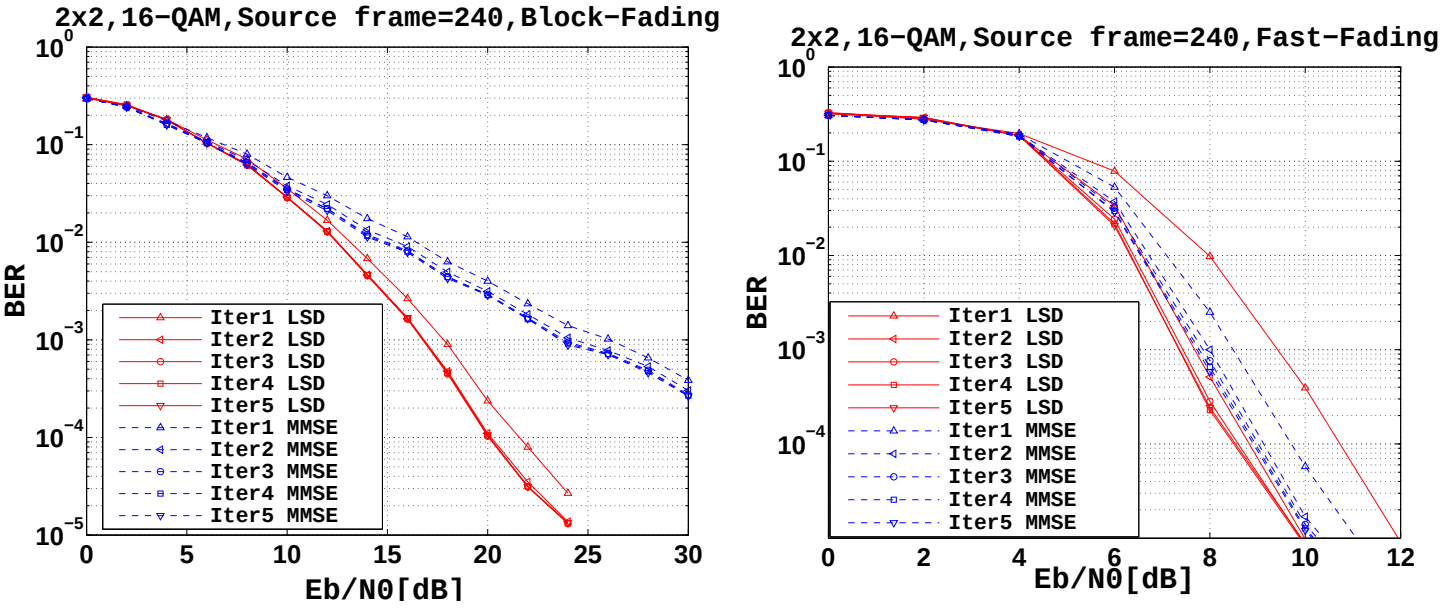


Figure 6.7: BER comparison between LSD (list size 256 for QPSK and 512 for 16QAM and 64QAM) and MMSE for different modulations and fast-fading channel.

6.3.3 Block and Fast Fading Channel for a 2×2-MIMO system

Considering flexibility requirements, it is important to see what happens also for a different number of antennas. Fig. 6.8(a)(b) shows results for 2×2-MIMO and 16-QAM, both for block and fast-fading. As before, also for this case, LSD has better performance for a block-fading channel. Although a list size of 64 is large enough for a 2×2 system, with a fast-fading channel, the achieved BER after two LSD iterations is comparable with that obtained with five MMSE iterations. We can then say that LSD is more adaptable than a linear detector, such as MMSE-IC, since changing the size of the list and the "LLR clipping" value, different performance-complexity trade-offs can be achieved. For what concerns the channel model, in case of fast-fading, for small constellations and system configurations, MMSE has similar performance to LSD. On the other hand, LSD has better BER performance than MMSE when the estimation of the channel conditions remain constant over a frame, like in the block-fading case.



(a) (b)

Figure 6.8: BER comparison between LSD (list size 64) and MMSE for 2×2-MIMO, 16-QAM, block (a) and fast-fading channel (b).

6.3.4 Complexity comparison

This section reports results on the number of operations required for the detection of a frame, in the block fading scenario. Table 6.2 shows the number of real additions/subtractions and multiplications for detecting an encoded frame of 480 bits, in the 4×4-MIMO case with 16-QAM. QR-decomposition for LSD and matrix inversion for MMSE-IC are not considered, so that these results are independent of the model of the channel. The complexity of MMSE-IC is fixed, with assigned antenna number and modulation, instead, for LSD it is variable with the SNR and Table 6.2 shows the average operation number, obtained after 1000 simulations for SNR=16dB (equivalent to a BER of 10^{-6}).
The LSD is mainly composed of two units: list generation, performed only in the first iteration, and LLRs computation, performed at each iteration. Similarly, MMSE-IC includes two main processing tasks: the computation of symbol expectations, which is performed according to eq. (6.4), and the evaluation at each iteration of soft mapper and demapper outputs. As shown in [81], the average complexity for an hard SD is $O(M_t^3)$. Since the LSD not only finds the ML solution but a list of candidates, it is claimed that the complexity of list generation grows also linearly with the size of the list, as $O(M_t^3 L)$. Moreover the complexity of LLR computation is $O(M_t^2 Q^2 L)$ [26] for one iteration. The average complexity for MMSE-IC is reported in [26] as $2O(M_t Q 2^Q) + O(M_t^3)$ for each iteration. These complexities are for a vector of M_t symbols, so they must be multiplied for the total number of vectors in the frame (here 30 vectors). As already known [26], LSD complexity is 10 times higher than MMSE. However LSD complexity can be reduced with a smaller list and adjusted clipping level. Considering, for example, the list generation for LSD with a list size of 64 and the symbol expectation for 5 iterations for MMSE-IC, the former has a number of additions/subtractions seven times higher than the latter. Even the difference between the LLR computation and soft mapper and demapper is of two orders of magnitude.

MMSE-IC	ADD/SUB	MUL
Symbol expectation 1 it	6.7e+03	2.9e+03
Symbol expectation 5 it	3.4e+04	1.4e+04
Soft Demapper+ Soft Mapper 1 it	1.3e+03	1.4e+03
Soft Demapper+ Soft Mapper 5 it	6.6e+03	7.2e+03
LSD	ADD/SUB	MUL
List generation	2.5e+05	2.3e+05
LLR comp. 1 it	3.7e+05	3.1e+04
LLR comp. 2 it	7.4e+05	6.3e+04

Table 6.2: Complexity estimation with a 4×4-MIMO and 16-QAM for LSD (list of 64, BER of 10^{-6}) and MMSE-IC

6.4 Discussion of the results

The iterative behavior of LSD detector has been analyzed using EXIT charts. If the *a priori* LLRs are properly constrained, the divergence problem can be mitigated with a relatively small list size. This last solution has been compared with a linear detector, such as MMSE-IC, in terms of performance and complexity, both for block and fast-fading channel models. In case of fast-fading, for small constellations and system configurations, MMSE-IC has similar performance to LSD. On the other hand, LSD has better BER performance than MMSE-IC when the estimation of the channel conditions remain constant over a frame. Although complexity of LSD is significantly higher than that of MMSE-IC, LSD outperforms MMSE-IC in two iterations. Solution in [82] shows how it is possible to reduce the complexity of the LSD to one half of a traditional LSD, thanks to Sorted QR-decomposition, MMSE preprocessing and Tuple Search. The performed analysis on possible complexity-performance trade-offs can be exploited to extend flexibility of current sphere decoders [3] and design a multi-mode flexible MIMO detector supporting both MMSE-IC and LSD with variable list size. In such a flexible detector, selection between the executed algorithm and choice of the main parameters should be based on channel conditions and targeted performance.

7

ASIP implementation of LSD

In chapters 5 and 6, the idea of a multi-mode flexible MIMO detector supporting both MMSE-IC and LSD has been extensively presented and investigated. This chapter, instead, focuses on the implementation aspects of such a kind of detector, whose main feature is the flexibility. Nowadays, one of the most flexible kind of implementation is certainly the Application-Specific- Instruction set-Processor (ASIP) [83] [84] [85] [86] [87].

An ASIP is a programmable microprocessor, where hardware and instruction set are designed together for one special application. It has the main characteristic of combining performance and energy efficiency of dedicated hardware solutions with the flexibility of a programmable solution. Anyway, other popular approaches for the implementation of specific functions in digital systems are exploited. The requirement of very high performance can be satisfied with an Application-Specific-Integrated-Circuits (ASICs) or even a physically optimized ICs, sacrificing completely the flexibility. On the contrary, programmable solutions, such as General-Purpose Processors (GPP), offer more flexibility and faster time-to-market. If some additional processing power is needed, there are specific processors, optimized for signal processing, such as Digital Signal Processors (DSPs), which offer some additional specialized instructions (e.g. Multiply-Accumulate, MAC). More flexibility is, instead, satisfied by programmable devices, like Field Programmable Gate Arrays (FPGAs), which are reconfigurable, at the price in terms of performance, power and cost, making them suitable in particular for prototyping. Among all these alternatives, ASIPs are the best

trade-off between performance and flexibility, and represent an intermediate solution between DSPs and FPGAs. Their instruction sets tailored to the application allow faster processing, while their programmability offers the flexibility needed to adapt to changing requirements. MIMO detection is one of the many domains, which can benefit from application-specific processors.

Several options are available to design an ASIP: it can be completely described through an Architecture Description Language (ADL), or obtained after a software programming of an existing processor, or using a so called transport triggered architecture (TTA). Next section provides a description of an ASIP design flow, accompanied by an overview of methodologies and tools.

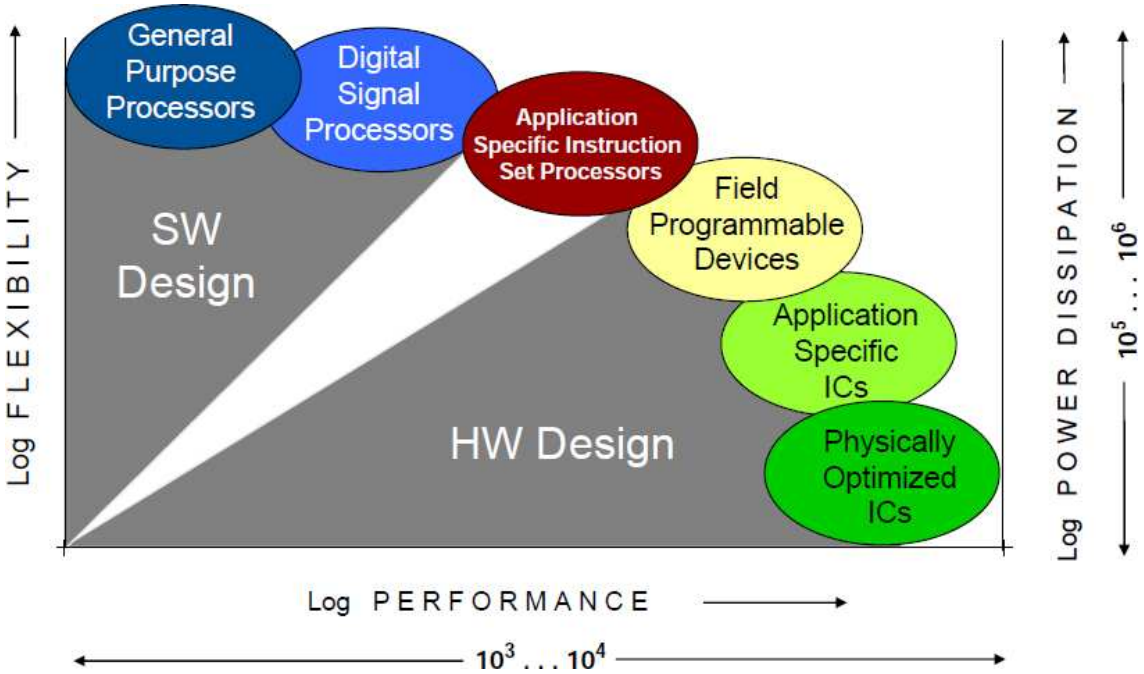


Figure 7.1: A complexity-flexibility trade-offs for different approaches for implementation in digital systems [11].

7.1 ASIP design flow

The most popular ASIP design technologies can be divided in four main categories [88]:

- Architecture Description Language (ADL) based solutions: they are also known as ASIP-from-scratch, because of the accessibility to each detail of the architecture. Stages of pipeline, number and type of resources and memory structures are completely specified by the designer, through an high-level language. Some example of tools, which adopt this approach, are the CoWare Processor Designer [89] and Target IP Designer [90]. This method has the highest flexibility and efficiency, but also requires a significant design effort.
- Configurable processor core based solutions: in this case, it is possible to add custom Instruction Set Extension (ISE) to a pre-defined and pre-verified core, so obtaining more efficiency. This approach has certainly less flexibility than ADL based solutions. Tensilica Xtensa [91] and ARC ARChitect [92] are two examples of such a kind of technology. Moreover also transport triggered architectures (TTAs) are included in this category. TTA is a programmable architecture, where processor is a made of independent function units and register files, which are connected with transport buses and sockets [93]. In conventional processor architectures, data transports are consequences of operations . In this way, the data transmission may become the bottle-neck of implementations, where high amount of data is processed. In TTAs, the situation is reversed and operations are consequences of data transports. In TTA design, the processor designer has free hands to built the optimal data transmission by adding enough buses between logic and memory. It is then programmed simply by defining the sources and destinations of these transports. The TTA-based Codesign Environment (TCE) [94] is an example of tool, employed to design TTAs.
- Software reconfigurable processors: like Stretch [95] and ADRES [96]. They present a fixed hardware architecture with a reconfigurable ISE, which can be modified to customize instructions. This approach has the most restricting degree of freedom with respect to previous solutions.

Independently of the employed technology, the tipical ASIP design flow starts with the identication of the so called “hot spots” [97] of the application and with the definition of an initial architecture and a custom instruction set, both able to

efficiently support those hot spots. Then the application is run on the processor and the designer verifies if the target requirements are met. If specifications are not satisfied, the flow is iterated.
Next subsection provides an overview of the ASIP design flow for ADL based solutions.

7.1.1 An ADL based tool: Coware Processor Designer

The ADL based method is the most powerful among all other above approaches, since the processor can be completely specified in all its parts, through an high-level language. Such a language is more abstract than hardware description languages, such as VHDL of Verilog. This description is accepted by a specialized tool that automatically generates both synthesizable Register Transfer Level (RTL) and software descriptions. Many ADLs have been introduced through the years, like nML [98], based on a mixed structural and behavioral model, Sim-nML [99], an extended version of the nML formalism, Instruction Set Description Language (ISDL) [100] and Language for Instruction-Set Architecture (LISA) [101] [102].

In this research thesis, a tool, namely CoWare Processor Designer [103], has been employed to design an ASIP for LSD. CoWare Processor Designer is a widespread commercial ASIP design environment, which is based on LISA [85]. Processor Designer is used to develop a wide range of processor architectures, including architectures with DSP-specific and RISC-specific features as well as SIMD and VLIW architectures. Moreover, processors with complex pipelines can be easily modeled. This includes the ability to describe architectures with complex execution schemes.
The main features of LISA are [83]:

1. to provide cycle-accurate processor models, specifying stages of pipeline and mechanisms, like stalls, flushes, reset and operation injection;
2. extension of the target class of processors including single instruction multiple data, VLIW, and super-scalar architectures of real-world processor architectures;

3. different levels of abstraction in the behavioral descriptions of parts of the process model: detailed bit-true description for simulation and implementation and assignment to arithmetical function for the instruction selection task of the compiler are distincted.
4. C/C++ based: LISA includes pure C/C++ behavioral operation description;
5. complex instruction coding schemes and instruction aliasing supported.

LISA descriptions are composed of *resources* and *operations*. The *resources* are the storage objects: registers, memories and pipelines. The *operations*, instead, are the basic objects in LISA. They represent the designers view of the behavior, the structure, and the instruction set of the programmable architecture. Moreover, the high degree of automation of Processor Designer reduces the time for developing the software tool suite and hardware implementation of the processor: in this way, designers can mainly focus on architecture exploration and development. The usage of a centralized description of the processor architecture ensures the consistency of the Instruction-Set Simulator (ISS), software development tools (compiler, assembler, and linker) and RTL implementation, minimizing the verification and debug effort.
The LISA machine description provides information consisting of the following model components [83]:

- *Memory model*: lists the registers and memories of the system with their respective bit widths, ranges, and aliasing. The compiler gets information on available registers and memory spaces. The memory configuration is provided to perform object code linking. During simulation, the entirety of storage elements represents the state of the processor, which can be displayed in the debugger. The HDL code generator derives the basic architecture structure.
- *Resource model*: describes the available hardware resources and the resource requirements of operations. Resources reflect properties of hardware structures that can be accessed exclusively by one operation at a time. The instruction scheduling of the compiler depends on this information. The HDL code generator uses this information for resource conflict resolution.

- *Instruction-set model*: identifies valid combinations of hardware operations and admissible operands. It is expressed by the assembly syntax, instruction-word coding, and the specification of legal operands and addressing modes for each instruction. Compilers and assemblers can identify instructions based on this model. The same information is used at the reverse process of decoding and disassembling.
- *Behavioral model*: it abstracts the activities of hardware structures to operations changing the state of the processor for simulation purposes. The abstraction level of this model can range widely between the hardware implementation level and the level of high-level language (HLL) statements.
- *Timing model*: it specifies the activation sequence of hardware operations and units. The instruction latency information lets the compiler find an appropriate schedule and provides timing relations between operations for simulation and implementation.
- *Microarchitecture model*: it allows grouping of hardware operations to functional units and contains the exact micro-architecture implementation of structural components such as adders, multipliers, etc. This enables the HDL generator to generate the appropriate HDL code from a more abstract specification.

After describing the architecture, using these above models, the Processor Designer generates a synthesizable HDL representation and the complete software tool suite automatically. The design flow starts from a LISA 2.0 description, which allows to specify all components, such as stages of pipeline, memories, register files, instructions and pins. Then Processor Designer automatically generates both ISS, complete software tool suite (C-compiler, assembler and linker) and the RTL implementation description (Verilog HDL, VHDL and System C). Through the simulator, the application can be run and debugged, verifying the processor model and evaluating the performance. If the specifications are not met, the flow comes back to the LISA description, since the design flow is a closed-loop, as shown in Fig. 7.2. When the design goals are completely satisfied, the HDL implementation can be synthesized, evaluating occupied Silicon area, clock frequency and power consumption.

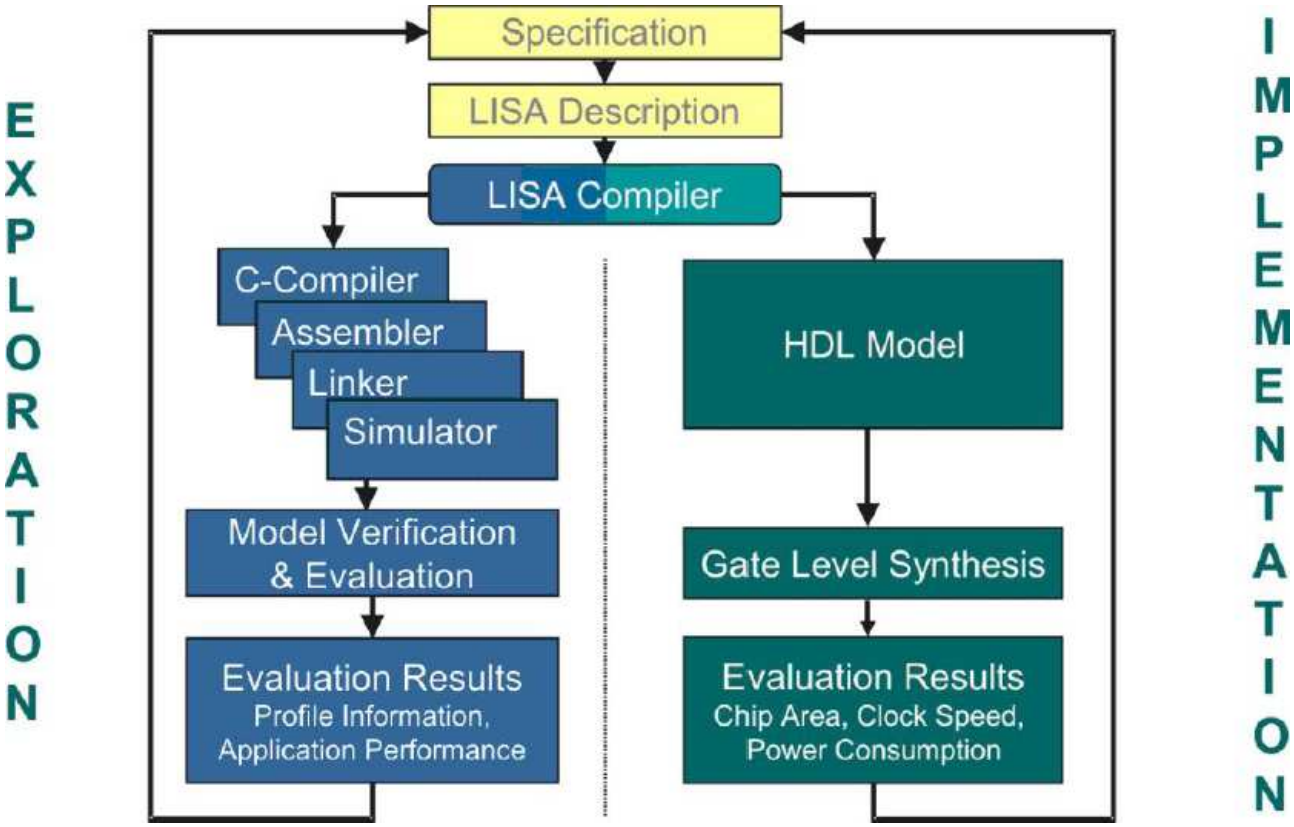


Figure 7.2: LISA design flow [11].

7.2 First suboptimal ASIP of LSD

As already described in chapter 6, the LSD is composed of two main blocks: list generation, performed only in the first iteration, and LLRs computation, performed at each iteration. Chosen the system and the list size, the former has the aim of finding the $|\mathcal{L}|$ candidates, which are nearest to the received signal vector. The number of clock cycles needed to fulfill this object is variable, depending on SNR value. The LLRs computation unit has to compute LLRs, based on the list: so this activity takes a constant number of clocks. Therefore, the complete architecture of LSD has to reflect this structure. It is composed of two ASIPs: the first devoted to list generation and the second to LLR computation. The rest of this thesis reports a detailed description of the ASIP1 (list generation), while the design of the ASIP2 (LLR computation) has been left to future works.

7.2.1 Flexibility parameters and architectural choices

Fig. 7.3 shows the high-level architecture of ASIP1, where memories, register files and input parameters are indicated. The considered flexibility parameters are three:

- **Number of antennas**: indicated as M in Fig. 7.3. Through this parameter, the user can choose the number of transmitters and receivers. Here, the supported values are 2, 3 and 4, which are respectively for 2×2, 3×3 and 4×4 MIMO systems.
  It is worth notifying that if if the number of transmitters and receivers is different, the square channel matrix can be obtained, using a mathematical trick [14]. This is applied in the QR-decomposition, which is not included in the ASIP design.
- **Type of modulation**: indicated as *PAM_size*. Since a real–valued system has been employed (see section 2.2), this parameter represents the number of PAM symbols, corresponding to a QPSK, 16QAM and 64QAM. So possible choices are 2, 4 and 8. In fact, QPSK has {-1,1}, as corresponding PAM symbols, the 16QAM has {-3,-1,1,3}, and the 64QAM {-7,-5,-3,-1,1,3,5,7}.
- **List size**: indicated as *list_size*. Currently, the ASIP1 supports a number of candidates up to 64, which is an acceptable range, taking into account that the main problem of LSD is memory requirement due to list, and that a list of 64 is a reliable choice for all the previous constellations, in particular in the block fading scenario (see chapter 6).

Anyway, list and modulation sizes can be easily extended to other values, different from those above indicated, just with minimal modifications, mainly related to the size of memories.

The *resources* of the processor are composed of one data memory and seven register files. The $2M \times (2M + 1)$ *Channel Data Mem* stores $\mathbf{R}$ matrix coefficients and ZF solutions, $\tilde{\mathbf{y}}$, obtained after QR-decomposition. When the ASIP1 starts working, the content of the memory is load into the two corresponding register files (*R RF* and *PSI RF*), in order to be directly read by the processor. During the detection, the *PSI RF* is also used to save intermediate results, together with

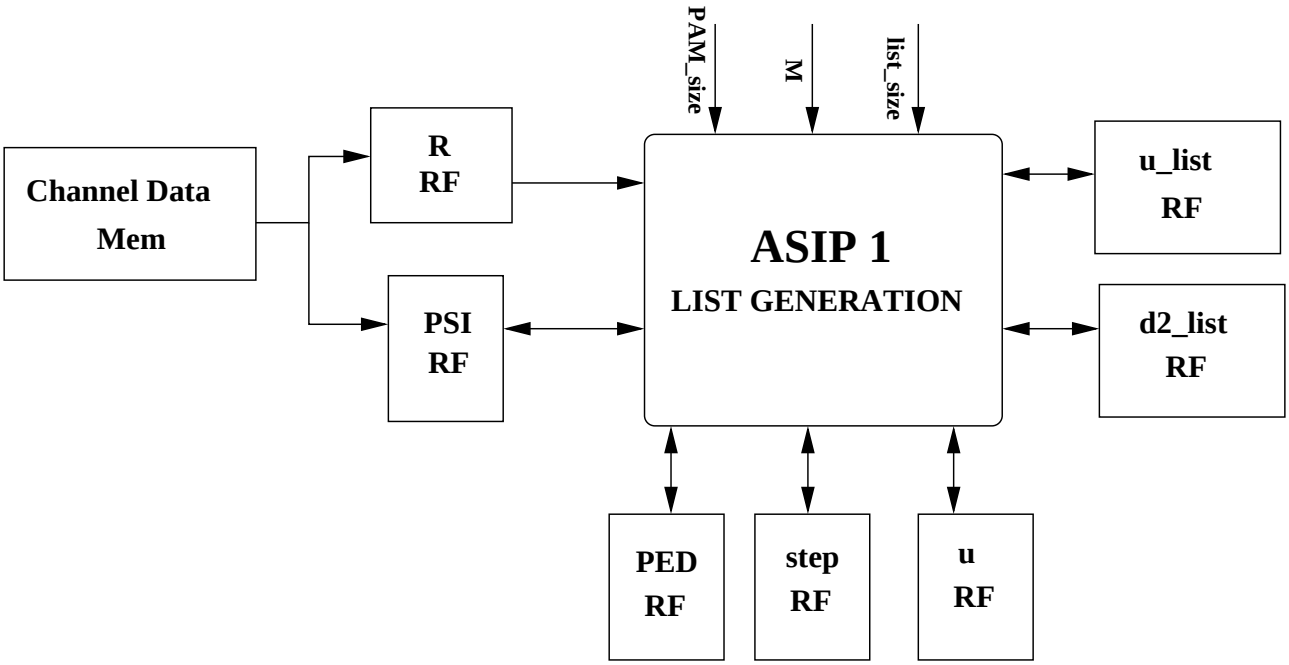


Figure 7.3: High level architecture of ASIP1.

other tree register files, namely *PED RF*, *step RF* and *u RF*. They contain the current visited symbol (*u RF*) and related SE initial direction(*step RF*), partial euclidean distance (*PED RF*) and ψ metric (*PSI RF*). All of them have $2M$ locations, while *R RF* has $2M \times 2M$. Finally, *u_list RF* and *d2_list RF* respectively contain the list of candidates and the related square distances. $list\_size \times 2M$ locations are necessary for the former, while only $list\_size$ for the latter.

The processor has to fulfill five main tasks:

- Babai point selection
- PED computation
- ψ computation
- SE enumeration
- List management

Hardware architecture of each of these processing units is described in the following sections. As for the implementation of SDA and LASDA, the internal

precision has been set to 16 bits, half of them used for the integer part and the other half for the fractional part. Some data are represented with less than 16-bits, as specified in the next paragraphs, and in order to enable the reuse of hardware resources, they are sign extended to 16-bit format. Moreover, a 32 to 16-bit conversion is performed at the outputs of the multipliers and to avoid overflow or underflow problems, outputs are superior or inferior limited.

7.2.1.1 Babai Point selection

This unit has the role of selecting the Babai Point, in the forward processing, according to eq. (2.14). The complexity , in terms of number of resources, depends only on the cardinality of the modulation, not on the number of antennas. It is quite similar to S block, shown in Fig. 3.13, but extended to cover also the 64QAM modulation. This is a classical divisor, implemented with left shift operations and subtractions, as detailed in Fig. 7.4. It takes $\psi^{l+1}[l]$ and $R_{l,l}$, as inputs, and computes the Babai Point, indicated as *res*, and the initial direction of SE enumeration, indicated as *step*. If *step* is equal to 1, the next chosen point will be the closest symbol at right, otherwise the closest at left. Apart from inputs, represented on 16 bits, the Babai point can be set to 4 bits, with a two's complement representation, which is the maximum precision required for the worst case, i.e. 64QAM.
Alternatively the Babai Point selection can be implemented, according to eq. (2.15), as a network of subtractors, multipliers and comparators, as shown in Fig. 7.5. This architecture implies to consider all symbols of the constellation. In this way, the number of resources is directly proportional to the cardinality of the modulation. Fig. 7.6 reports synthesis results of both architectures using Synopsys Design Compiler version Z-2007.03-SP1. A 0.13 μm CMOS Standard Cell technology is adopted. The occupied Silicon area is reported on y-axis, while the x-axis shows delays, from about 4.1 ns to the minimum critical path obtained, for each one. A quick look at those results, allow to state that the ad-hoc divisor has better performance than the alternative architecture, both in terms of area and in terms of clock frequency.

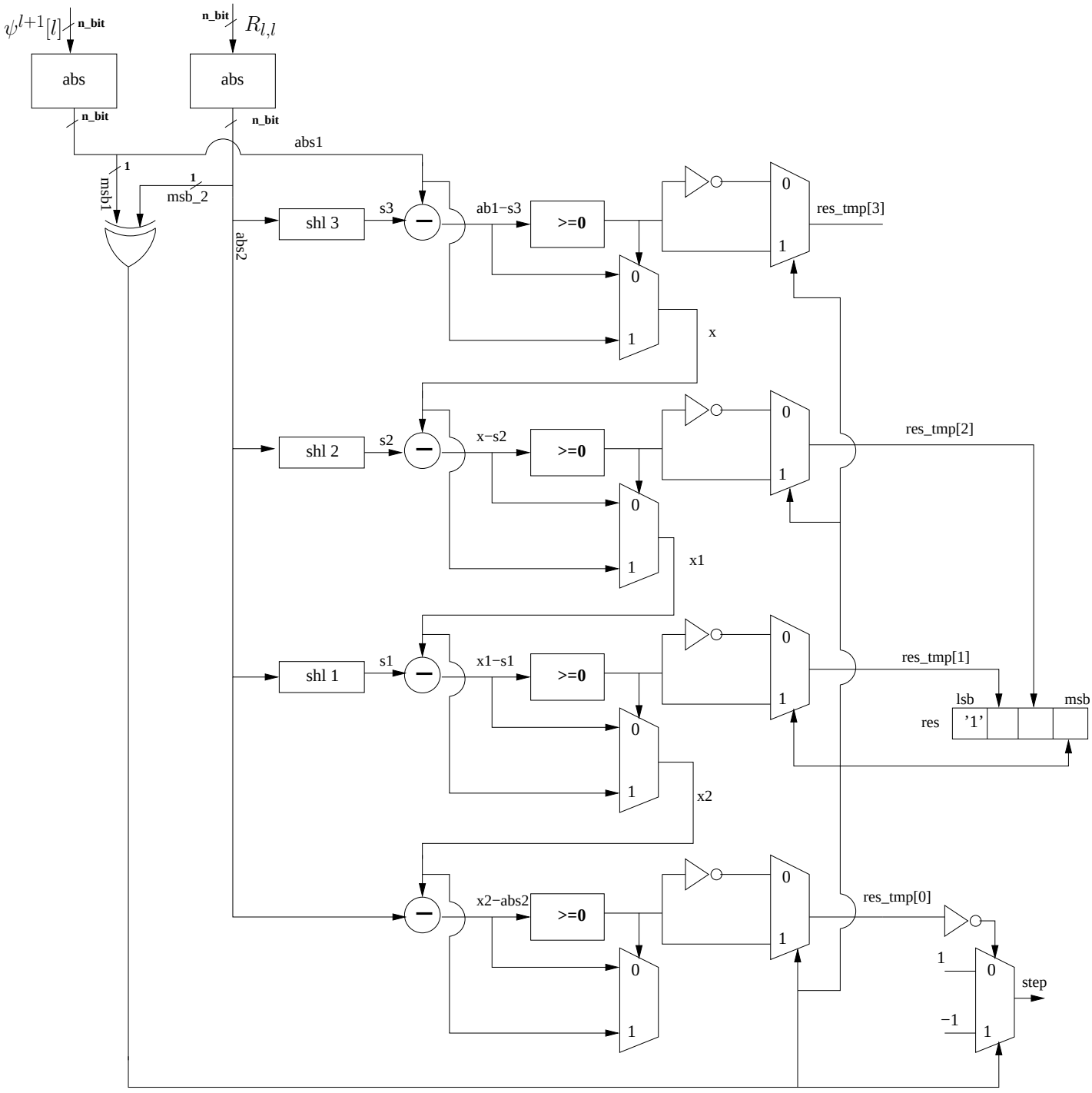


Figure 7.4: Scheme of the Babai selection unit.

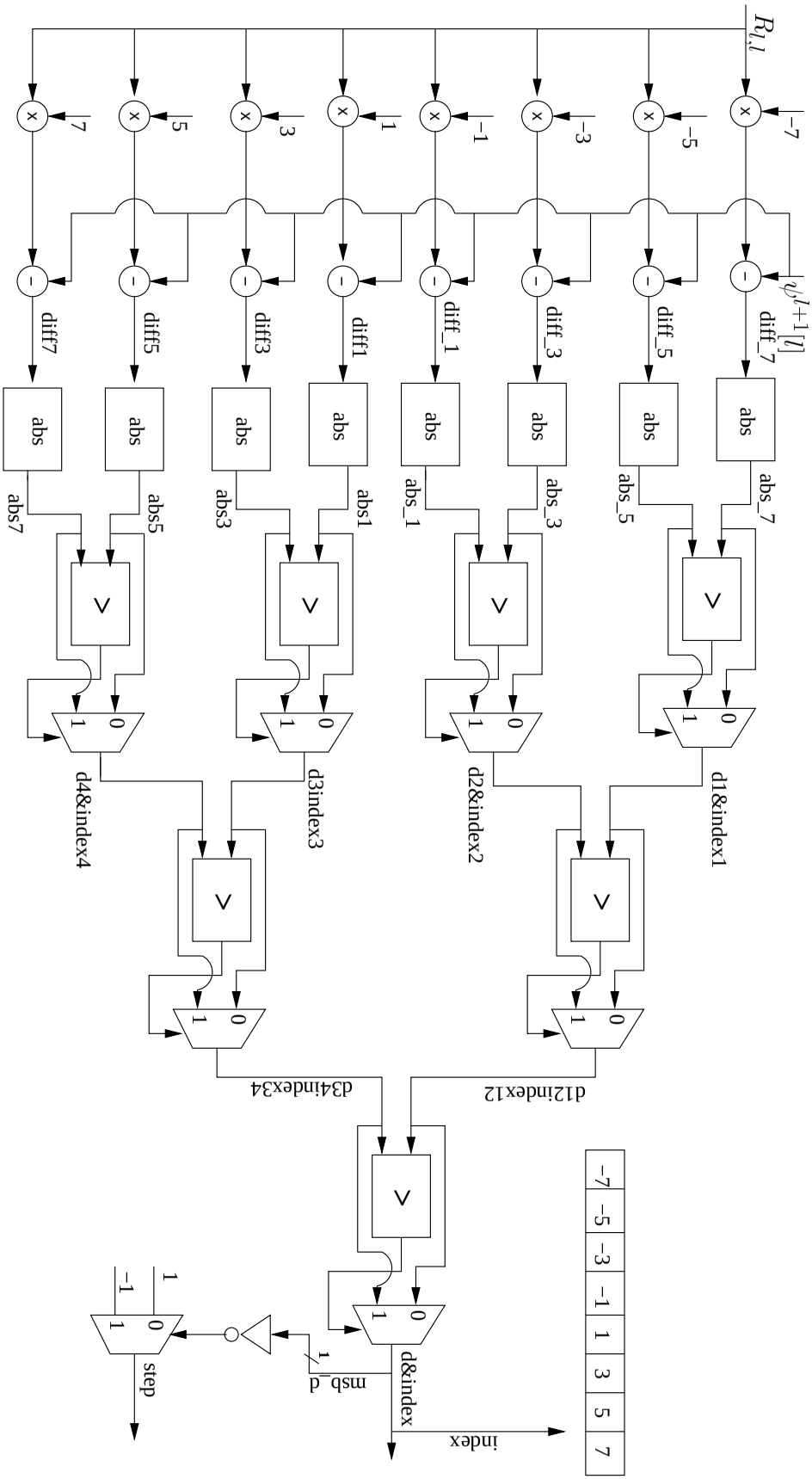


Figure 7.5: Alternative architecture for the Babai Point selection unit.

7.2.1.2 PED computation

This unit has to compute the partial euclidean distance of the Babai Point, during the forward processing; during the backward processing, it computes the alternative points, based on SE enumeration, (eq. (2.8)). It receives s_l, $R_{l,l}$, $\psi^{l+1}[l]$ and T_{l+1}, as inputs and produces the current partial euclidean distance T_l. It is indipendent of both cardinality of constellation and number of antennas, making this block quite simple, composed only of two multipliers, one subtractor and one adder. It is worth clarifying here that the two multipliers do not have the

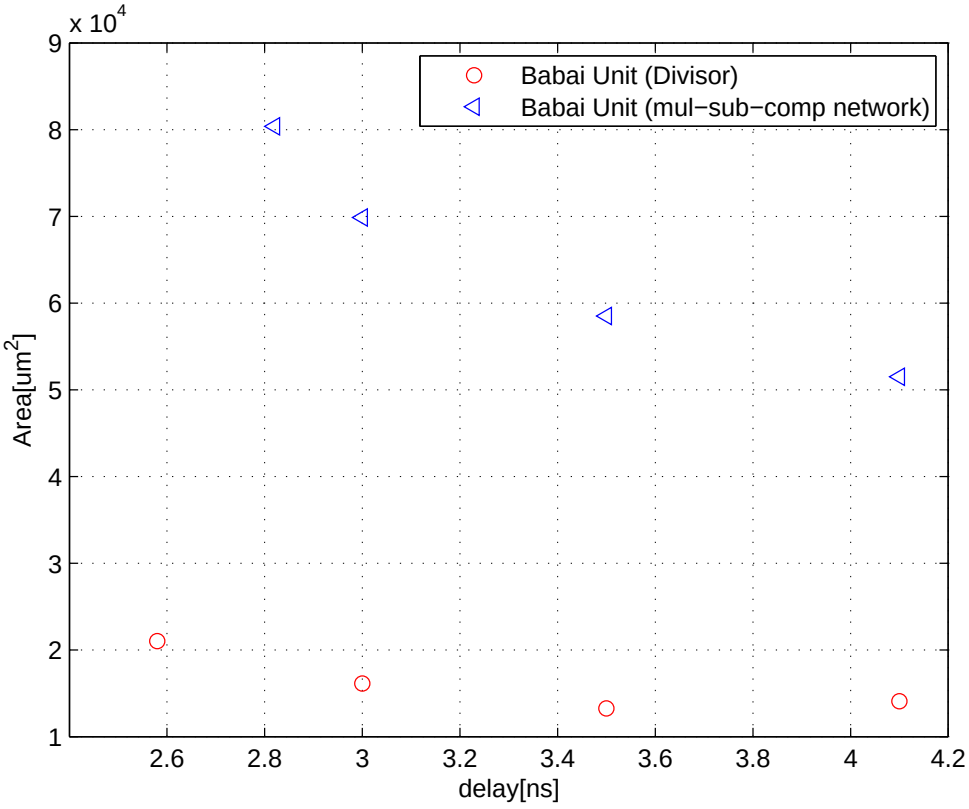


Figure 7.6: Area-delay graph for two implementations of the Babai Point selection.

same complexity, since the precision of their inputs is different. In particular, the multiplication at the bottom of Fig. 7.7(a) is more complex than that at the top, since it involves two 16-bits signals, obtaining a results on 32-bits, to be brought

back to 16 bits.

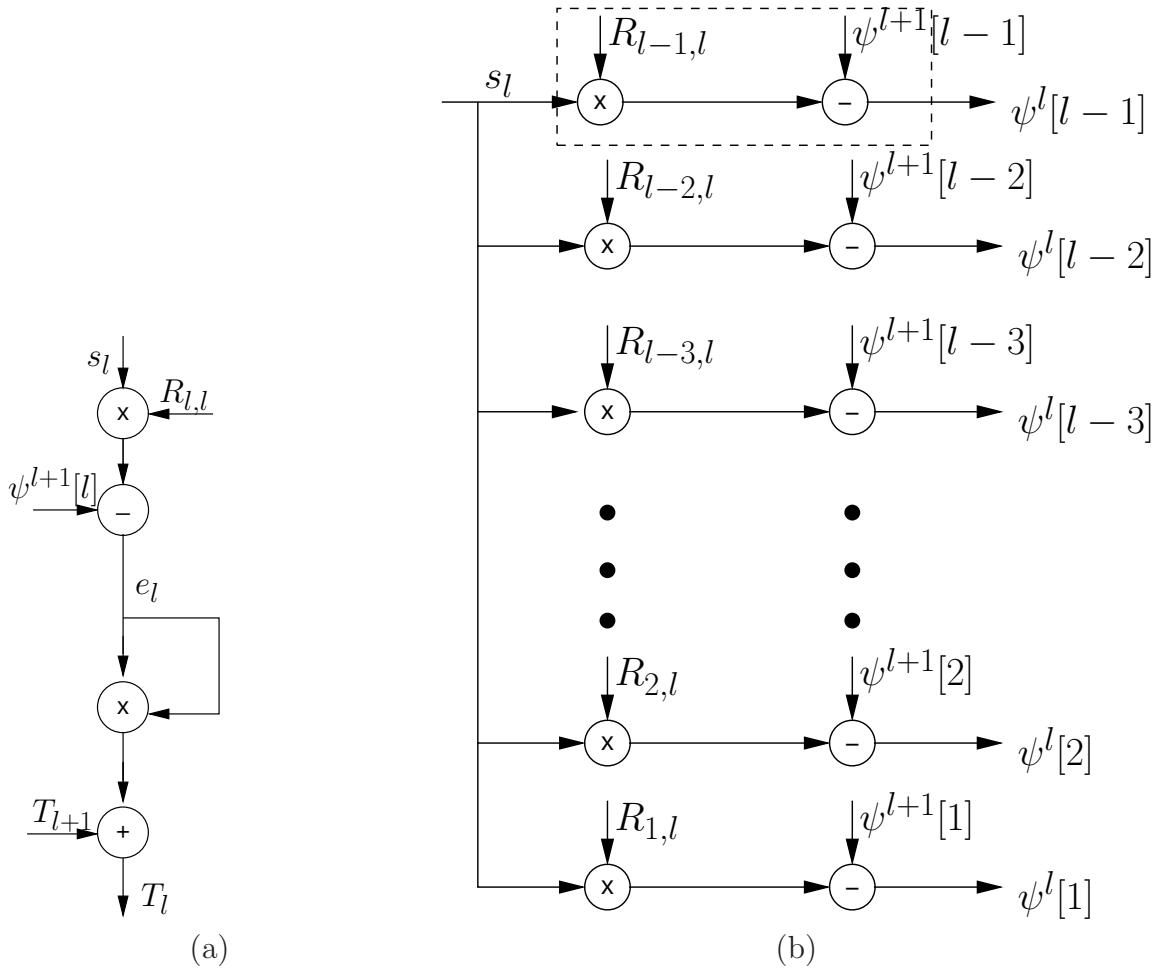


Figure 7.7: Schemes of the PED computation (a) and ψ computation (b) units.

7.2.1.3 ψ computation

The aim of this block consists in computing $\psi^l[i]$ amounts that will be required at the lower tree levels, according to eq. (3.9). At each level l of the tree, it receives the current visited symbol s_l, $R_{i,l}$ and $\psi^{l+1}[i]$ elements for $i = l-1, \ldots, 1$ and gives $\psi^l[i]$ as results. The basic unit is composed by a multiplier and an adder, which are allocated a number of times equal to $l-1$. The complexity of this block is therefore dependent on number of antennas, but not on the cardinality of the constellation. The necessary hardware to be allocated is specified by the

worst case: for a 4×4 system the root level of the tree is $l = 7$ and the maximum number of basic units, working in parallel, is equal to 6. For smaller systems, some units will not be employed.

7.2.1.4 SE enumeration

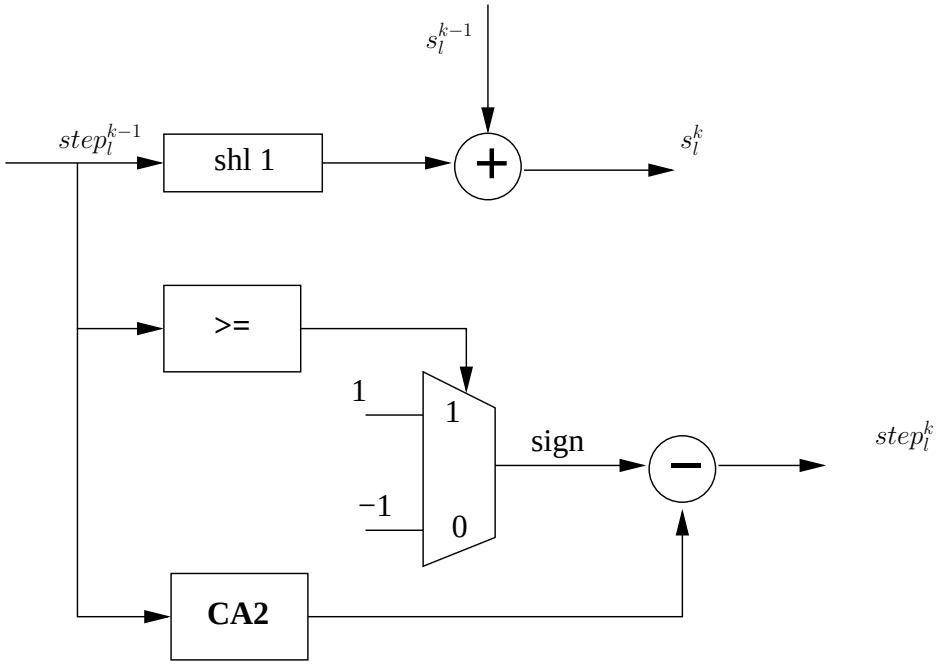


Figure 7.8: Scheme of the SE enumeration unit.

This unit has to compute the next alternative symbols, according to SE enumeration:

$$\begin{cases} s_l^k = s_l^{k-1} + 2step_l^{k-1} \\ step_l^k = -step_l^{k-1} - sign(step_l^{k-1}) \end{cases} \qquad k > 1 \tag{7.1}$$

As shown in Fig. 7.8, it is a quite simple unit, composed of a multiplexer, an adder, a subtractor, a comparator, a left shift and a two's complement operator. The number of resources to be allocated is indipendent of number of antennas and cardinality of the constellation. Instead, the range of values that s_l^k and $step_l^k$ can assume, clearly depends on the symbols, belonging to the modulation.

7.2.1.5 List management

One of the main issues of LSD is the management of the list, not only in terms of memory requirement, but also regarding how to handle it. If a large list size is preferred, the sorting and storing of symbol vectors quickly becomes the bottleneck of the algorithm. Two main approaches are exploited to maintain the list: memory-based [104] and register-based. The former provides a more energy-efficient but slower solution to the problem, while the latter guarantees an highly optimized and faster hardware, even if it tends to have an high energy consumption. Here the register-based approched has been adopted, since it will provide less energy-efficient, but certainly faster solution to the problem [104].

The LSD is a generalization of the SD. Rather than finding only the best argument, if finds the best $|\mathcal{L}|$ ones. It stores each of these arguments u_i and their corresponding value $d2_i$ in a list $\mathcal{L}_i = \{u_i, d2_i\}$ for $i = 1 \dots |\mathcal{L}|$. Each time a possible argument is found, the LSD checks whether it is better than all arguments in the list, and if so it exchanges them. This search requires an order of $|\mathcal{L}|$ comparisons, and is executed for every vector checked inside the sphere. In particular, each visited leaf, which satisfies the radius constraint, has to be inserted. When the detection starts, the initial radius is set to infinity, i.e. the maximum representable value with the desired precision, and only when the list will be full it will be updated to the biggest ED stored into the list. For this reason, it is useful to maintain the list ordered. In fact, in the ascending order, the first candidate has the smallest ED, corresponding to the ML solution, and the last has the biggest, corresponding to the radius. Later on, if a new leaf has a smaller distance with respect to the radius, it must be inserted, the biggest candidate is eliminated and the radius updated to the new largest element in the list.
Shortly, for each candidate found, three activities can be identified:

1. Position finder: it consists in finding the right position in the list;
2. Insertion: it implies to right shift all next candidates, with respect to the current position of the new argument, and to insert the new one. In this way, the last element is automatically eliminated;

3. Radius update: the radius is updated to the PED of the last element in the list.

As the latency of inserting a new symbol to the list is very crucial for the overall performance of the LSD algorithm, an efficient data structure is needed for sorting and storing of symbols. Heap data structure [105] has been suggested for LSD in [106] and [107] as an efficient algorithmic solution. The work in [104], shows an ASIP implementations of a K-best memory-based LSD, with detailed explanation of the heap utilization. Heap is an efficient choice for long lists: comparator networks for constructing binary heaps, presented in [108], have a complexity of order $O(|\mathcal{L}|log_2log_2|\mathcal{L}|)$, in terms of number of comparators, and $O(log_2|\mathcal{L}|)$, in terms of delay.

The bubble sorting algorithm is more suitable for K-best approaches, since it is shown that the cost of sorting is linear with the value of K [109] [110], and can be employed in a sequential fashion [15]. Since the K-best is a single-direction searching algorithm, the sorting can take more than one clock to be completed. This is not the case of depth-first LSD, which is double-direction, and the list must be ordered in one clock.

A sorting network proposed in [111] has a depth of $O(log_2|\mathcal{L}|log_2log_2|\mathcal{L}|)$: it makes use of an operation called (m,k)-*sort-and-shuffle*, which operates on an array of m elements, by first sorting elements in each block of size k, and then shuffling the m/k sorted blocks together. The shuffling simply consists in moving each element from a position to another position of the array, according to a specific rule.

The quite complicate structure of this last solution, which also needs to apply a "random" permutation and the recursion [111], makes it more suitable for software application. The heap-sort-based approach is, on the contrary, a good hardware solution, mainly addressed to low-area requirements. But if the main purpose is high data-rate, and, therefore, high clock frequency, a tree-based network of comparators, like for the heap-sort, is probably not the best answer. In this first version of the ASIP, the very basic and straight-forward implementation of the list management unit, whose pseudocode is illustrated in Algorithm 4, has been adopted, as shown in Fig. 7.9. This highly parallel and very easy to implement architecture is very fast, at a price of an increased occupied area,

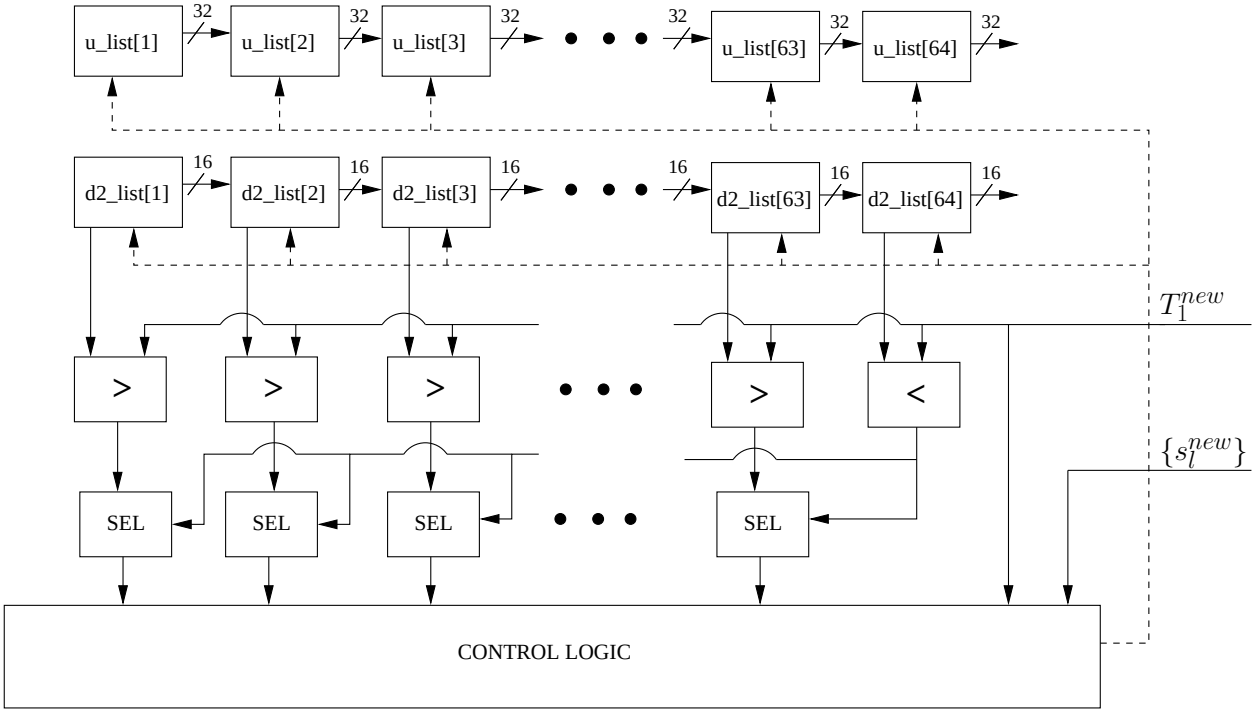


Figure 7.9: Scheme of the list management unit.

with respect to above described methods. This is pointed out comparing [104] and [112]: the former solution employs heap structure for the list, while the latter make use of our same architecture to manage the list. The list unit in [112] occupies 5400 GE on a 130 nm technology and 100 MHz of clock frequency, while the same unit, implemented with heap structure, occupies 2300 GE, for same technology and frequency. This means about 134% increase of area. On the other hand, the maximum clock frequency obtained [104] is 150 MHz, versus 280 MHz in [112] (86% more).

7.2.2 Instruction Set Architecture

The ASIP has 5 pipeline stages:

1. Instruction Fetch (ISTRF): it fetches the instruction from the program memory and the fetch address register is prepared for the next instruction, checking if there is a jump or not;

Algorithm 4 List management

if $T_1^{new} < d2\_list[|\mathcal{L}|]$ **then**
  if $T_1^{new} > d2\_list[|\mathcal{L}|-1]$ **then**
    $d2\_list[|\mathcal{L}|] := T_1^{new}$, $u\_list[|\mathcal{L}|] := \{s_l^{new}\}$ with $l = 1, \ldots, M$
  else if $T_1^{new} > d2\_list[|\mathcal{L}|-2]$ **then**
    $d2\_list[|\mathcal{L}|] := d2\_list[|\mathcal{L}|-1]$, $u\_list[|\mathcal{L}|] := u\_list[|\mathcal{L}|-1]$
    $d2\_list[|\mathcal{L}|-1] := T_1^{new}$, $u\_list[|\mathcal{L}|-1] := \{s_l^{new}\}$ with $l = 1, \ldots, M$
  else if $T_1^{new} > d2\_list[|\mathcal{L}|-3]$ **then**
    $d2\_list[|\mathcal{L}|] := d2\_list[|\mathcal{L}|-1]$, $u\_list[|\mathcal{L}|] := u\_list[|\mathcal{L}|-1]$
    $d2\_list[|\mathcal{L}|-1] := d2\_list[|\mathcal{L}|-2]$, $u\_list[|\mathcal{L}|-1] := u\_list[|\mathcal{L}|-2]$
    $d2\_list[|\mathcal{L}|-2] := T_1^{new}$, $u\_list[|\mathcal{L}|-2] := \{s_l^{new}\}$ with $l = 1, \ldots, M$
  else if $T_1^{new} > d2\_list[|\mathcal{L}|-4]$ **then**
    $d2\_list[|\mathcal{L}|] := d2\_list[|\mathcal{L}|-1]$, $u\_list[|\mathcal{L}|] := u\_list[|\mathcal{L}|-1]$
    $d2\_list[|\mathcal{L}|-1] := d2\_list[|\mathcal{L}|-2]$, $u\_list[|\mathcal{L}|-1] := u\_list[|\mathcal{L}|-2]$
    $d2\_list[|\mathcal{L}|-2] := d2\_list[|\mathcal{L}|-3]$, $u\_list[|\mathcal{L}|-2] := u\_list[|\mathcal{L}|-3]$
    $d2\_list[|\mathcal{L}|-3] := T_1^{new}$, $u\_list[|\mathcal{L}|-3] := \{s_l^{new}\}$ with $l = 1, \ldots, M$
    ⋮ ⋮ ⋮
    ⋮ ⋮ ⋮
  end if
end if

2. Operand Fetch (OPF): the instruction is decoded and the operands are fetched from register files to pipeline registers;
3. Execution 1 (EX1): this stage is devoted to execute arithmetic and logic operations, based on the specific instruction;
4. Memory/Execution 2 (EX2): it has been used with a double role: to load the content of data memory locations and also as a second execution stage for all other instructions, which do not directly communicate with memory;
5. Write Back (WB): results of the previous stage are written to register files.

The instruction set of the proposed ASIP is composed of two groups: the former includes some traditional instructions and the latter has specialized ones, which are properly designed for the LSD algorithm. The first group is composed of common instructions, such as LW, ADDI, NOP, BNE and JUMP. The LW ("load word") is used to load the content of *Channel Data Mem* into *R RF* and *PSI RF*. The instruction word is shown in Fig. 7.11. In particular, the

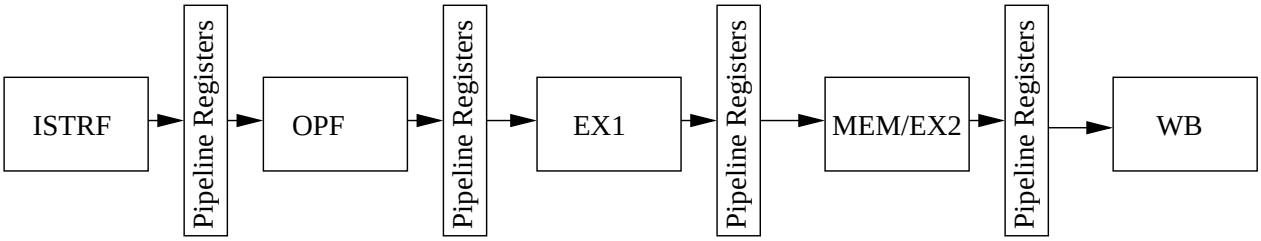


Figure 7.10: Stages of pipeline.

least significant bit is used to select either $R\ RF$ or $PSI\ RF$. The reg_dest and reg_adx fields represent the index for a general purpose register file, used as pointer to other RFs: the first is related to the index of $R\ RF$ or $PSI\ RF$, the second, instead is the relative address, adjusted with an immediate value imm, of the data memory. The ADDI ("add immediate") adds a constant value to

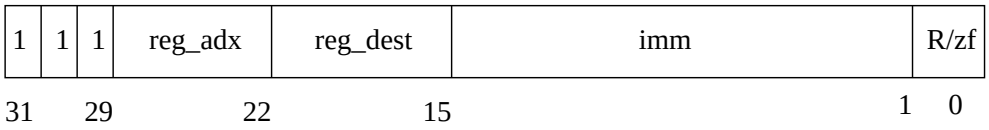


Figure 7.11: Instruction format of LW.

a relative address, while NOP instruction is employed when empty cycles are required during the execution of the program. The jumps are managed by the BNE ("branch if not equal") for conditional ones, used to check a condition if there is a loop into the program, and JUMP for those which are unconditional. In particular, BNE is useful together with LW, in order to read the specified number of locations from the data memory. If $reg1 \neq reg2$ in Fig. 7.12, the costant value $number$ is added to the program counter. The JUMP is, instead,

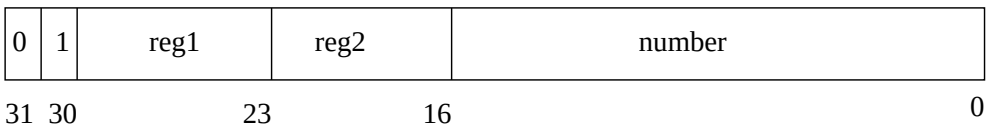


Figure 7.12: Instruction format of BNE.

employed since the average number of clock cycles, needed to detect a symbol vector, is variable with SNR value: in these case the main loop, which handles forward and backward processing of the tree in the LSD algorithm, is repeated

a number of times which is not fixed and not a priori predictable. The JUMP format is represented in Fig. 7.13. As it will be explained later, until the detection

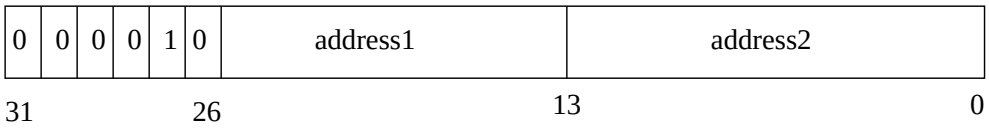


Figure 7.13: Instruction format of JUMP.

is not finished, the program jumps to *address*1, otherwise to *address*2.

The specialized instructions, which implement the tree search and synchronize the units presented in subsection 7.2.1, according to forward and backward processing are three: INIT, BABAI and CHECK. Next paragraphs contain a detailed description of each one.

7.2.2.1 INIT instruction

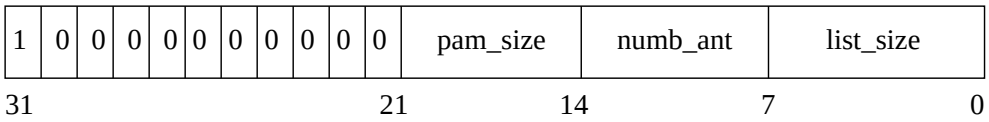


Figure 7.14: Instruction format of INIT.

This instruction is designed to initialize some internal registers of the ASIP. Through the fields *pam_size*, *numb_ant* and *list_size* in Fig. 7.14, the user can choose the desired system configuration. *pam_size* is the cardinality of the PAM constellation, and can assume values $\{2, 4, 8\}$, corresponding respectively to QPSK, 16QAM and 64QAM. The number of transmitters and receivers can be fixed with *numb_ant*, choosing among $\{2, 3, 4\}$, corresponding respectively to 2×2, 3×3 and 4×4 MIMO. Finally, the list can have a number of candidates from 1 to 64, thanks to the *list_size* field, which can assume integer values $\{0, 1, 2, \ldots 63\}$. Since the only operation done by INIT is simply to set some global register to a certain value, this can be done only in one stage of pipeline, the OPF. In this way, this instruction effectively occupies only the first two stages, and is completed in two clocks.

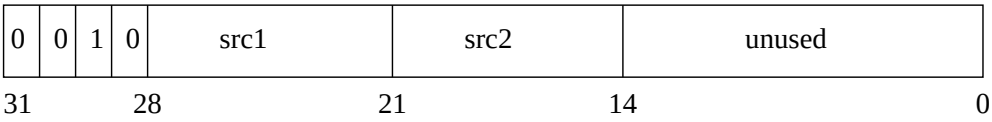


Figure 7.15: Instruction format of BABAI.

7.2.2.2 BABAI instruction

This instruction has been initially thought to accomplish the forward processing, which implies to select the Babai Point, and to compute related ψ and PED metrics. For this reason, fields $src1$ and $src2$ of the instruction format in Fig. 7.15 are respectively $\psi^{l+1}[l]$ and $R_{l,l}$, which are inputs of the Babai selection unit (see Fig. 7.4).

The whole processing has been scheduled in the stages of pipeline, as shown in Fig. 7.16. In the OPF stage, the instruction is decode and operands are load from register files to pipeline registers: an entire row of R RF ($R_{i,l}$ with $i = 1, \ldots, l$), one element T_{l+1} from PED RF, and one element $\psi^{l+1}[l]$ from PSI RF. In the EX1 stage, both Babai selection and ψ computation are performed, during the forward processing, while the PED computation is carried out in EX2. Finally, in the WB stage, results are written into RFs: T_l in PED RF, s_l and related $step_l$ respectively in u RF and $step$ RF, and all $\psi^l[i]$ with $i = 1, \ldots, l-1$ in PSI RF. The forward and backward processings have the common task of calculating the PED and ψ metrics for a symbol, which is the Babai Point during the former and another alternative sibling, according to SE enumeration, during the latter. In order to enable the resource sharing, the BABAI instruction can be used also in the backward processing, disabling the BABAI selection unit. In this case, the input symbol to the ψ computation unit comes from a pipeline register and it is generated by the CHECK instruction, as explained in the following paragraph.

The choice of performing the Babai selection and the computation of the ψ metrics in the same stage comes from data dependency. In the forward visiting, the Babai Point needs the proper ψ element to be computed, according to (2.14). The idea is to design an ASIP, which is not so far from a one-node-per-cycle architecture, also because LSD has an high and variable latency, due to the sequential nature of the algorithm itself.

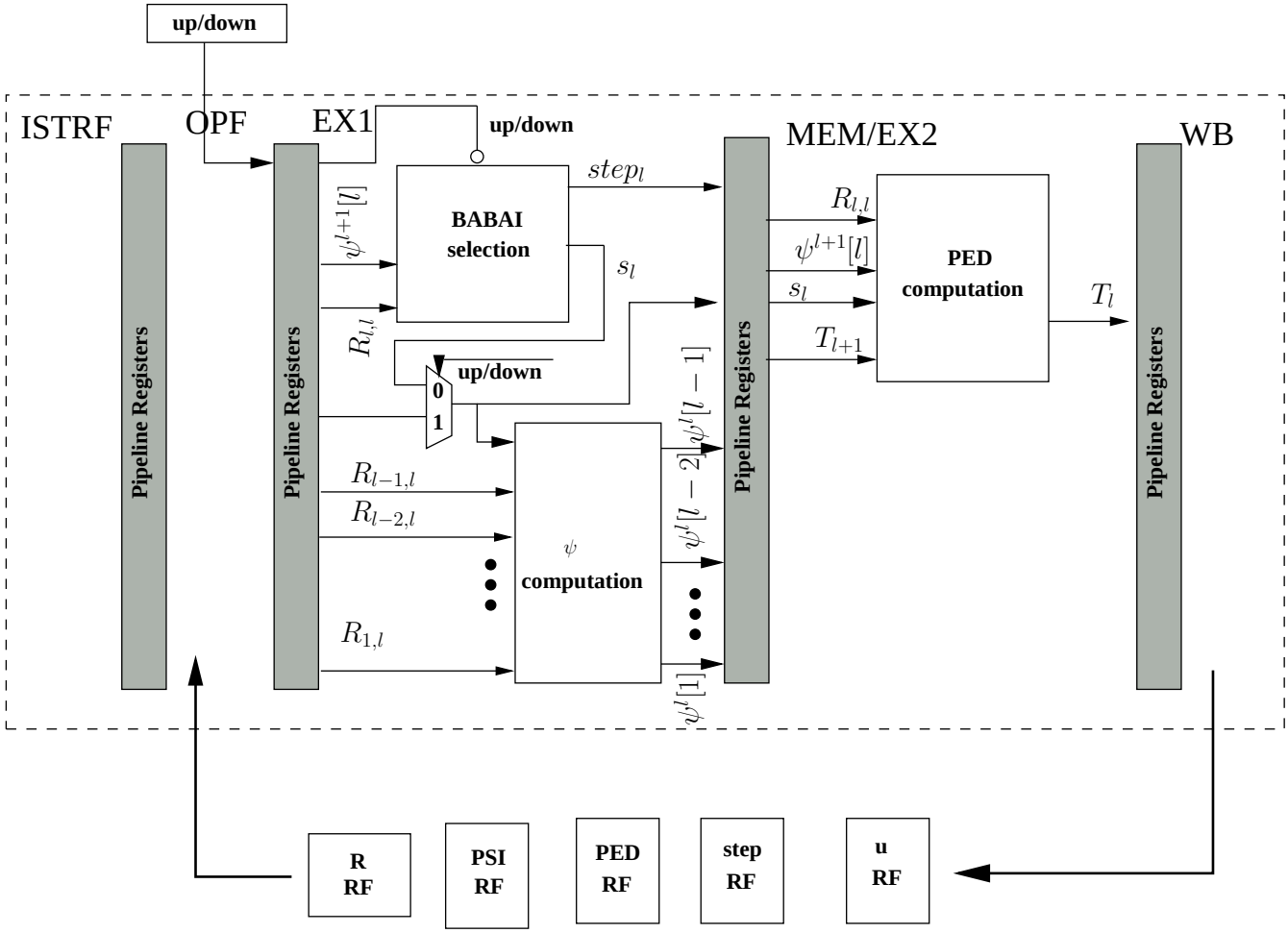


Figure 7.16: BABAI instruction and stages of pipeline.

7.2.2.3 CHECK instruction

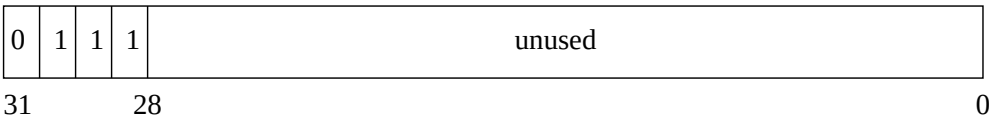


Figure 7.17: Instruction format of CHECK.

The CHECK instruction has the aim of managing the tree search: through the radius constraint verification, it can decide which will be next level of the tree to be visited. For these purpose, two global registers are allocated: *level* and *up/down*. As their names suggests, *level* contains the current level of the

tree, while *up/down* indicates the direction of the visit. Fig. 7.18 shows how the

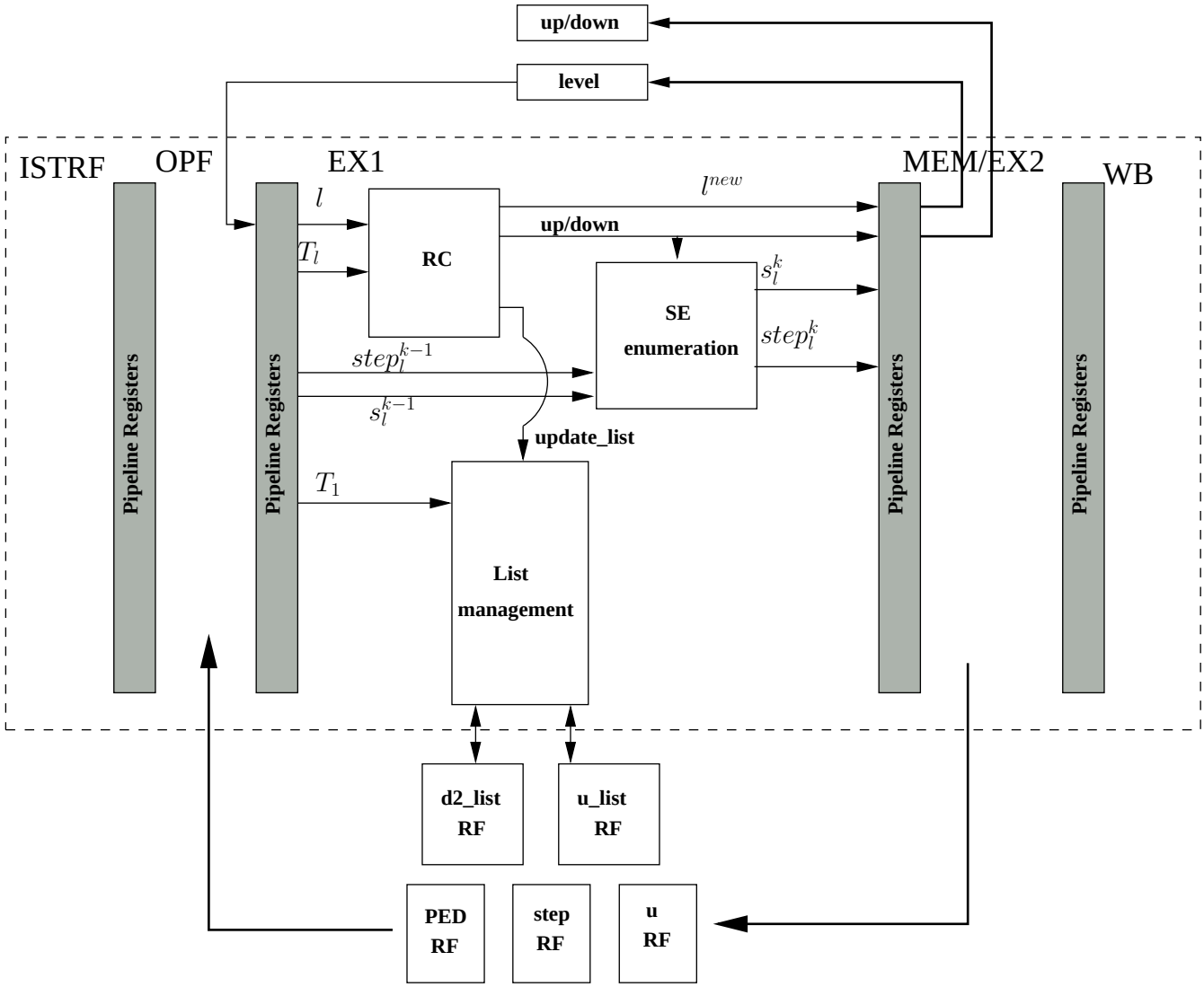


Figure 7.18: CHECK instruction and stages of pipeline.

CHECK instruction works along the pipeline. In the OPF, operands are loaded: one element T_l from $PED\ RF$, the current symbol and related *step* respectively from $u\ RF$ and $step\ RF$, and the current level of the tree from the *level* register. Then, all operations are concentrated in one clock. RC block verifies the radius constraint, according to (2.7) and, based on this, the *up/down* bit is set to 0 if the forward processing has to be accomplished, also enabling the BABAI selection in

the BABAI instruction, or to 1, activating the SE enumeration block otherwise. When the algorithm reaches a leaf, the *update_list* signal enables the list management unit, which directly reads and writes *d2_list RF* and *u_list RF*. Finally, in the MEM/EX2 stage, proper elements of *u RF* and *step RF* are updated, and l^{new} and *up/down* are written in the respective global registers.
As shown in Fig. 7.17, only opcode bits are used in the instruction format. Nothing else is necessary, since the addresses for reading RFs are directly derived from the content of *level* register.

7.2.3 Sample program

In this subsection sample program to implement the LSD algorithm is described. The assembly code line starting with ";" denotes a comment line.

The user can choose the desired configuration through the INIT instruction: this sample code is for a 4×4 MIMO, 64QAM and $|\mathcal{L}| = 64$. After the initialization of the ASIP, the first M locations of data memory, containing ZF solutions, are loaded into $PSI\ RF$, using a loop (line 4). Then the program loads the remaining $2M \times 2M$ locations, containing channel coefficients, into $R\ RF$ (loop at line 9). After that the LSD can first start to process the highest level of the tree, and continues the tree search jumping back to "check_up_down", until the detection is not complete, and to "finish" otherwise.

Listing 7.1: LSD assembly code

```
1  ;Initialization, choosing respectively pam_size,numb_ant and list_size
2         INIT $8 $4 $63
3  ;Load the first M locations of Channel Data Memory into PSI RF
4  load_zf:ADDI $0, $0 , 1
5         LW $0,0,$0 ,0
6         BNE $0, $1, load_zf
7         NOP
8  ;Load the successive 2M*2M locations of Channel Data Memory into R RF
9  load_R: ADDI $0, $0 , 1
10        ADDI $3, $3 , 1
11        LW $3,0,$0 ,1
12        BNE $3, $2, load_R
13 ;Compute Babai Point at the level M
14        BABAI $4,$5
15        NOP
16        NOP
17 ;Continue LSD untill it has not finished
18 check_up_down: CHECK
19                NOP
20                BABAI $4,$5
21                J check_up_down, finish
22                NOP
23                NOP
24 finish: NOP
```

7.2.4 Synthesis results

The Coware Processor Designer has been adopted to design the ASIP, describing the processor with LISA language. The tool provides software development suite (simulator, compiler, assembler, debugger and linker), so that, executing the application program, performance can be verified. Moreover, the VHDL code is automatically generated for hardware synthesis. The generated RTL description has been synthesized as ASIC, using Synopsys Design Compiler version Z-2007.03-SP1, on a 0.13 μm CMOS Standard Cell technology. In Table 7.1, the results of synthesis are summarized. The clock frequency is quite low. Before comparing

Table 7.1: Synthesis results of LSD ASIP

Technology	ST 130nm
Frequency	92 MHz
Area	816113 um^2 (151 KGE)

the proposed ASIP with the state of art implementations, next paragraph reports an improved design, faced to increase the clock frequency.

7.3 Improved ASIP: increased clock frequency

Analyzing the LSD in the Listing 7.1 (from line 14 to the end), it can be derived that the current application is able to visit one node per 6 clock cycles. This is due to the sequential nature of the algorithm, which implies data dependency between the results provided by the BABAI instruction and the radius verification of the CHECK. Therefore, the first modification that can be applied to the ASIP in order to increase clock frequency is to distribute the processing in a better way along the pipeline, just breaking each potential critical path. The main modifications involve BABAI and CHECK instructions, as Fig. 7.19 and 7.20 show :

1. BABAI: the PD computation can be moved in the MEM/EX2 stage;
2. CHECK: the list management can be shifted in the MEM/EX2 stage.

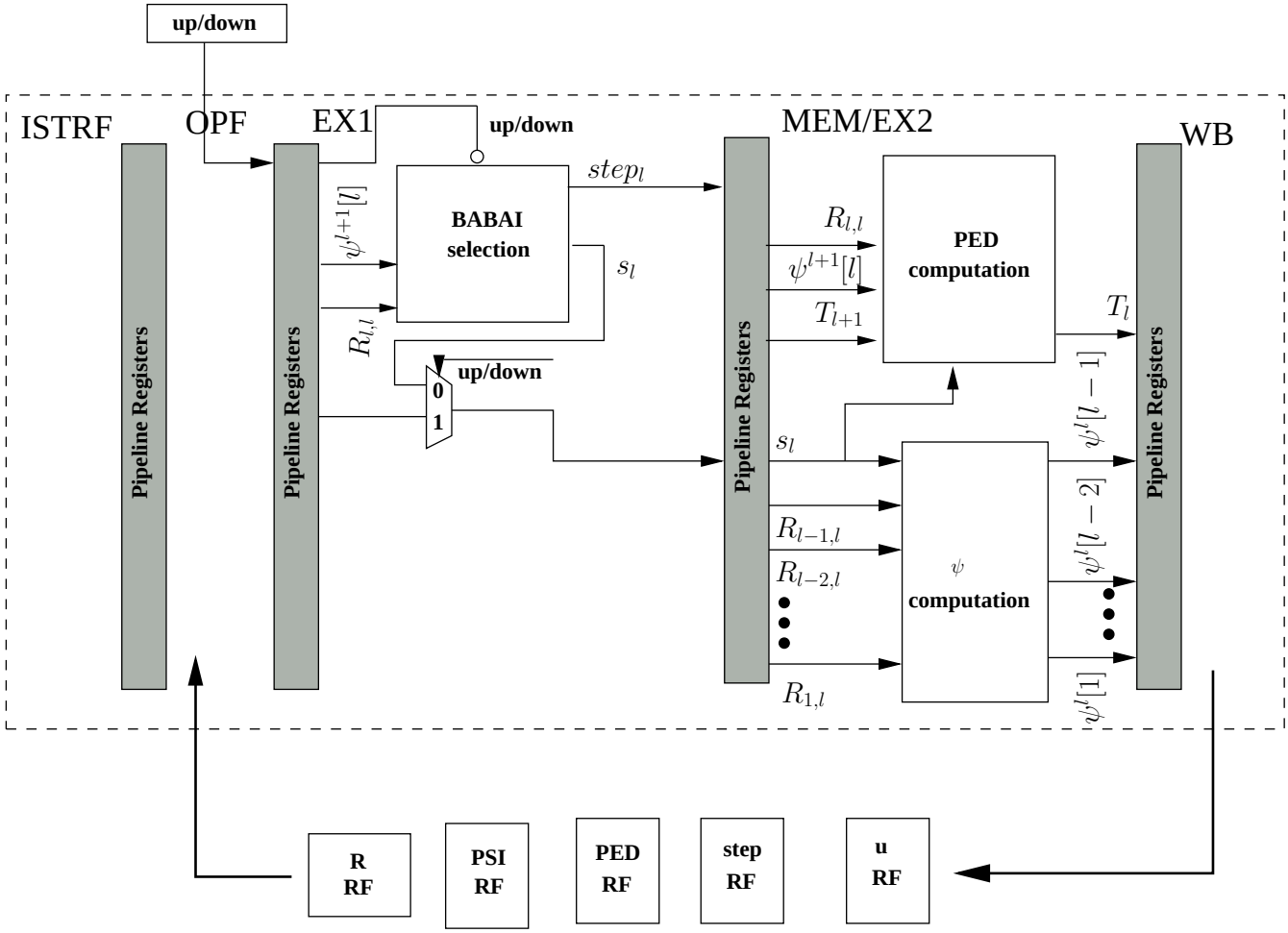


Figure 7.19: Modified BABAI instruction and stages of pipeline.

In Table 7.1, the results of synthesis are summarized. Now, the clock frequency is more than doubled, with respect to the previous ASIP design, while the occupied area is almost the same. Then, Table 7.3 details the area for each block. The external data memory is not included in these results. The architecture can be partitioned in three main units: the pipeline, the register files and the interface for external memory. While interface for channel data memory, indicated as $Area_{MemoryFile}$ is negligible, the area of register files ($Area_{RF}$) represents more than the half (59.5 %) of the whole architecture. In fact, the high memory requirement, mainly due to the list, is one of the major problem of LSD. The

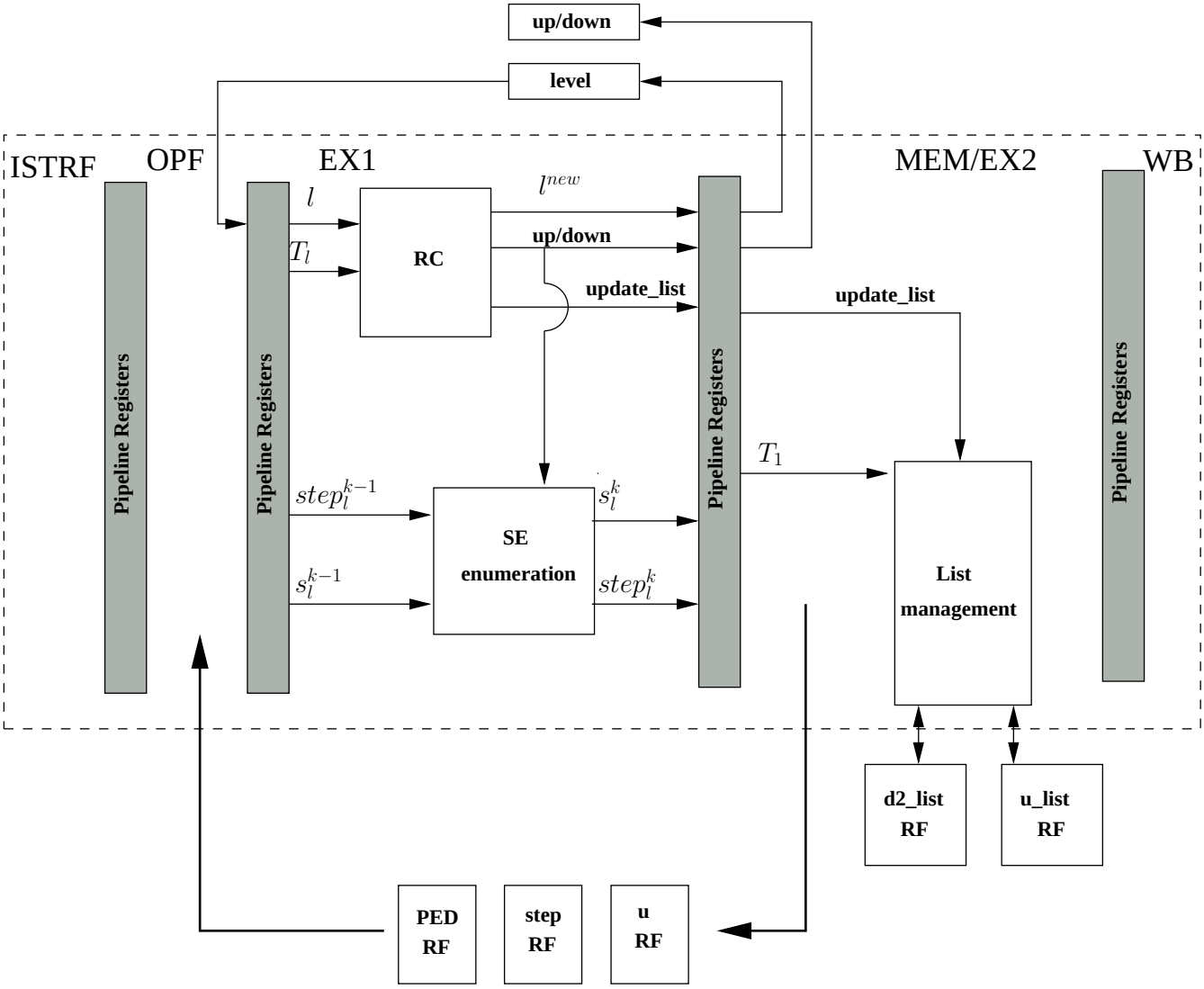


Figure 7.20: Modified CHECK instruction and stages of pipeline.

remaining 40.4 % is due to the pipeline ($Area_{pipe}$) and Table 7.3 also reports the area of all sub-blocks of it. At each pipeline stage corresponds one block, so it is quite easy to analyze these results. Most of the complexity is concentrated in the MEM stage (32.8 %). In fact, it is devoted to compute PED and ψ metrics, and to manage the list, besides loading the content of the external memory into proper registers.

Table 7.2: Synthesis results of improved LSD ASIP

Technology	ST 130nm
Frequency	228 MHz
Area	888575 um^2 (164 KGE)

Table 7.3: Detailed Area results of improved LSD ASIP

$Area_{MemoryFile}$	231 um^2 (0%)
$Area_{RF}$	528625 um^2 (59.5 %)
$Area_{pipe}$	359323 um^2 (40.4%)
$Area_{ISTRF}$	480 um^2 (0.1%)
$Area_{ISTRF\_OPF}$	1676 um^2 (0.2%)
$Area_{OPF}$	21747 um^2 (2.4%)
$Area_{OPF\_EX}$	5862 um^2 (0.7%)
$Area_{EX}$	25396 um^2 (2.9%)
$Area_{EX\_MEM}$	7238 um^2 (0.8%)
$Area_{MEM}$	291516 um^2 (32.8%)
$Area_{MEM\_WB}$	4309 um^2 (0.5%)
$Area_{WB}$	1059 um^2 (0.1%)

7.3.1 Performance

The throughput of the ASIP is detailed for different system configurations in Tables 7.4, 7.5 and 7.6. The values, reported for each SNR, are average numbers, obtained after 10^6 simulations, according to [21]:

$$Th = \frac{RQM_t}{\mathrm{E}(N_{en})} f_{clk} \; [bit/s] \tag{7.2}$$

with R being the code rate, Q the number of bits per symbol, M_t the number of transmitters f_{clk} the clock frequency and $\mathrm{E}(N_{en})$ the average number of clock cycles N_{en}, needed to detect one symbol vector . Here a convolutional code $R = 0.5$ has been adopted.

Chosen system and modulation, the throughput has not a big variation with SNRs, on the contrary it is almost constant. A traditional SDA, instead, has variation of about 80 % between low and high SNRs, as already shown in Fig. 3.12. This is due to the fact that, once the ML solution has been found, SDA stops the search, and if the noise level is high it requires more time to complete

Table 7.4: Throughput of the improved LSD ASIP for a 2×2 MIMO system.

		SNR $[dB]$	Th $[Mbps]$				
			$\lvert\mathcal{L}\rvert = 4$	$\lvert\mathcal{L}\rvert = 8$	$\lvert\mathcal{L}\rvert = 16$	$\lvert\mathcal{L}\rvert = 32$	$\lvert\mathcal{L}\rvert = 64$
2×2	QPSK	0	3,07	2.51	2.27	2.27	2.27
		6	3,21	2.57	2.30	2.30	2.30
		12	3,25	2.59	2.31	2.31	2.31
		18	3,29	2.60	2.31	2.32	2.31
		24	3,28	2.59	2.32	2.32	2.32
		30	3,27	2.59	2.32	2.32	2.32
	16QAM	0	3.79	2.76	1.96	1.41	1.04
		6	4.46	2.96	2.09	1.47	1.07
		12	4.60	3.08	2.12	1.50	1.08
		18	4.63	3.14	2.18	1.49	1.08
		24	4.54	3.14	2.15	1.50	1.08
		30	4.59	3.11	2.12	1.50	1.08
	64QAM	0	3.25	2.52	1.88	1.32	0.93
		6	4.15	3.06	2.19	1.50	1.01
		12	4.81	3.40	2.32	1.56	1.04
		18	4.96	3.55	2.34	1.60	1.06
		24	5.12	3.46	2.36	1.58	1.06
		30	5.01	3.49	2.31	1.56	1.05

his work. The LSD has the additional role of filling a list, whose size is fixed: this makes the throghput almost indipendent of the SNR.

Table 7.5: Throughput of the improved LSD ASIP for a 3×3 MIMO system.

		SNR $[dB]$	Th $[Mbps]$				
			$\lvert\mathcal{L}\rvert = 4$	$\lvert\mathcal{L}\rvert = 8$	$\lvert\mathcal{L}\rvert = 16$	$\lvert\mathcal{L}\rvert = 32$	$\lvert\mathcal{L}\rvert = 64$
3×3	QPSK	0	1.85	1.39	1.06	0.87	0.79
		6	2.05	1.46	1.10	0.89	0.80
		12	2.14	1.51	1.12	0.89	0.80
		18	2.15	1.51	1.12	0.90	0.80
		24	2.17	1.53	1.13	0.90	0.80
		30	2.17	1.53	1.14	0.90	0.80
	16QAM	0	1.50	1.12	0.81	0.57	0.39
		6	2.16	1.45	0.98	0.65	0.43
		12	2.50	1.62	1.05	0.70	0.46
		18	2.64	1.65	1.10	0.71	0.46
		24	2.64	1.73	1.10	0.73	0.47
		30	2.62	1.67	1.11	0.72	0.47
	64QAM	0	0.62	0.51	0.41	0.35	0.25
		6	0.91	0.69	0.50	0.35	0.23
		12	1.38	0.90	0.62	0.42	0.27
		18	1.53	1.04	0.68	0.44	0.29
		24	1.60	1.06	0.68	0.47	0.29
		30	1.56	1.02	0.69	0.45	0.30

Table 7.6: Throughput of the improved LSD ASIP for a 4×4 MIMO system.

		SNR $[dB]$	Th $[Mbps]$				
			$\lvert\mathcal{L}\rvert = 4$	$\lvert\mathcal{L}\rvert = 8$	$\lvert\mathcal{L}\rvert = 16$	$\lvert\mathcal{L}\rvert = 32$	$\lvert\mathcal{L}\rvert = 64$
4×4	QPSK	0	1.15	0.84	0.62	0.45	0.35
		6	1.35	0.94	0.66	0.48	0.36
		12	1.35	0.94	0.66	0.48	0.36
		18	1.49	1.01	0.69	0.49	0.36
		24	1.56	1.04	0.71	0.50	0.37
		30	1.56	1.06	0.72	0.50	0.37
	16QAM	0	0.56	0.41	0.33	0.25	0.17
		6	1.01	0.73	0.50	0.35	0.22
		12	1.01	0.73	0.50	0.35	0.22
		18	1.43	0.93	0.62	0.39	0.25
		24	1.58	1.03	0.68	0.43	0.27
		30	1.68	1.07	0.69	0.44	0.27
	64QAM	0	0.09	0.08	0.07	0.06	0.05
		6	0.36	0.29	0.22	0.18	0.13
		12	0.36	0.29	0.22	0.18	0.13
		18	0.74	0.52	0.39	0.29	0.19
		24	1.14	0.75	0.51	0.34	0.21
		30	1.32	0.86	0.57	0.36	0.23

7.3.2 Comparison with the State of the Art

To facilitate comparisons, Table 7.7 reports the main features of a number of soft MIMO detector implementations: three different ASIC implementations proposed in [113], [21] and [15] (columns 2 to 4), and two ASIPs of K-Best LSD in [114] and [112] (columns 5 and 6). Column 7 reports the achieved results for the LSD ASIP implementation.

The soft-output sphere decoder architecture, presented in [113], avoids the complexity related to PED calculation and sorting, employing a close-to Schnorr-Euchner order, with a simple look-up table. Work in [21] proposes a VLSI architecture for Soft-In-Soft-Out Single-Tree-Search SD, following the one-node-per-cycle (ONPC) paradigm, while a modified K-Best SE decoding algorithm is proposed in [15] to improve the performance of the soft-output KSE with low complexity and low-power features incorporated the VLSI architecture. The ASIP implementation of K-best list sphere detector in [114] makes use of the transport triggered architecture (TTA): it is based on using memory and heap data structure for symbol vector sorting. This architecture has been improved and presented in [112], showing low hardware and design complexities, together with various general-purpose properties.

Table 7.7 gives for each implementation the corresponding technology, maximum clock frequency (f_{CK}), area complexity (Area), expressed in equivalent gate count (EG), and average throughput (Av. Throughput). For our work and [21] the throughput depends on SNRs, as specified in the comments to the table, while all others presented a fixed one. To evaluate the efficiency of the compared solutions, row 11 gives the Throughput to Area Ratio (TAR).

In order to have a fair comparison, last two rows report respectively throughput and TAR for the processing of the considered system configuration on our ASIP.

Table 7.7: Comparisons with the State of the Art

Reference	[113]	[21]	[15]	[114]	[112]	This work
Imp.	ASIC			ASIP		
Type of detec.	SO SD	SISO	SO MKSE	K-LSD	SO K-Best LSD	SE LSD
MIMO system	2×2	4×4			2×2	2×2-3×3-4×4
Modulation	B,Q,16,64-QAM	16 QAM			64-QAM	Q,16,64-QAM
BER Perf.	Close to ML					
Tech [μm]	0.045	0.09	0.13	0.13	0.13	0.13
f_{CK} [MHz]	312	250	200	100	280	228
Area [EG]	262K	96K	97K	170K	25K	164K
Av. Th.[Mbps]	374	10-90 @12-17dB	106.6 [(6)]	5.3[(6)] $\mathcal{L}$=7,N=5[(1)]	7.6 [(6)] $\mathcal{L}$=16,N=1[(1)]	
TAR [$Mbps/KEG$]	**1.43**	**0.1-0.9** @12-17dB	**1.1**	**0.03**	**0.3**	
Our Th.[(4)]	0.93-5.16[(8)]	0.25-1.58	0.93-5.16[(7)]	12.6-32.3[(2)(7)]	3.75-4.86[(3)(7)]	
Our TAR[(4)]	**0.01-0.04**	**0.001-0.01**	**0.01-0.03**	**0.02-0.06**[(5)]	**0.03-0.04**	

[(1)]N=number of symbol vectors, detected in parallel.
[(2)]Worst case (0dB) and best case(30dB). Also a reduction in the number of clock cycles per node from 6 to about 2.2 has been considered since pipelining-interleaving (see subsection 7.3.3) has been applied.
[(3)]Worst case (0dB) and best case(26dB).
[(4)]Throughput and TAR evaluated for SE LSD on our ASIP are reported in the last two rows of each column. The system configurations change from column to column according to the references.
[(5)] This TAR value has been computed considering an estimated area of $556KGE = A_{RF} * 5 + A_{pipe} + Area_{MemoryFile}$
[(6)] This work doesn not take into account the code rate in the computation of the throughput.
[(7)] R has been eliminated in eq. (7.2) for the computation of this value.
[(8)] Worst case (64QAM, $\mathcal{L} = 64$ and 0dB) and best case (64QAM, $\mathcal{L} = 4$ and 26dB). See Table 7.4.

Looking at Table 7.7, it is clear that having a precise comparison is not easy at all, due to differences especially in the algorithm and in the type of the implementation. In particular, the efficiency of ASIC is at least ten times higher than ASIP. [113]-[21]-[15] are better than our work by a factor 100, but, apart from [113] which supports more than one modulation, they are not flexible in terms of number of antennas and constellations. Our work is, instead, comparable with that in [114]: for low SNR (0dB) 7 of our ASIPs would be necessary to reach the same efficiency, while for high SNR (30dB) it is better by a factor 2. It is worth noticing that results in [114] are for 5 processors working in parallel. Therefore, as clarified by notes $^{(2)}$ and $^{(5)}$ of the table, in the computation of our throughput, a reduction in the number of clock cycles per node has been considered, since the pipeline can be exploited in a better way, filling empty cycles. A reduction from 6 to 2.2 has been estimated, and clearly the throughput has been multiplied by a factor 5. The increase in terms of area has been considered, using $A_{RF} * 5 + A_{pipe} + Area_{MemoryFile}$. Finally, solution [112] is better than our ASIP by a factor between 7.5 and 10. They state to visit a fixed number of 328 nodes to detect one symbol vector, while our ASIP is able to visit an average number of nodes per SNR value that is smaller (less than the half (122@0dB-99@30dB)) for the same case (2×2, 64QAM and $\mathcal{L} = 16$). Therefore, this factor is not due to the number of visited nodes, but to the fact that [112] is able to almost meet the one-node-per-cycle constraint (0.74 node per cycle), while our processor needs 6 clocks per node. Moreover, it is important to highlight that the implementation in [112] is targeted for 2×2 MIMO system, while our ASIP, when also works for a 2×2, is targeted for a 4×4, and this produce certainly an effect on complexity. If our ASIP would be designed to support only a 2×2 MIMO, the area should be reduced. In particular, it is possible to estimate a reduction of the area of register files by a factor 2. Also the $Area_{pipe}$ should have a reduction, in the ψ computation unit, but it is not easy to estimate it. Therefore, to have a comparison with [112] and also [113], an area value of $115KGE = Area_{pipe} + 0.5Area_{RF} + Area_{MemoryFile}$ has been adopted (see TAR values in column 2 and 6). It is worth noticing that, [112], together with [15] and [114], do not take into account the code rate in the computation of the throughput, so, also for our reported throughput, R has been eliminated in

eq. (7.2).

Anyway, it is evident that the current design has to be furthermore improved in order to be more competitive with other ASIP implementations of LSD. Next subsection explains a good idea to additionally refine the proposed architecture.

7.3.3 Efficient pipeline usage: the pipeline-interleaving

Another important observation, which can be derived looking at Listing 7.1 (from line 14 to the end), is that the pipeline is not exploited at 100 %, but only around the 40%. A good method to have a more efficient usage of the pipeline is the well-known "pipeline interleaving"[2]. This solution consists in detecting two or more symbol vectors in parallel, and not allocating as many processors as necessary, but only with one ASIP, exploiting at maximum the pipeline.
The LISA description of the ASIP has been modified in order to support 5 symbol vectors to be detected. Each register file has been allocated 5 times, one for each vector, except for the $R\ RF$, since a block-fading scenario has been assumed. Also the interfaces between register files and the pipeline have been modified, so that each instruction can select the right resource at each time. In Table 7.8, the results of synthesis are summarized. Table 7.9 reports the area results of each block of the architecture.

Table 7.8: Synthesis results of improved LSD ASIP with "pipeline interleaving"

Technology	ST 130nm
Frequency	210 MHz
Area	2809788 um^2 (520 KGE)

The Listing 7.2 reports the assembly code for the ASIP with "pipeline interleaving". The INIT instruction initializes some configuration parameters: this sample code is for a 4×4 MIMO, 64QAM and $|\mathcal{L}| = 4$. From line 4 to line 26, five loops are dedicated to load the first $5M$ locations of data memory, containing ZF solutions for each vector, into the corresponding register file ($PSI\ RF$ 1, $PSI\ RF$ 2, $PSI\ RF$ 3, $PSI\ RF$ 4 and $PSI\ RF$ 5). The remaining $2M \times 2M$ locations, containing channel coefficients, are, then, loaded into $R\ RF$ (loop at

Table 7.9: Detailed Area results of LSD ASIP with "pipeline interleaving"

$Area_{MemoryFile}$	231 um^2 (0%)
$Area_{RF}$	2134751 um^2 (76 %)
$Area_{pipe}$	674806 um^2 (24 %)
$Area_{ISTRF}$	494 um^2 (0%)
$Area_{ISTRF\_OPF}$	1646 um^2 (0%)
$Area_{OPF}$	25888 um^2 (0.9%)
$Area_{OPF\_EX}$	6542 um^2 (0.2%)
$Area_{EX}$	48943 um^2 (1.7%)
$Area_{EX\_MEM}$	22593 um^2 (0.8%)
$Area_{MEM}$	556062 um^2 (19.8%)
$Area_{MEM\_WB}$	4875 um^2 (0.2 %)
$Area_{WB}$	5729 um^2 (0.2 %)

line 28). After that the LSD can first start to process the highest level of each tree. The search continues jumping back to "check_up_down", until the detection of each symbol vector is not complete, and to "finish" otherwise.

Now the ASIP is able to visit 2.6 node per cycle (5 nodes per 13 cycles). Moreover, the pipeline is exploited at about 85 %. A neglibile reduction of the clock frequency of about 0.7 % with respect to the ASIP without "pipeline-interleaving" has to be noticed. This is mainly due to the additional complexity as interface towards the register files. It can be estimated an increase in terms of throughput by a factor $5\frac{6}{2.6}0.92 = 10.6$, and in terms of occupied area by a factor 3.2. Therefore, the TAR is improved by a factor 3.3. Table 7.10 reports the comparison of this new results with the state of the art. The advantage of throughput is quite evident. In fact, our ASIP is better than other ASIPs: by a factor from 2 and 5, with respect to [114], and between 5 and 7, with respect to [112]. Moreover, the gap with ASIC implementations is not any more so large. The work in [21], which is the most similar to our solution, since it adopts a depth–first Schnrorr Euchner policy, is now better than ours by a factor between 4 and 5, against 40 and 56 of the previous version. We claim that the "pipeline interleaving" is a good approach to be competitive also with ASIC implementations. For example, with 10 symbol vector detected in parallel, it is possible have comparable results. The main drawback is the increase of area, which makes the ASIP not yet competitive with ASICs in terms of TAR. For what concerns ASIPs solutions, our work is

Listing 7.2: LSD assembly code with "pipeline interleaving"

```
1  ;Initialization choosing respectively pamsize numbant and listsize
2         INIT $8 $4 $3
3  ; Load the first M locations of Channel Data Memory into PSI RF 1
4  load_zf1 : ADDI $0 , $0 , 1
5            LW $0,0,$0 ,0,$1
6            BNE $0 , $1 , load_zf1
7  ; Load the successive M locations of Channel Data Memory into PSI RF 2
8            ADDI $0, $0 , -8
9  load_zf2 : ADDI $3 , $3 , 1
10           LW $3,8,$3 ,0,$2
11           BNE $3 , $1 , load_zf2
12 ; Load the successive M locations of Channel Data Memory into PSI RF 3
13           ADDI $3, $3 , -8
14 load_zf3 : ADDI $0 , $0 , 1
15           LW $0,16,$0 ,0,$3
16           BNE $0 , $1 , load_zf3
17 ; Load the successive M locations of Channel Data Memory into PSI RF 4
18           ADDI $0, $0 , -8
19 load_zf4 : ADDI $3 , $3 , 1
20           LW $3,24,$3 ,0,$4
21           BNE $3 , $1 , load_zf4
22 ; Load the successive M locations of Channel Data Memory into PSI RF 5
23           ADDI $3, $3 , -8
24 load_zf5 : ADDI $0 , $0 , 1
25           LW $0,32,$0 ,0,$5
26           BNE $0 , $1 , load_zf5
27 ; Load the successive 2M *2M locations of Channel Data Memory into R RF
28 load_R : ADDI $0 , $0 , 1
29          ADDI $3 , $3 , 1
30          LW $3,32,$0 ,1,$1
31          BNE $3,$2, load_R
```

```
highest_level:  BABAI $4,$5, $1
                BABAI $6,$7, $2
                BABAI $9,$10, $3
                BABAI $11,$12, $4
                BABAI $13,$14, $5
check_up_down: CHECK $1
                CHECK $2
                CHECK $3
                CHECK $4
                CHECK $5
                BABAI $4,$5, $1
                BABAI $6,$7, $2
                BABAI $9,$10, $3
                BABAI $11,$12, $4
                BABAI $13,$14, $5
                J check_up_down, finish
                NOP
                NOP
                NOP
finish:   NOP
```

better of about 67 % with respect to [114], showing that the estimation reported in Table 7.7 was quite good. The work in [112] is now better only by a factor between 1.87 and 2.5, with an improvement of 75 %, with respect to the ASIP without "pipeline interleaving". As in Table 7.7, to have a comparison with [112] and also [113], an area value of $323KGE = Area_{pipe} + 0.5Area_{RF} + Area_{MemoryFile}$ has been adopted (see TAR values in column 2 and 6 of Table 7.10).

We can then state that the "pipeline-interleaving" provides a significant increase of throughput, since now our ASIP has the highest throughput among the other two ASIPs in [114] and [112]. The price for the flexibility is, in this case, represented by the area, which is already too much high. This aspect can be certainly improved: probably a better way to manage a such high memory requirement is to have external memories and use a stack to access to them. Moreover, it has to be noticing that 2.6 node per cycle is achieved when all vectors are processed in parallel, i.e. the best case. The detection of one vector can finish before than another one, this means a reduction of the throughput, which can be in part limited with an improved assembly program.

Table 7.10: Comparisons of the ASIP with "pipeline interleaving " with the State of the Art

Reference	[113]	[21]	[15]	[114]	[112]	This work
Imp.	ASIC			ASIP		
Type of detec.	SO SD	SISO	SO MKSE	K-LSD	SO K-Best LSD	SE LSD
MIMO system	2×2	4×4			2×2	2×2-3×3-4×4
Modulation	B,Q,16,64-QAM	16 QAM			64-QAM	Q,16,64-QAM
BER Perf.	Close to ML					
Tech $[\mu m]$	0.045	0.09	0.13	0.13	0.13	0.13
f_{CK} [MHz]	312	250	200	100	280	210
Area $[EG]$	262K	96K	97K	170K	25K	520K
Av. Th.[Mbps]	374	10-90 @12-17dB	106.6 [(6)]	5.3[(6)] $\mathcal{L}$=7,N=5[(1)]	7.6 [(6)] $\mathcal{L}$=16,N=1[(1)]	
TAR $[Mbps/KEG]$	**1.43**	**0.1-0.9** @12-17dB	**1.1**	**0.03**	**0.3**	
Our Th.[(4)]	9.8-54.6[(8)]	2.6-16.7	9.8-54.6[(7)]	9.8-25.1[(2)(7)]	39.7-51.5[(3)(7)]	
Our TAR[(4)]	**0.03-0.17**	**0.005-0.03**	**0.02-0.10**	**0.02-0.05**[(5)]	**0.12-0.16**	

[(1)] N=number of symbol vectors, detected in parallel.
[(2)] Worst case (0dB) and best case(30dB). The actual reduction in the number of cycles per node is 6 to 2.6.
[(3)] Worst case (0dB) and best case(26dB).
[(4)] Throughput and TAR evaluated for SE LSD on our ASIP are reported in the last two rows of each column. The system configurations change from column to column according to the references.
[(5)] This TAR value has been computed considering an actual area of 520 KGE.
[(6)] This work doesn not take into account the code rate in the computation of the throughput.
[(7)] R has been eliminated in eq. (7.2) for the computation of this value.
[(8)] Worst case (64QAM, $\mathcal{L} = 64$ and 0dB) and best case (64QAM, $\mathcal{L} = 4$ and 26dB).

7.4 Discussion of the results

A first suboptimal ASIP implementation of the list generation unit of LSD has been presented. A description of architectural choices and flexibility parameters is detailed. Results of synthesis are also provided. Then, an improved version of the ASIP in terms of clock frequency is given and compared with other state of the art implementations. At the best of our knowledge, this is the first ASIP of LSD, supporting multiple antennas, modulation orders and list sizes. But this flexibility is paid in terms of lower throughput and lower throughput to area ratio, than other implementations. In order to enable high throughput and to provide a more efficient usage of the pipeline, the "pipeline-interleaving" of 5 symbol vectors has been developed. Targeting 130 nm CMOS standard cell technology, synthesis results show that the proposed solution achieves the best throughput among other ASIPs of LSD. It is even better than other methods, with a comparable flexibility, such as the reconfigurable MIMO detector implemented using a Graphic Processor Unit (GPU) in [115]. Moreover, throughput results are of the same order of magnitude of most ASIC solutions, able to satisfy many standard requirements. This demonstrates the feasibility of the ASIP approach, which has not been sufficiently exploited up to now. Certainly, further aspects could be investigated and improved, in particular some efforts could already be spent to manage the memory requirement in a more efficient way, with respect to a register-based implementation. This probably enables a reduction of area, improving the TAR.

8

Conclusions

This research thesis deals with the theme of MIMO detection, both hard and soft. It focuses on algorithmic, architectural and implementation aspects of the Sphere Decoder Algorithm and one of its soft versions, the List Sphere Decoder. From a detailed analysis on the state of the art, software models and architectural issues, some critical aspects become relevant: the throughput and the flexibility. This PhD thesis has the purpose to find solutions to this two problems.

In the hard detection scenario, a detailed study on state of the art MIMO detection algorithms has been conducted. The description of the depth–first Schnorr Euchner Sphere Decoder Algorithm, which represents the starting point of this research thesis, has been presented. Also, other versions of SDA, exploited in literature, are introduced. Then a modified Sphere Decoder Algorithm has been proposed. While most of state of the art solutions achieve performance that is close to ML, this research work faces the problem of a high throughput full ML implementation. The look-ahead, pipelining and retiming techniques, combined with a modified search strategy (Test & Restart), produce a significant increase of throughput, with no loss of error rate performance. The VHDL design of the presented solution, called LASDA (Look–ahead SDA), has been targeted for 4×4 MIMO system and 16QAM modulation. Synthesis results on a 130 nm technology show a throughput speedup of 92% at $SNR = 20$ dB , with respect to a serial unpipelined SDA solution, and a limited additional complexity of 24% in terms of occupied area.

It is important to underline that this high throughput ML solution can easily be applied for most soft-output detection schemes. The main advantage that can be obtained is an increase of about 50 % of the initial clock frequency.

In the soft detection scenario, the idea of a multi-mode flexible MIMO detector has been presented. This idea is supported by a detailed analysis of existing hard and soft MIMO detector implementations. The key word in this context is the flexibility, which can be seen in three different meanings: performance-complexity trade-offs, the capability of supporting multiple antennas and modulation orders, and the adaptability to change the detection algorithm according to channel condition and targeted performance. In order to achieve the maximum degree of flexibility, among the soft versions of SDA, the List Sphere Decoder has been chosen. In fact, it is the easiest extension of a traditional depth-first hard SDA, and offers some parameters, such as the size of the list and the LLR clipping, which can be easily tuned, to obtain the desired performance-complexity trade-off. The idea is a multi-algorithm detector, able to combine performance of LSD and complexity of a linear detector, such as Minimum-Mean-Squared-Error-Interference-Canceller (MMSE-IC).

To exploit the previous idea, a detailed convergence and flexibility analysis on LSD has been conducted. In particular, the iterative behavior of LSD detector has been analyzed using EXIT charts, in order to deeply understand the problem of divergence. Moreover, the LSD has been compared with MMSE-IC, in terms of performance and complexity, both for block and fast-fading channel models. MMSE-IC has similar performance to LSD for small constellations and system configurations, in the fast-fading scenario. On the other hand, LSD has better BER performance than MMSE-IC when the estimation of the channel conditions remain constant over a frame. Although the complexity of LSD is at least ten times higher than that of MMSE-IC, LSD is able to outperform MMSE-IC in two iterations. There are solutions in literature, showing how it is possible to reduce the complexity of the LSD. Therefore, the performed analysis on possible complexity-performance trade-offs can be furthermore exploited to extend flexi-

bility and to support again the idea of a multi-mode detector.

The implementation of LSD is the first step towards the achievement of the presented idea and the ASIP approach has been adopted, to obtain the required trade-off between flexibility and performance. After a first suboptimal ASIP architecture of the list generation unit of LSD, an improved version in terms of clock frequency has been presented. The ASIP flexibility allows its reuse for a 2×2, 3×3 and 4×4 MIMO systems, with QPSK, 16QAM, and 64QAM, and list size from 1 to 64. At the best of our knowledge, this is the first ASIP of LSD, supporting multiple antennas, modulation orders and list sizes. When targeting a 130 nm technology, the proposed architecture enables lower throughput and lower throughput to area ratio than other dedicated (ASIC based) state of the art implementations. The "pipeline interleaving" of 5 symbol vectors has been developed, obtaining an increase of throughput by a factor 10.6, by far exceeding other ASIPs, exploited in literature. Moreover, this results are also of the same order of magnitude of those of ASICs. This shows the effectiveness of the ASIP approach, combined with the "pipeline interleaving" method. It even allows to exceed ASICs performance, with many more symbols vector in parallel, but paying with an increased area. This is the price for the flexibility.

From short term perspectives, further aspects of the presented ASIP architecture could be investigated and improved. In particular some additional efforts should be spent to manage the memory requirement in a more efficient way, with respect to a register-based implementation. For example, the memory-based approach could be employed in combination with a data structure, like a FIFO. This methods probably allows a reduction of area, making the solution more competitive with the state of the art.

From the mean and long term work perspective, the LLR generation unit has to be designed, in order to complete the LSD detector. After that, the possibility that LSD and MMSE-IC share some resource has to be investigated, in order to modify the architecture, supporting both algorithms. This is, probably, another possible solution to increase the throughput.

References

[1] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichter, and H. Bolcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE Journal of Solid-State Circuits*, vol. 40, pp. 1566–1577, Jul. 2005. vi, 2, 14, 16, 17, 21, 49, 50, 51

[2] A. Burg, M. Wenk, and W. Fichtner, "VLSI implementation of pipelined sphere decoding with early termination," in *Proc. of European Signal Proc. Conf. (EUSIPCO 2006)*, Florence, Italy, Sep. 2006. vi, 49, 50, 51, 52, 140

[3] B. Cerato, G. Masera, and E. Viterbo, "Decoding the golden code: a VLSI design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 1, pp. 156–160, Jan. 2009. vi, 2, 10, 16, 21, 43, 45, 46, 49, 50, 51, 52, 103

[4] C. Studer, A. Burg, and H. Bolcskei, "Soft-output sphere decoding: Algorithms and VLSI implemention," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 2, pp. 290–300, Feb. 2008. vi, ix, 3, 56, 57, 77, 78, 79, 82, 84, 85, 86

[5] M. Wenk, A. Burg, M. Zellweger, C. Studer, and W. Fichtner, "VLSI implementation of the list sphere algorithm," in *the 24th Norchip Conference*, Nov. 2006, pp. 107–110. vi, 79, 80, 91

[6] A. R. Jafri, A. Baghdadi, and M. Jézéquel, "A universal mmse-ic linear equalizer for multi wireless standards." vi, vii, ix, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86

[7] P. Bhagawat, R. Dash, and C. Gwan, "Dynamically reconfigurable soft output mimo detector," in *Computer Design, 2008. ICCD 2008. IEEE In-*

ternational Conference on, Oct. 2008, pp. 68–73. vii, ix, 77, 78, 79, 80, 81, 82

[8] K. Amis, G. Sicot, and D. Leroux, "Reduced complexity near-optimal iterative receiver for Wimax full-rate space-time code," in *Turbo Codes and Related Topics, 2008 5th International Symposium on*, Sept. 2008, pp. 102–106. vii, 81, 82

[9] O. Paker, S. Eckert, and A. Bury, "A low cost multi-standard near-optimal soft-output sphere decoder: Algorithm and architecture," in *Design, Automation & Test in Europe Conference & Exhibition (DATE'10)*. vii, ix, 77, 78, 82, 83

[10] J. Liu and J. Li, "Turbo processing for an OFDM-based MIMO system," *IEEE Transactions on Wireless Communications*, vol. 4, no. 5, pp. 1988–1993, Sept. 2005. vii, 4, 90, 91, 96, 97, 98

[11] T. Noll, EECS, RWTH Aachen. [Online]. Available: http://www.eecs.rwth-aachen.de vii, viii, 105, 110

[12] T. Cupaiuolo, M. Siti, and A. Tomasoni, "Low-complexity and high throughput VLSI architecture of soft-output ML MIMO detector," in *Design, Automation & Test in Europe Conference & Exhibition (DATE'10)*. ix, 3, 77, 78, 79, 81, 82

[13] M. T. Gamba and G. Masera, "Look–ahead Sphere Decoding: Algorithm and VLSI Architecture," *IET communications*, accepted on 6th of December 2010. ix, 76, 77, 78, 79, 82, 84

[14] H. Bolcskei, D. Gesbert, C. B. Papadias, and A. J. Van Der Veen (Editors), *Space-time wireless systems : from array processing to MIMO communications*. Cambridge University Press, 2006. 2, 16, 111

[15] 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. 2, 49, 50, 120, 137, 138, 139, 144

[16] Jin Jie, Chi-ying Tsui, and Wai-ho Mow, "A threshold-based algorithm and VLSI architecture of a k-best lattice decoder for MIMO systems,"

in *IEEE International Symposium on Circuits and Systems, 2005. ISCAS 2005*, vol. 4, May 2005, pp. 3359 – 3362. 2, 49, 50

[17] Z. Guo and P. Nilsson, "VLSI implementation issues of lattice decoders for MIMO systems," in *Proceedings of the 2004 International Symposium on Circuits and Systems, 2004. ISCAS '04*, vol. 4, May 2004, pp. 477–480. 2, 21

[18] B. M. Hochwald and S. T. Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Transactions on Communications*, vol. 51, no. 3, pp. 389–399, March 2003. 3, 52, 57

[19] C. Studer, M. Wenk, A. Burg, and H. Bolcskei, "Soft-output MIMO detection algorithms: Performance and implemention aspects," in *Proc. 40h Asilomar Conf. on Signals, Systems, and Computers*, Oct. 2006, pp. 2071–2076. 3, 57

[20] P. Marsch, E. Zimmermann, and G. Fettweis, "Smart candidate adding: A new low-complexity approach towards near-capacity MIMO detection," in *Proc. 13h European Signal Processing Conf. (EUSIPCO)*, Sep. 2005. 3

[21] E. M. Witte, F. Borlenghi, G. Ascheid, R. Leupers, and H. Meyr, "A scalable VLSI architecture for Soft-Input Soft-Output Single Tree-Search Sphere Decoding," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 57, no. 9, pp. 706 –710, Sep. 2010. 3, 133, 137, 138, 139, 141, 144

[22] Y. Sun and J. R. Cavallaro, "High throughput VLSI architecture for soft-output MIMO detection based on a greedy graph algorithm," in *Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI'09*, New York, USA, Mar. 2009, pp. 445 – 450. 3

[23] 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, March 2003. 4, 88, 89, 91, 94

[24] Y. Dai, S. Sun, Z. Lei, and Y. Li, "A list sphere decoder based turbo receiver for groupwise space time trellis coded (GSTTC) systems," in *Proceedings of*

the 59th IEEE Vehicular Technology Conference, VTC 2004-Spring, vol. 2, May 2004, pp. 804–808 Vol.2. 4, 90, 91, 96

[25] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Transactions on Communications*, vol. 49, no. 10, pp. 1727–1737, Oct 2001. 4, 91, 94

[26] X. Wang, K. Niu, Z. He, W. Wu, and X. Zhang, "List sphere decoding combined with linear detection-based iterative soft interference cancellation via exit chart," *Proceedings of the 17th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 1–5, 2006. 4, 102

[27] D. Gesbert, M. Shafi, Da-shan 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, pp. 281–302, Apr. 2003. 8

[28] *Multiple-input multiple-output in UTRA (Release 7)*, 3GPP Std. TR25.876, 2006. 8

[29] 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. 12

[30] C. Schnorr and M. Euchner, "Lattice basis reduction: improved practical algorithms and solving subset sum problems," *Math. Computat.*, vol. 66, no. 2, pp. 181–191, 2004. 12

[31] D. Wubben, R. Bohnke, V. Kuhn, and K.-D. Kammeyer, "MMSE extension of V-BLAST based on sorted QR decomposition," vol. 1, oct. 2003, pp. 508 – 512 Vol.1. 14

[32] B. Mennenga and G. Fettweis, "Search sequence determination for tree search based detection algorithms," in *IEEE Sarnoff Symposium, SARNOFF '09*, Mar. 2009, pp. 1–6. 16, 52

[33] S. Lee, J. Lee, S. Seo, and S. C. Park, "VLSI implementation of area-efficient list sphere decoder," in *International Symposium on Intelligent Signal Processing and Communications, ISPACS 2006*, Dec. 2006, pp. 610 – 613. 17

[34] M. Myllyla, M. Juntti, and J. Cavallaro, "Implementation aspects of list sphere detector algorithms," in *IEEE Global Telecommunications Conference, GLOBECOM '07*, Nov. 2007, pp. 3915 – 3920. 17

[35] R. S. C. K. Wong, C. Tsui and W. Mow, "A VLSI architecture of a Kbest lattice decoding algorithm for MIMO channels," vol. 1, 2002, pp. 273 – 276. 18

[36] U. Fincke and M. Pohst, "Improved methods for calculating vectors of short lenght in a lattice, including a complexity analysis," *Math. Computat.*, vol. 40, no. 170, pp. 463–471, 1985. 18

[37] R. Gowaikar and B. Hassibi, "Efficient near-ml decoding via statistical pruning," in *Information Theory, 2003. Proceedings. IEEE International Symposium on*, 2003. 19

[38] L. Barbero and J. Thompson, "A fixed-complexity MIMO detector based on the complex sphere decoder," in *Signal Processing Advances in Wireless Communications, 2006. SPAWC '06. IEEE 7th Workshop on*, 2006, pp. 1 –5. 19

[39] C. Hess, M. Wenk, A. Burg, P. Luethi, C. Studer, N. Felber, and W. Fichter, "Reduced complexity MIMO detector with close-to ML error rate performance," in *Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI'07*, Stresa-Lago Maggiore, Italy, Mar. 2007. 19, 21

[40] A. Burg, M. Borgmanr, M. Wenk, 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, 2006, p. 6 pp. 20

[41] L. Hsin-Lei, R. Chang, and C. Hung-Lien, "A high-speed SDM-MIMO decoder using efficient candidate searching for wireless communication," *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, vol. 55, no. 3, pp. 289–293, 2008. 21

[42] L. Barbero and J. Thompson, "Performance analysis of a fixed-complexity sphere decoder in high-dimensional MIMO systems," in *Proceedings of the*

IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2006, May 2006. 21

[43] M. Siti and M. Fitz, "On layer ordering techniques for near-optimal MIMO detectors," in *Proceedings of the IEEE Wireless Communications and Networking Conference, WCNC 2007*, Mar. 2007, pp. 1199–1204. 21

[44] Jin Lee, Hyung Soon Kim, and Sin-Chong Park, "Parallel architecture for sphere decoder with runtime constraint," in *Proceedings of the IEEE Consumer Communications and Networking Conference, CCNC 2008*, Jan. 2008, pp. 181–184. 21

[45] L. Barbero and J. Thompson, "Fixing the complexity of the sphere decoder for MIMO detection," *IEEE Transactions on Wireless Communications*, vol. 7, no. 6, pp. 2131–2142, June 2008. 21

[46] S. Mondal, A. Eltawil, C. A. Shen, and K. N. Salama, "Design and implementation of a Sort-Free K-Best Sphere Decoder," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, May 2010, to be published. 22, 49, 50

[47] Tae Ho Im, Insoo Park, Jinmin Kim, Joohyun Yi, Jaekwon Kim, Sungwook Yu, and Yong Soo Cho, "A new signal detection method for spatially multiplexed MIMO systems and its VLSI implementation," *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, vol. 56, no. 5, pp. 399 – 403, 2009. 22, 49, 50

[48] K. Parhi, *VLSI Digital Signal Processing Systems*. USA: John Wiley and Sons, Inc., 1959. 22, 29, 32

[49] B. Cerato, "Design of digital architectures for telecommunication systems with special emphasys in integrated circuits for MIMO-OFDM wireless channels," Ph.D. dissertation, Politecnico di Torino, Torino, 2007. 34

[50] B. Cerato, G. Masera, and E. Viterbo, "Enabling VLSI processing blocks for MIMO-OFDM communications," *VLSI Design Journal*, vol. 8, no. 2, pp. 1–10, Jan. 2008. 48

[51] S. Le Goff, A. Glavieux, and C. Berrou, "Turbo-codes and high spectral efficiency modulation," in *Communications, 1994. ICC '94, SUPERCOMM/ICC '94, Conference Record, 'Serving Humanity Through Communications.' IEEE International Conference on*, May 1994, pp. 645 –649 vol.2. 55

[52] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on*, vol. 2, May 1993, pp. 1064 –1070 vol.2. 55

[53] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *Communications, IEEE Transactions on*, vol. 44, no. 10, pp. 1261 –1271, Oct. 1996. 55, 59

[54] C. Berrou, R. Pyndiah, P. Adde, C. Douillard, and R. Le Bidan, "An overview of turbo codes and their applications," in *Wireless Technology, 2005. The European Conference on*, 2005, pp. 1 –9. 55

[55] O. Damen, A. Chkeif, and J.-C. Belfiore, "Lattice code decoder for space-time codes," *Communications Letters, IEEE*, vol. 4, no. 5, pp. 161 –163, may 2000. 55, 88

[56] D. Sorrenti, "Architetture digitali per ricevitori multi-antenna," M. Eng. thesis, Politecnico di Torino, Torino, Italia, May 2009. 57

[57] L. Ping, L. Liu, K. Wu, and W. Leung, "Interleave division multiple-access," *IEEE Transactions on Wireless Communications*, vol. 5, no. 4, pp. 938–947, April 2006. 58

[58] L. Liu, W. Leung, and L. Ping, "Simple iterative chip-by-chip multiuser detection for CDMA systems," in *Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual*, vol. 3, 2003, pp. 2157–2161. 61

[59] B. N. Datta, *Numerical Linear Algebra*. California: Brooks/Cole Publishing Company, 2004. 61

[60] W. Keing, "Semi-random forward error correction codes and space-time codes," Ph.D. dissertation, City University of Hong Kong, Hong Kong, 2005. [Online]. Available: http://hdl.handle.net/2031/4738 65

[61] P. Salmela, A. Happonen, T. Jarvinen, A. Burian, and J. Takala, "DSP implementation of Cholesky decomposition," in *Mobile Future, 2006 and the Symposium on Trends in Communications. SympoTIC '06. Joint IST Workshop on*, 2006, pp. 6–9. 69

[62] F. Echman and V. Owall, "A scalable pipelined complex valued matrix inversion architecture," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, vol. 5, May 2005, pp. 4489–4492. 84

[63] M. Myllyla, J.-H. Hintikka, J. Cavallaro, M. Juntti, M. Limingoja, and A. Byman, "Reduced complexity near-optimal iterative receiver for Wimax full-rate space-time code," in *Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference on*, Nov. 2005, pp. 75–81. 84

[64] M. Karkooti, J. Cavallaro, and C. Dick, "FPGA implementation of matrix inversion using QRD-RLS algorithm," in *Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference on*, Nov. 2005, pp. 1625–1629. 84

[65] P. Luethi, A. Burg, S. Haene, D. Perels, N. Felber, and W. Fichtner, "VLSI implementation of a high-speed iterative sorted MMSE QR decomposition," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, May 2007, pp. 1421–1424. 84

[66] P. Luethi, C. Studer, S. Duetsch, E. Zgraggen, H. Kaeslin, N. Felber, and W. Fichtner, "Gram-Schmidt-based QR decomposition for MIMO detection: VLSI implementation and comparison," in *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, Dec. 2008, pp. 830–833. 84

[67] L. Kuang-Hao, R. Chang, H. Chien-Lin, C. Feng-Chi, and L. Shih-Chun, "Implementation of QR decomposition for MIMO-OFDM detection systems," in *Electronics, Circuits and Systems, 2008. ICECS 2008. 15th IEEE International Conference on*, Sept. 2008, pp. 57–60. 84

[68] A. R. Jafri, A. Baghdadi, and M. Jezequel, "ASIP-Based Universal Demapper for Multiwireless Standards," *Embedded Systems Letters, IEEE*, vol. 1, no. 1, pp. 9–13, 2009. 85

[69] M. Tuchler, A. Singer, and R. Koetter, "Minimum mean squared error equalization using a priori information," *Signal Processing, IEEE Transactions on*, vol. 50, no. 3, pp. 673 –683, Mar 2002. 89

[70] H. Xiaodong Wang, Poor, "Iterative (turbo) soft interference cancellation and decoding for coded CDMA," *IEEE Transactions on Communications*, vol. 47, no. 7, pp. 1046 –1061, Jul 1999. 89

[71] Q. Chen, H. Wang, M. Chen, and S. Cheng, "An improved turbo-BLAST system for quasistatic channel," vol. 3, Sept. 2004, pp. 1588 – 1591. 89

[72] C. Laot, R. Le Bidan, and D. Leroux, "Low-complexity MMSE turbo equalization: a possible solution for EDGE," *IEEE Transactions on Wireless Communications*, vol. 4, no. 3, pp. 965 – 974, May 2005. 90

[73] Y. de Jong and T. Willink, "Iterative tree search detection for MIMO wireless systems," in *Proceedings of the 56th IEEE Vehicular Technology Conference, VTC 2002-Fall.*, vol. 2, 2002, pp. 1041–1045 vol.2. 91

[74] E. Zimmermann, D. Milliner, J. Barry, and G. Fettweis, "Optimal LLR clipping levels for mixed hard/soft output detection," in *Proceedings of the IEEE Global Telecommunications Conference, GLOBECOM 2008*, 30 2008-Dec. 4 2008, pp. 1–5. 91, 95

[75] S. ten Brink, "Designing iterative decoding schemes with the extrinsic information transfer chart," *AËU Int. J. Electron. Commun.*, vol. 54, no. 6, pp. 389–398, Dec 2000. 91, 92

[76] T. Cover and J. Thomas, *Elements of Information Theory*. New York: Wiley, 1991. 92

[77] J. Hagenauer, "The exit chart - introduction to extrinsic information transfer," in *in Iterative Processing, In Proc. 12th Europ. Signal Proc. Conf (EUSIPCO*, 2004, pp. 1541–1548. 93

[78] C. Laot, R. Le Bidan, and D. Leroux, "Low-complexity MMSE turbo equalization: a possible solution for EDGE," *IEEE Transactions on Wireless Communications*, vol. 4, no. 3, pp. 965–974, May 2005. 97

[79] X. Wang and H. Poor, "Iterative (turbo) soft interference cancellation and decoding for coded CDMA," *IEEE Transactions on Communications*, vol. 47, no. 7, pp. 1046–1061, Jul 1999. 97

[80] A. Jafri, D. Karakolah, A. Baghdadi, and M. Jezequel, "ASIP-based flexible MMSE-IC linear equalizer for MIMO turbo-equalization applications," in *Proceedings of the Europe Conference Exhibition on Design and Automation Test, DATE '09.*, 20-24 2009, pp. 1620 –1625. 98

[81] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm i. expected complexity," *Signal Processing, IEEE Transactions on*, vol. 53, no. 8, pp. 2806 – 2818, aug. 2005. 102

[82] B. Mennenga, R. Fritzsche, and G. Fettweis, "Iterative soft-in soft-out sphere detection for MIMO systems," in *Proceedings of the 69th Vehicular Technology Conference, VTC Spring 2009.*, 26-29 2009, pp. 1 –5. 103

[83] A. Hoffmann, T. Kogel, A. Nohl, G. Braun, O. Schliebusch, O. Wahlen, A. Wieferink, and H. Meyr, "A novel methodology for the design of application-specific instruction-set processors (ASIPs) using a machine description language," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 20, no. 11, pp. 1338 –1354, Nov. 2001. 104, 107, 108

[84] H. Meyr, "Application specific instruction-set processors (ASIP's) for wireless communications: design, cost, and energy efficiency vs. flexibility," in *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*, 2004, pp. 1 – 2. 104

[85] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, O. Wahlen, and H. Meyr, "A methodology for the design of application specific instruction set processors (ASIP) using the machine description language lisa," in *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*, 2001. 104, 107

[86] D. Liu, "ASIP (Application Specific Instruction-set Processors) design," in *ASIC, 2009. ASICON '09. IEEE 8th International Conference on*, 2009, p. 16. 104

[87] A. Jafri, A. Baghdadi, and M. Jezequel, "Rapid Prototyping of ASIP-based Flexible MMSE-IC Linear Equalizer," in *Rapid System Prototyping, 2009. RSP '09. IEEE/IFIP International Symposium on*, 2009, pp. 130 –133. 104

[88] R. Leupers, "From ASIP to MPSoC - Architectures and Design Tools for Communication and Multimedia systems," in *Computer Engineering Colloquium at TU Delft*, 2006. 105

[89] CoWare Processor Designer Homepage. [Online]. Available: http://www.coware.com/products/processordesigner.php 106

[90] Target IP Designer Homepage. [Online]. Available: http://www.retarget.com/resources.php 106

[91] Tensilica Xtensa 7 Homepage. [Online]. Available: http://www.tensilica.com/products/x7_processor_generator.htm 106

[92] ARC Configurable Cores Homepage. [Online]. Available: http://www.arc.com/configurablecores/ 106

[93] H. Corporaal, *Microprocessor Architectures: from VLIW to TTA*. John Wiley and Sons, Ltd., 1998. 106

[94] TTA-based Codesign Environment (TCE) Homepage. [Online]. Available: http://tce.cs.tut.fi 106

[95] Stretch Software-Configurable Processors Homepage. [Online]. Available: http://www.stretchinc.com/technology/ 106

[96] B. Mei, A. Lambrechts, J.-Y. Mignolet, D. Verkest, and R. Lauwereins, "Architecture exploration for a reconfigurable architecture template," *Design Test of Computers, IEEE*, vol. 22, no. 2, pp. 90 – 101, 2005. 106

[97] W. D. NEWCOM++(FP7). Report on the state for the art on hardware architectures for flexible radio and intensive signal processing. [Online]. Available: http://www.newcom-project.eu:8080/Plone/public-deliverables/research/DR.C.1_final.pdf 106

[98] A. Fauth, J. Van Praet, and M. Freericks, "Describing instruction set processors using nML," in *European Design and Test Conference, 1995. ED TC 1995, Proceedings.*, Mar. 1995, pp. 503 –507. 107

[99] V. Rajesh and R. Moona, "Processor modeling for hardware software codesign," in *VLSI Design, 1999. Proceedings. Twelfth International Conference On*, Jan. 1999, pp. 132 –137. 107

[100] G. Hadjiyiannis, S. Hanono, and S. Devadas, "Isdl: An instruction set description language for retargetability," in *Design Automation Conference, 1997. Proceedings of the 34th*, Jun. 1997, pp. 299 –302. 107

[101] V. Zivojnovic, S. Pees, and H. Meyr, "LISA-machine description language and generic machine model for HW/SW co-design," in *VLSI Signal Processing, IX, 1996., [Workshop on]*, 1996. 107

[102] A. Hoffmann, H. Meyr, , and R. Leupers, *Architecture Exploration for Embedded Processors with LISA*. Springer, 1st edition November 2002. 107

[103] S. Pees, A. Hoffmann, V. Zivojnovic, and H. Meyr, "LISA-machine description language for cycle-accurate models of programmable DSP architectures," in *Design Automation Conference, 1999. Proceedings. 36th*, 1999. 107

[104] J. Antikainen, P. Salmela, O. Silven, M. Juntti, J. Takala, and M. Myllyla, "Application-specific instruction set processor implementation of list sphere detector," in *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*, 2007, pp. 943 –947. 119, 120, 121

[105] R. L. Rivest, T. H. Cormen, C. E. Leiserson, and C. Stein, *Introduction to Algorifhnis (Second Edition)*. MIT Press and McGraw-Hill, 2002. 120

[106] A. Wiesel, X. Mestre, A. Pages, and J. Fonollosa, "Efficient implementation of sphere demodulation," in *Signal Processing Advances in Wireless Communications, 2003. SPAWC 2003. 4th IEEE Workshop on*, 2003, pp. 36 – 40. 120

[107] B. Widdup, G. Woodward, and G. Knagge, "A highly-parallel VLSI architecture for a list sphere detector," in *Communications, 2004 IEEE International Conference on*, vol. 5, 2004, pp. 2720 – 2725 Vol.5. 120

[108] G. S. Brodal and M. Pinotti, "Comparator networks for binary heap construction," *Theoretical Computer Science*, vol. 250, no. 1-2, pp. 235 – 245, 2001. 120

[109] K. wai Wong, C. ying Tsui, R.-K. Cheng, and W. ho Mow, "A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels," in *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, 2002. 120

[110] P. Bengough and S. Simmons, "Sorting-based VLSI architectures for the M-algorithm and T-algorithm trellis decoders," *Communications, IEEE Transactions on*, vol. 43, no. 234, pp. 514 –522, 1995. 120

[111] T. Leighton, Y. Ma, and T. Suel, "On probabilistic networks for selection, merging, and sorting," in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, ser. SPAA '95. New York, NY, USA: ACM, 1995, pp. 106–118. [Online]. Available: http://doi.acm.org/10.1145/215399.215429 120

[112] J. Antikainen, P. Salmela, O. Silveny, M. Juntti, J. Takala, and M. Myllyla, "Fine-grained application-specific instruction set processor design for the k-best list sphere detector algorithm," in *Embedded Computer Systems: Architectures, Modeling, and Simulation, 2008. SAMOS 2008. International Conference on*, 2008, pp. 108 –115. 121, 137, 138, 139, 141, 143, 144

[113] O. Paker, S. Eckert, and A. Bury, "A low cost multi-standard near-optimal soft-output sphere decoder: Algorithm and architecture," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 1402 –1407. 137, 138, 139, 143, 144

[114] J. Antikainen, P. Salmela, O. Silven, M. Juntti, J. Takala, and M. Myllyla, "Application-specific instruction set processor implementation of list sphere detector," in *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*, 2007, pp. 943 –947. 137, 138, 139, 141, 143, 144

[115] M. Wu, Y. Sun, and J. Cavallaro, "Reconfigurable real-time MIMO detector on GPU," in *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on*, 2009, pp. 690 –694. 145

List of Publications

Journal

[1] M. Troglia Gamba and G. Masera, "Look–ahead Sphere Decoding: Algorithm and VLSI Architecture", IET communications, accepted for publication on December 2010.

Proceedings of International Conferences

[1]M. Troglia Gamba and G. Masera, "Look-ahead Sphere Decoding: Algorithm and Performance Evaluation", Proceedings of the 6th International Symposium on Wireless Communication Systems (ISWCS'09), Siena, Italy, September 7-10, 2009.

[2]M. Troglia Gamba, G. Masera and A. Baghdadi, "Iterative MIMO Detection: Flexibility and Convergence Analysis of Soft-Input Soft-Output List Sphere Decoding and Linear MMSE Detection", Proceedings of the 18th International Conference on Software Telecommunications and Computer Networks (SoftCOM 2010), Bol-Split, Croatia, September 23-25, 2010.

Workshop Presentations

[1]Workshop FIRB-2008, Title of the presentation: "VLSI architectures for high throughput detection of MIMO signals", Politecnico di Torino, November 25, 2008.

[2]NEWCOM++ Winter School on "Flexible Radio and related technologies, Title of the presentation: "VLSI architectures for high throughput detection of MIMO signals", UMIC Research Centre, Aachen, Germany, 5 February 2009.

[3]NEWCOM ++ Dissemination Day, Title of the poster: "An Application Specific Instruction Set processor for Signal Detection in Multiple Antenna Systems", Firenze, June 18, 2010.

[4]Workshop FIRB-2010, Title of the presentation: "Iterative MIMO Detection: Flexibility and Convergence Analysis", Universita' degli studi di Pavia, September 28, 2010.