Efficient VLSI Implementation of Soft-input Soft-output Fixed-complexity Sphere Decoder

Efficient VLSI Implementation of Soft-input Soft-output Fixed-complexity Sphere Decoder

Bin Wu, Guido Masera
VLSI Lab, Department of Electronics, Politecnico di Torino
Corso Duca degli Abruzzi 24, 10129 Turin, Italy
Email: bin.wu@polito.it, guido.masera@polito.it

April 12, 2011

Abstract

Fixed-complexity sphere decoder (FSD) is one of the most promising techniques for the implementation of multiple-input multiple-output (MIMO) detection, with relevant advantages in terms of constant throughput and high flexibility of parallel architecture. The reported works on FSD are mainly based on software level simulations and a few details have been provided on hardware implementation. In this paper, we present the study based on a four-nodes-per-cycle parallel FSD architecture with several examples of VLSI implementation in 4×4 systems with both 16-QAM and 64-QAM modulation and both real and complex signal models. Implementation aspects and details of the architecture are analyzed in order to provide a variety of performance-complexity trade-offs. We also provide a parallel implementation of log-likelihood-ratio (LLR) generator with optimized algorithm to enhance the proposed FSD architecture to be a soft-input soft-output (SISO) MIMO detector. To our best knowledge, this is the first complete VLSI implementation of an FSD based SISO MIMO detector. The implementation results show that the proposed SISO FSD architecture is highly efficient and flexible, making it very suitable for real applications.

Keywords–fixed-complexity sphere decoder, parallel architecture, MIMO detection, complex signal model, LLR generation, VLSI implementation.

1 Introduction

Multiple-input multiple-output (MIMO) system is one of the most important elements of novelty in next generation wireless communications, and provides increased data throughput and link range without additional bandwidth and transmit power [1]. It has already been included in multiple wireless communications standards, such as the IEEE 802.16e WiMax and the 3rd generation partnership project (3GPP) long term evolution (LTE) [2].

A large variety of MIMO detection techniques have been proposed, including the family of sphere decoders (SD). The originally proposed sphere decoder performs depth-first tree traversal by employing Schnorr-Euchner enumeration (SESD) [3]. This technique achieves maximum-likelihood (ML) performance and its hardware implementation has been extensively explored in both hard and soft output versions [4] [5] [6]. Unfortunately the SESD offers an intrinsically variable throughput, which tends to decrease significantly at low signal-to-noise ratio (SNR) [4] [5]. Another disadvantage of the SESD is the sequential tree search order, which makes it difficult to employ parallel architectures to improve the throughput. Different forms of pipelining have been proposed to avoid this SD drawback [7] [8]. Some sub-optimal algorithms, such as the K-best algorithm and the fixed-complexity sphere decoder (FSD), are proposed to obtain constant throughput and reduce hardware complexity, with an acceptable degree of performance loss [9] [10].

Receivers with concatenated MIMO detection and channel decoding have been recently proposed using powerful error correcting codes, such as Turbo and low-density parity-check (LDPC) codes in order to achieve near-capacity performance [11]. Soft information could be obtained by extending the existing hard-output sphere decoders, which could be SESD, K-best SD or FSD, into a list sphere decoder (LSD), which generates a list of candidates instead of the only ML solution. Then the log-likelihood-ratios (LLR) are evaluated based on the list and the *a-priori* information. Finally the LLRs are forwarded to the channel decoder [9] [12] [13].

Although the soft-output SESD achieves optimal performance, it still suffers from variable throughput [14], and the performance drops dramatically when the number of visited nodes is bounded in order to limit the complexity and the latency in hardware

implementation [15]. The recently proposed soft-output single tree search (STS) SD is also based on the depth-first SESD algorithm. It can be implemented efficiently, and achieves excellent throughput at medium to high SNR values, but it is much less efficient at low SNR region [6]. Furthermore, it is difficult to map these algorithms to highly parallel architectures, because of the natural sequential search order.

The sub-optimal algorithms, such as the K-best and the FSD, could also be improved into soft-output versions, which guarantee constant throughput at the cost of a certain performance loss. The K-best algorithm keeps a certain number (*i.e.* K) of best nodes in each level while traversing the tree, by applying sorting operations, which require additional hardware resources [9]. The FSD algorithm also achieves constant throughput but with relatively lower hardware complexity. The most outstanding feature of FSD is the regular tree traversal path, which enables the design of highly-efficient parallel architectures [10].

The reported works on FSD are mainly based on software level simulations and most of investigations on hardware are implemented on FPGA devices, such as the FPGA prototypes with pipeline architectures reported in [16] and [17]. These FPGA prototypes achieve very high throughput, but at a very high cost of occupied hardware resources, making them impractical in real applications.

In this paper, we improve a four-nodes-per-cycle parallel FSD architecture, which is implemented in real signal model as reported in [18], into complex signal model, and present a panorama of FSD implementations considering different types of modulation and both real and complex signal models, in order to provide a variety of performance-complexity trade-offs for different practical situations. We find that the four-nodes-per-cycle FSD architecture can be efficiently implemented, especially in complex signal model, in terms of throughput per area unit.

The LSD acts only as a list generator in the SISO MIMO detection and an LLR generator is required to iteratively refine LLR for each codeword bit based on the candidate list and on the feedback information received from the channel decoder [9] [12] [13]. The LLR generator usually shows comparable computational complexity as LSD, but it did

not attract enough attention by researchers. In this paper we also provide a parallel implementation of LLR generator in order to construct an efficient SISO MIMO detector combined with the four-nodes-per-cycle FSD architecture.

This paper is organized as follows: Section 2 introduces the system model of sphere decoders; Section 3 details the four-nodes-per-cycle architecture together with several implementation aspects and the parallel implementation of LLR generator; Section 4 shows the implementation results and compares the overall performance among the FSD implementations and other SD implementations; Section 5 concludes the paper.

2 System Model and Sphere Decoding Algorithm

The diagram of iterative MIMO decoding system with N_t transmit antennas and N_r receive antennas is shown in Fig. 1. The source bits are firstly encoded by a channel encoder, which could be for instance a Turbo encoder [19] or an LDPC encoder [20]. Then the coded bit stream is interleaved in order to overcome correlated channel noise. The interleaved bits are mapped into an N_t-dimensional transmit signal vector $\mathbf{s} = [s_{N_t-1}, \ldots, s_1, s_0]^T$. Each symbol of the vector is chosen independently from a complex constellation Ω with M binary bits per symbol, *i.e.*, $|\Omega| = 2^M$. The transmission rate is denoted as $R = N_t M$ bits per channel use (bpcu). The received vector can be denoted as

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{n}, \tag{1}$$

where $\mathbf{H}$ is the $N_r \times N_t$ complex channel matrix, assumed to be perfectly known at the receiver through channel estimation, $\mathbf{n} = [n_{N_r-1}, \ldots, n_1, n_0]^T$ is an N_r-dimensional complex additive i.i.d. (independent and identically distributed) white Gaussian noise vector.

In Fig. 1, the SISO MIMO detector is employed to generate soft information which is required by the concatenated channel decoder. The SISO MIMO detector could be an LSD or employ other MIMO detection algorithms, such as the STS SD or the minimum mean square error (MMSE) decoder [21]. In this paper we assume LSD is employed,

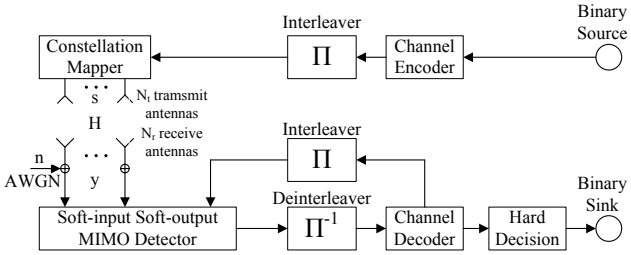

Figure 1: Iterative MIMO decoding system.

which generates a candidate list in order to compute LLRs. Soft information is exchanged between the inner MIMO detector and the outer channel decoder through a pair of interleaver and deinterleaver. After a certain number of iterations, decoded bit stream is available through hard decision depending on the required throughput or bit-error-rate (BER) performance [11].

In the transmit symbol vector constellation Ω^{N_t} there exists an ML solution that can be expressed as

$$\mathbf{s}^{ML} = \arg \min_{\mathbf{s} \in \Omega^{N_t}} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2. \tag{2}$$

$\mathbf{s}^{ML}$ can be obtained through exhaustive search in a small MIMO system, such as a 4 $\times$ 4 system with QPSK modulation (totally $2^{2\times4} = 256$ possible solutions) [22]. But for a large system, it is impractical to perform exhaustive search, because of the very large number of possible solutions to be examined [23]. For example in a 4 $\times$ 4 system with 16-QAM modulation, there are $2^{4\times4} = 65,536$ possible solutions. The sphere decoders are therefore proposed to reduce the computational complexity by formulating the calculation of (2) as a tree visit problem and applying certain pruning criteria to decrease the number of visited nodes [24].

A likelihood metric, generally in form of partial Euclidean distance (PED), is evaluated for each visited node in order to prune the tree or determine the traversal path. For hard-output sphere decoders, the transmit vector with minimum PED is chosen as the final solution [4], whereas for LSD, a certain number of visited tree leaves with the lowest PED values are sent to an LLR generator to compute the soft output information [10].

2.1 Complex Signal Model

Before the decoding stage, the preprocessing with QR decomposition is applied as

$$\mathbf{H} = \mathbf{QR}, \tag{3}$$

where $\mathbf{Q}$ is $N_r \times N_r$ complex orthogonal matrix, $\mathbf{R}$ is $N_t \times N_r$ complex upper triangular matrix with positive diagonal elements. The QR decomposition is of critical importance in many MIMO detection algorithms to reduce the complexity of receivers and to get better decoding results [25].

Then the PED for each visited node is given by

$$d_i = d_{i+1} + |e_i|^2, i = N_t - 1, N_t - 2, \ldots, 0, \tag{4}$$

where

$$e_i = y_i^{ZF} - \sum_{j=i}^{N_t-1} R_{i,j} s_j = b_i - R_{i,i} s_i, \tag{5}$$

$$\mathbf{y}^{ZF} = \mathbf{Q}^H \mathbf{y}, \tag{6}$$

$$b_i = y_i^{ZF} - \sum_{j=i+1}^{N_t-1} R_{i,j} s_j. \tag{7}$$

$|e_i|^2$ is the increment of PED in the i^{th} level and $\mathbf{y}^{ZF}$ is the zero-forcing solution [4].

2.2 Real Signal Model

As an alternative to the described complex signal based processing, the SD algorithm can be performed also in a real-valued signal model, by transforming the complex channel matrix into real values:

$$\hat{\mathbf{H}} = \begin{bmatrix} Re\{\mathbf{H}\} & -Im\{\mathbf{H}\} \\ Im\{\mathbf{H}\} & Re\{\mathbf{H}\} \end{bmatrix}, \tag{8}$$

$$\hat{\mathbf{y}} = \begin{bmatrix} Re\{\mathbf{y}\} \\ Im\{\mathbf{y}\} \end{bmatrix}, \tag{9}$$

$$\hat{\mathbf{s}} = [\hat{s}_{2N_t-1}, \ldots, \hat{s}_1, \hat{s}_0]^T = \begin{bmatrix} Re\{\mathbf{s}\} \\ Im\{\mathbf{s}\} \end{bmatrix}, \tag{10}$$

where $Re\{x\}$ and $Im\{x\}$ denote the real and imaginary parts of x, respectively.

After the transformation, the $N_r \times N_t$ dimensional channel matrix $\mathbf{H}$ turns to $2N_r \times 2N_t$ dimensional. Then the QR decomposition is applied on real values:

$$\hat{\mathbf{H}} = \hat{\mathbf{Q}}\hat{\mathbf{R}}, \tag{11}$$

where $\hat{\mathbf{Q}}$ is $2N_r \times 2N_r$ real-valued orthogonal matrix, $\hat{\mathbf{R}}$ is $2N_t \times 2N_r$ real-valued upper triangular matrix with positive diagonal elements.

The extended matrix dimensions imply that the number of tree levels is doubled and the PED calculation in (4) $\sim$ (7) can be expressed as

$$d_i = d_{i+1} + |e_i|^2, i = 2N_t - 1, 2N_t - 2, \ldots, 0, \tag{12}$$

where

$$e_i = \hat{y}_i^{ZF} - \sum_{j=i}^{2N_t-1} \hat{R}_{i,j}\hat{s}_j = b_i - \hat{R}_{i,i}\hat{s}_i, \tag{13}$$

$$\hat{\mathbf{y}}^{ZF} = \hat{\mathbf{Q}}^H \hat{\mathbf{y}}, \tag{14}$$

$$b_i = \hat{y}_i^{ZF} - \sum_{j=i+1}^{2N_t-1} \hat{R}_{i,j}\hat{s}_j. \tag{15}$$

For each vector in the transmit constellation, the calculated PED values are same in either real and complex signal models. However, it should be noticed that the choice of signal model could significantly impact on the BER performance, the throughput and the hardware complexity of sphere decoders.

2.3 Depth-first SESD

Before describing the FSD, firstly we look at the fundamental depth-first SESD. The SESD performs tree traversal in depth-first style and prunes the tree by comparing the PED of each visited node with current radius of the sphere, which is set to infinity at the beginning and is updated during the traversal. The tree traversal moves downwards from the top level, as shown in Fig. 2, with the example of 2×2 system with QPSK modulation; in this case, the tree has two levels and each node in the top level has four child nodes. The dark nodes numbered as 1 to 6 denote the visiting order. The SESD starts the depth-first tree traversal by choosing firstly the child node with minimum PED (node 1) and moves downwards. When reaching the bottom level, the minimum PED of the child nodes is compared with the current radius. If the PED is smaller than the radius, the radius is updated with the value of the PED (nodes 2) and the other child nodes are discarded. Then the SESD returns to the sibling nodes in the upper level and performs the same procedure. If the PED of the node is smaller than the current radius (node 3), the tree traversal goes downwards (node 4), otherwise the whole branch under the node is pruned (nodes 5 and 6). By applying the tree pruning, the total number of visited nodes is significantly reduced compared with the exhaustive search, while still yielding ML performance.

The soft-output SESD performs the similar depth-first tree traversal as the hard-output SESD. The difference is that the radius will not be updated until a required number of candidates are obtained, and sorting operations are necessary to insert new candidates into the list. After the tree traversal, the final candidate list is sent to an LLR generator to compute the soft information.

The SESD needs to reach the bottom tree level immediately, in order to update the radius and perform tree pruning: the depth-first sequential order makes it hard to adopt parallel processing architectures. The resulting throughput is variable and tends to drop down significantly at low SNR region. Furthermore, the performance becomes much worse when the number of visited nodes is bounded with the purpose of reducing the detection complexity and latency in real applications [15]. In order to obtain a constant throughput,

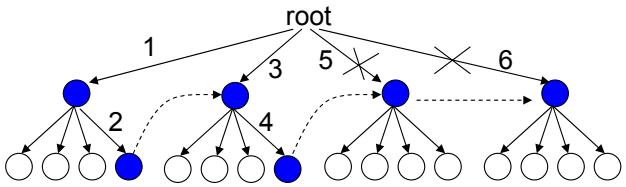

Figure 2: Example of depth-first SESD.

a constant number of nodes must be visited. Sub-optimal detection methods have been proposed to achieve this result and the FSD technique is one of the most interesting ones.

3 FSD Architecture

The FSD algorithm is also based on PED calculation but does not need to update a radius to prune the tree. Both the number and the positions of the nodes to be visited in each level are pre-determined before decoding, depending on the so-called node distribution. According to its definition, the processing complexity of FSD is fixed, therefore yielding a constant throughput. Furthermore, because it is not necessary to reach the bottom level immediately to update the radius, the traversal order is very flexible, which can be either in depth-first or in breadth-first style. One of the most outstanding advantages of the FSD is the flexibility to adopt parallel architectures for its deterministic and regular tree traversal path. Several pipeline architectures implemented on FPGA devices have been reported to achieve very high throughput, however, they also consume very large amount of hardware resources [16] [17]. However we find that highly area efficient implementation of FSD is possible with the four-nodes-per-cycle parallel architecture.

3.1 Four-nodes-per-cycle Architecture

The four-nodes-per-cycle FSD architecture proposed in [18] employs breadth-first tree traversal in order to shorten the critical path. The three major computational tasks, including b_i calculation, direct enumeration (DE) and d_i calculation, are distributed on three different groups of nodes in each clock cycle. The tree traversal order is shown in Fig. 3 in 4×4 system with 16-QAM modulation, complex signal model and node distribution

{1, 1, 1, 16}, which means that all the 16 nodes are visited in the top level, and only one child node with minimum PED is chosen in the lower levels for each survived parent node. Each group of four nodes (denoted as $G_{level,column}$) in dashed blocks are processed in parallel according to the breadth-first order. This arrangement of groups of nodes processed in parallel can be also applied to other cases, for instance the real signal model with different number of antennas and list size.

The block scheme of the four-nodes-per-cycle FSD architecture is shown in Fig. 4, for a 4×4 system with 16-QAM modulation and complex signal model. The signals with index *crt* are referred to the current cycle, while those with index *prv* are related to the previous cycle. The signal lines with slashes are referred to four different values for each of the four nodes in a certain group. Each of the main arithmetic tasks employs four units with identical internal structure, in order to process four nodes in parallel per clock cycle. As shown in Fig. 3, the tree traversal paths in FSD are highly regular, which insures that the nodes being processed in each cycle have definite positions. The FSD performs the arithmetic tasks in each cycle on different groups of nodes according to their positions, which are controlled by a level counter and a column counter in the control unit.

The b_i units are responsible for calculating b_i in (7). When computing b_i, because both real and imaginary parts of s_i is chosen from a small set $\{+3, +1, -1, -3\}$ for 16-QAM modulation, the multiplications between $R_{i,j}$ and s_i can be transformed into additions between $\pm 2Re\{R_{i,j}\}$, $\pm Re\{R_{i,j}\}$, $\pm 2Im\{R_{i,j}\}$ and $\pm Im\{R_{i,j}\}$ (with additional $\pm 4Re\{R_{i,j}\}$, $\pm 8Re\{R_{i,j}\}$, $\pm 4Im\{R_{i,j}\}$, $\pm 8Im\{R_{i,j}\}$ for 64-QAM modulation). The values of $\pm 2Re\{R_{i,j}\}$ and $\pm 2Im\{R_{i,j}\}$ can be easily obtained through left shifting operation. In order to shorten the delay path, all these values are compressed into a Wallace compress tree of carry save adder (CSA), which is followed by a common ripple carry adder (RCA) [26].

The DE units are employed to select child nodes to be visited. Because all the candidates are derived from a common parent (*i.e.* with the same d_{i+1}), the solution is to compare $|e_i|$ among all the child nodes and choose the node with minimum $|e_i|$. Since b_i is already calculated in the previous cycle, the task of DE unit is simply comparing

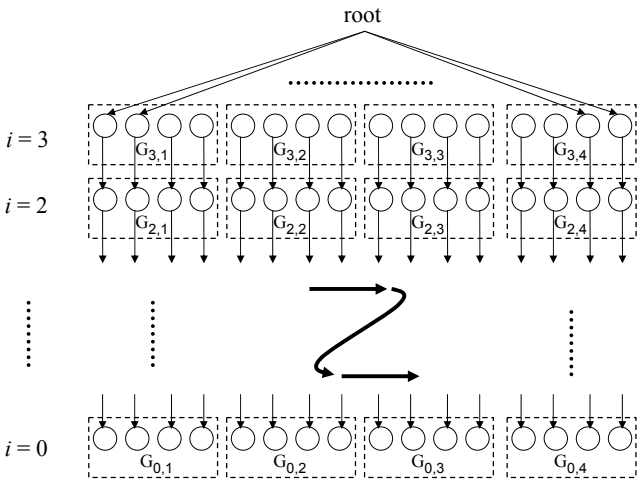

Figure 3: Tree traversal order of the four-nodes-per-cycle architecture in 4 × 4 system with 16-QAM modulation, in complex signal model.

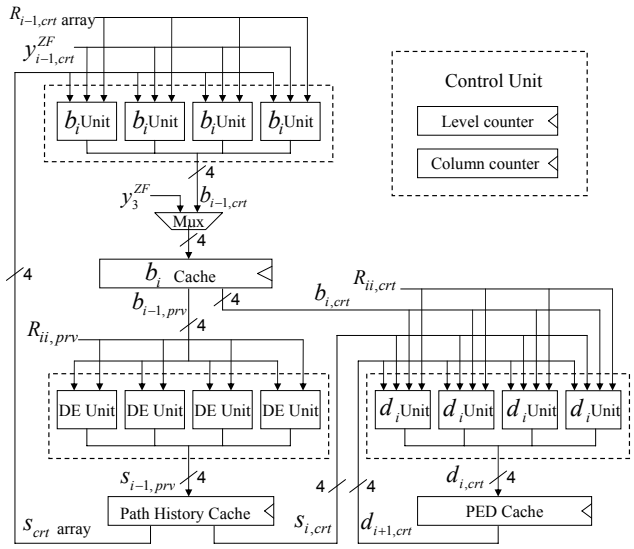

Figure 4: Diagram of the four-nodes-per-cycle FSD architecture in 4 × 4 system with 16-QAM modulation, in complex signal model.

$|b_i - R_{i,i}s_i|$ among the child nodes. However this solution results in a large hardware complexity. The complexity can be reduced by applying the method introduced in [17]. The modulation constellation is divided into several small parts and the proper s_i is chosen by comparing the real and the imaginary parts of b_i with the threshold values 0, $\pm 2Re\{R_{i,i}\}$ and $\pm 2Im\{R_{i,i}\}$ that define the boundaries of each symbol.

The d_i units compute the PED for each visited node. The calculation of d_i is implemented in the straightforward way according to (12). The complex value of e_i is firstly

calculated by adding $R_{i,i}s_i$ to b_i. Then two multipliers are employed to calculate the square of both real and imaginary parts, followed by an addition with d_{i+1} (PED of the parent node).

The four-nodes-per-cycle architecture features scalable parallelism which varies in the range between the common one-node-per-cycle architecture and the pipeline architectures, depending on the system size (*i.e.* number of antennas and modulation type) and the performance requirements (*e.g.* throughput or BER) of targeted wireless communications standards. For example, it can be extended into an eight-nodes-per-cycle version by doubling the arithmetic units, including the b_i units, the DE units and the d_i units, in order to provide nearly doubled throughput.

3.2 Complexity of FSD

The complexity of LSD is usually mentioned in terms of the number of visited nodes, which is mainly affected by the following factors: number of transmit and receive antennas, modulation type, required list size, the choice of real or complex signal model, and the node distribution specifically for FSD.

Because the number of tree levels is doubled when the real signal model is adopted compared with the complex signal model, the number of visited nodes will be increased significantly with the same list size. For example, in 4×4 system with 16-QAM modulation and list size = 16, totally 116 nodes are visited in real signal model with node distribution {1, 1, 1, 1, 1, 1, 4, 4}. To the contrary, only 64 nodes are visited in complex signal model with node distribution {1, 1, 1, 16}, approximately only half of the number in real signal model.

From another point of view, the implementation in complex signal model requires more computational resources, such as adders and multipliers, because both real and imaginary parts should be calculated in (4) $\sim$ (7) at the same time [17]. Several works have been reported considering the choice of real or complex signal model. Burg *et al.* argue that performing sphere decoding directly on the complex constellation is more efficient in VLSI implementations [4]. To the contrary, Myllylä *et al.* show that the real valued

algorithms are less complex and feasible for practical LSD implementation [15]. However, a quantitative comparison based on fully synthesized hardware systems and extended to several MIMO systems between the two alternatives is currently not available. We will see later that the complex implementation achieves higher efficiency in terms of throughput per area unit, which is more meaningful compared with considering only on the Silicon area.

Some channel ordering algorithms can be applied to improve the performance. We find that even a simple sorted QR decomposition (SQRD) improves the performance significantly. The SQRD algorithm reorders the sequence of detection to minimize the risk of error propagation, by maximizing $|R_{i,i}|$ for $i = N_t - 1, N_t - 2, \ldots, 0$ [29]. Therefore the transmit antennas with strongest signal is assigned to the levels closer to the root of the tree [25].

In addition to the signal models, the node distribution also affects both performance and complexity of FSD. Although it is difficult to provide a comprehensive analysis of the node distribution to achieve optimal performance, a general method proposed in [27] could be used for an arbitrary constellation and for any number of antennas. The number of visited child nodes of each parent node in one level can be chosen from a small set $\{1, N_b\}$, where N_b is the number of branches per node. In the top levels, a full search is performed by visiting all the N_b branches, whereas in the lower levels, only one branch is chosen as survivor [28]. When combined with the SQRD, this method guarantees that the nodes with highest signal-to-noise ratio (SNR) are firstly visited. In this paper we adopt this kind of node distribution together with the SQRD to maximize the BER performance, with an exception for the case with 64-QAM modulation and list size = 128. It is implemented with the node distribution {1, 1, 2, 64}, which also follows the basic rule that expands the paths in the top levels.

3.3 Parallel LLR Generator

After the sphere decoding, the candidate list is forwarded to the LLR generator. Then the LLR for each coded bit is evaluated based on the candidate list and the *a-priori*

information vector $\mathbf{L_A}$ coming from the outer channel decoder. Let $L_{E,k}$ denote the extrinsic information, *i.e.*, LLR, for the k^{th} bit in the coded bit vector, $k = R-1, \ldots, 1, 0$. The max-log approximation of $L_{E,k}$ calculation is formulated according to [11]

$$\begin{aligned} L_{E,k} \approx & \frac{1}{2} \max_{\mathbf{x}_i \in \mathbf{X}_{k,+1}} \{-\frac{d_i}{\sigma^2} + \mathbf{x}_{i,[k]}^T \cdot \mathbf{L}_{\mathbf{A},[k]}\} \\ & -\frac{1}{2} \max_{\mathbf{x}_i \in \mathbf{X}_{k,-1}} \{-\frac{d_i}{\sigma^2} + \mathbf{x}_{i,[k]}^T \cdot \mathbf{L}_{\mathbf{A},[k]}\}, \end{aligned} \tag{16}$$

where $\mathbf{X}_{k,+1}$ and $\mathbf{X}_{k,-1}$ represent the sets of vector $\mathbf{x}_i$ having $x_{i,k} = +1$ and $x_{i,k} = -1$ respectively, σ^2 is the noise variance, $\mathbf{x}_{i,[k]}^T$ is the sub-vector of $\mathbf{x}_i^T$ omitting the bit $x_{i,k}$, and $\mathbf{L}_{\mathbf{A},[k]}$ is the sub-vector of the *a-priori* information vector $\mathbf{L_A}$ with omitted $L_{A,k}$, $k = R-1, \ldots, 1, 0$ and $i = P-1, \ldots, 1, 0$ (P denotes the list size).

The whole computation task shows considerable complexity $O(PR^2)$ in terms of additions. The LLR generation methods reported in [12] and [30] are soft-output only and do not support iterative decoding with channel decoder, whereas the strategy of mixed STS and LLR generation reported in [6] is soft-input soft-output but can not be applied to the LSD. In this paper we optimize the algorithm in (16) by reusing intermediate data and reduce the complexity of LLR generation to $O(PR)$.

Let $T_i^k = \mathbf{x}_{i,[k]}^T \cdot \mathbf{L}_{\mathbf{A},[k]}$ and $T_i = \mathbf{x}_i^T \cdot \mathbf{L_A}$, then we have

$$T_i = T_i^k + x_{i,k} L_{A,k}, \tag{17}$$

where $i = P-1, \ldots, 1, 0$.

When evaluating LLRs as expressed in (16) in a straightforward way, T_i^k needs to be calculated for PR times. For different coded bits, T_i^k can be expressed as the difference between T_i and $x_{i,k}L_{A,k}$:

$$T_i^k = T_i - x_{i,k} L_{A,k}. \tag{18}$$

Therefore the approach for reducing complexity is to get T_i for each of the P candidates at the beginning of processing, with only P accumulations required for all T_i. Then T_i could be reused in the following procedure for evaluating LLRs for different coded bits,

and get T_i^k by subtracting $x_{i,k}L_{A,k}$ from T_i. Pseudo code of the optimized LLR generation is given in Fig. 5. For simplicity, we denote $Q_i = T_i - d_i/\sigma^2$ and $Q_i^k = T_i^k - d_i/\sigma^2$.

Based on the optimized LLR generation algorithm, we designed several compact functional blocks in order to build an efficient LLR generator, which is highly parallel and scalable to improve further the throughput.

The architecture of the LLR generator is shown in Fig. 6. The implementation is targeted to the 4×4 MIMO systems with 16-QAM modulation ($R = P = 16$). P accumulation units (ACC_UNIT_i) concurrently calculate Q_i for each of the candidates in the list. The comparison stage to get final $L_{E,k}$ is performed by the R comparison units (COM_UNIT_k). Each of them evaluates $L_{E,k}$ for the k^{th} coded bit. Pipeline registers are allocated between the ACC_UNITs and the COM_UNITs in order to improve the throughput. Based on the fixed-point evaluation, the input data width is chosen to be 12 bits for d_i, 8 bits for $(1/\sigma^2)$ and 8 bits for $L_{A,k}$, but the internal data width is always 12 bits for T_i, Q_i and Q_i^k to avoid overflow. The output $L_{E,k}$ is clipped to 8 bits, the same width as for the input $L_{A,k}$.

The ACC_UNITs are responsible to compute $Q_i = \mathbf{x}_i^T \cdot \mathbf{L_A} - d_i/\sigma^2$. The value of $(1/\sigma^2)$ is assumed to be ready for use through channel estimation. To reduce complexity, the multiplication $(1/\sigma^2) \times d_i$ is transformed into a series of addition and shift operations, which are executed in parallel with the accumulation $T_i = \mathbf{x}_i^T \cdot \mathbf{L_A}$. The accumulation of T_i requires totally R cycles for all the R coded bits. One additional cycle is needed to subtract $(1/\sigma^2) \times d_i$ from T_i. Then the accumulation stage is finished and Q_i is ready for the following comparison stage. Therefore totally $(R + 1)$ cycles are required by the accumulation stage.

The COM_UNITs evaluate the LLRs, *i.e.*, $L_{E,k}$ for each coded bit. Firstly Q_i is selected and subtracted from $x_{i,k}L_{A,k}$ to get Q_i^k. Then Q_i^k is classified into two lists, *i.e.*, `POS_LIST` and `MIN_LIST` in Fig. 5, which contain Q_i^k for the candidates with $x_{i,k} = +1$ or $x_{i,k} = -1$ respectively. If Q_i^k is moved into one of the two lists, it is compared with the current value in the corresponding register, which is set to be the minimum possible value `LLR_MIN` at the beginning. When Q_i^k is greater than the current registered value, The

```
//Accumulation stage
for (i=P-1;i>=0;i--) {
   Calculate T_i = x_i^T · L_A;
   Calculate Q_i = T_i − (1/σ^2)×d_i;
}
//Comparison stage
for (k=R-1;k>=0;k--) {
   for (i=P-1;i>=0;i--) {
      Calculate Q_i^k = Q_i − x_{i,k}×L_{A,k};
      if (x_{i,k} = +1)
         Move Q_i^k into POS_LIST;
      else if (x_{i,k} = −1)
         Move Q_i^k into MIN_LIST;
   }
   MAX_SEARCH in POS_LIST for LLR_POS;
   MAX_SEARCH in MIN_LIST for LLR_MIN;
   L_{E,k} = (LLR_POS − LLR_MIN)/2;
}
```

Figure 5: Pseudo code for the optimized LLR generation.

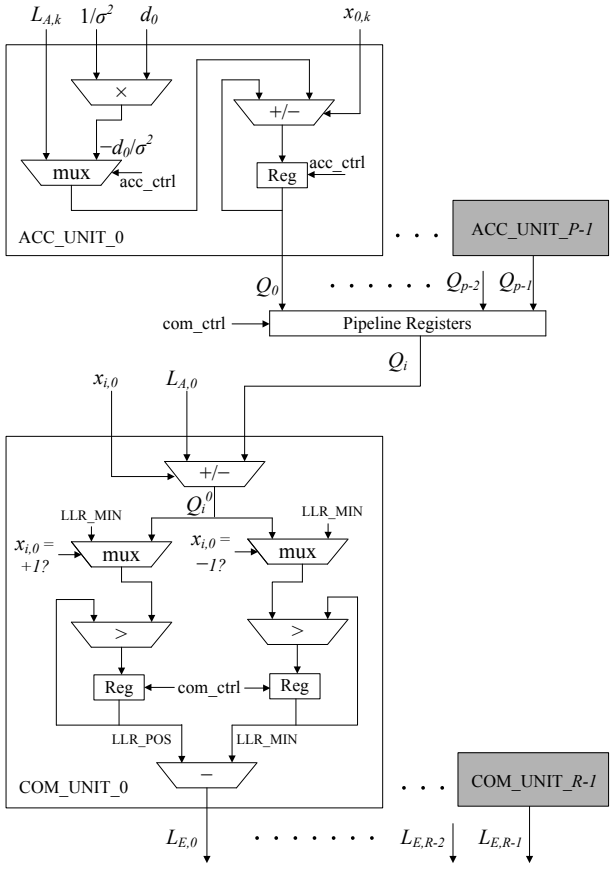

Figure 6: Parallel architecture of the LLR generator.

register is updated with Q_i^k. For the opposite list, `LLR_MIN` is employed to compare with the registered value. It actually performs a `NOP` operation and the registered value stays unchanged. After P cycles the comparison among Q_i^k is finished. The registers contain the maximum Q_i^k in each of the two lists respectively, which are denoted as `LLR_POS` and

Table 1: Silicon complexity and breakdown of the five FSD implementations

	FSD 1	**FSD 2**	**FSD 3**	**FSD 4**	**FSD 5**
b_i units (KG)	11.6	12.1	49.9	43.0	44.6
DE units (KG)	0.5	0.8	2.3	4.8	5.0
d_i units (KG)	2.9	5.5	10.0	15.0	15.5
Reg & ctrl (KG)	9.2	9.1	33.8	36.2	74.3
Total (KG)	24.2	27.5	96.0	99.0	139.4

`LLR_MIN` in Fig. 5. The difference between `LLR_POS` and `LLR_MIN` is the final output $L_{E,k}$.

4 Implementation Results and Analysis

We improve the four-nodes-per-cycle FSD architecture described in [18] into complex signal model to validate the impacts on implementation efficiency, and further extend the architecture into eight-nodes-per-cycle version to provide doubled throughput for 64-QAM modulation. For 16-QAM and 64-QAM modulation, the four- and the eight- nodes-per-cycle architectures are employed respectively. For simplicity, all the five cases are denoted as FSD 1 $\sim$ FSD 5 in the following parts:

FSD 1: 16-QAM, list size = 16, real model;

FSD 2: 16-QAM, list size = 16, complex model;

FSD 3: 64-QAM, list size = 64, real model;

FSD 4: 64-QAM, list size = 64, complex model;

FSD 5: 64-QAM, list size = 128, complex model.

Communications performance of the four-nodes-per-cycle FSD is given in Fig. 7, in 4 $\times$ 4 system with 16-QAM modulation, coupled with a four state, 1 / 3 code rate Turbo decoder, which executes 8 decoding iterations; two iterations between inner MIMO detector and outer Turbo decoder are performed. The performance of FSD with list size = 16 is given in both real and complex signal model and the other cases are all in complex signal model. List size is 16 for SESD. For K-best SD, $K = 8$ and $K = 12$ are adopted. We can see that both FSD and K-best SD exhibit performance gaps compared with the optimal SESD algorithm. The gaps can be compensated in certain degree by increasing

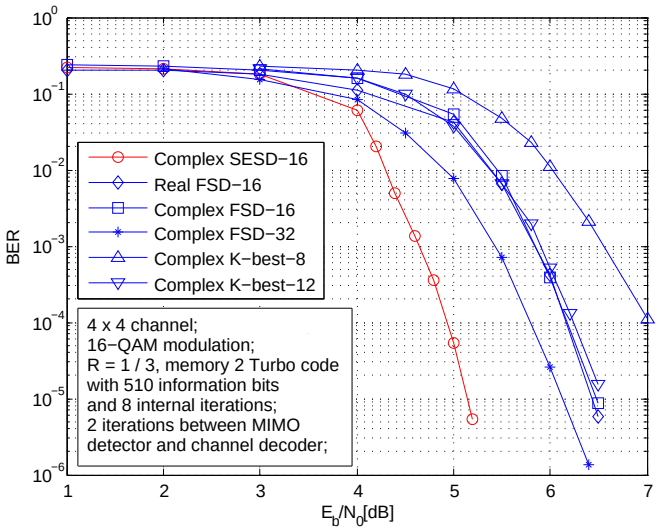

Figure 7: Performance comparison among SESD, K-best SD and FSD.

the list size or K with higher complexity, as shown between FSD-16 and FSD-32, and between K-best-8 and K-best-12. The BER curves for real and complex FSD are almost overlapped and the performance of FSD-16 is slightly higher than K-best-12 at BER = 10^{-5}. However, the complexity of K-best-12 is much higher compared with FSD-16 in terms of visited nodes. The difference comes from the tree traversal strategy of the two algorithms. The K-best SD needs to compute the PED for all the child nodes among the K parents followed by a complicated sorting network to select the K survived nodes in each level, but for the FSD, because the child nodes are selected only among the siblings with a common parent, it is not necessary to compute the PED for all child nodes. For example, in Fig. 7, the complex FSD with list size = 16 needs to visit only $(4 \times 16) = 64$ nodes, but the K-best SD with $K = 12$ needs to visit $(16+3\times12\times16) = 592$ nodes, which is much higher compared with the FSD. Therefore we conclude that among sub-optimal algorithms, the FSD algorithm is more efficient than the K-best algorithm.

The five cases of FSD implementation are synthesized on the same 0.13 μm CMOS technology. The Silicon complexity and breakdown is shown in Table 1. The Silicon area is measured in terms of gate equivalents (KG). We can see that the Silicon area of complex FSD increases slightly compared with real FSD with the same modulation and list size, by only 3 KG for both 16-QAM and 64-QAM modulation. However, the area of FSD 5 increases by 40 KG compared with FSD 4. The area of all the computational blocks

(b_i units, DE units and d_i units) remains approximately the same in FSD 5 compared with FSD 4, but the proportion of registers is increased sharply, nearly doubled. It can be concluded that for FSD the area increment comes mainly from the increased list size, with nearly negligible impact from the choice of real or complex model.

The b_i units occupy the largest proportion in the whole area. When moving from real signal model to complex model, two identical carry save adder (CSA) trees are employed to calculate both the real and the imaginary parts of b_i. However, the number of required CSAs for each of the CSA trees is significantly reduced because of the reduced number of tree levels, making the overall proportion of b_i units even decrease by a small amount.

The DE units need to select both real and imaginary parts of s_i, so the occupied area is increased. But it does not impact on the total area too much because the DE units share only a very small proportion (less than 3% for 16-QAM modulation, list size = 16) in the whole architecture.

The d_i units need more computational resources in complex signal model because e_i is a complex value and both real and imaginary parts should be considered.

The same number of registers are required by both s_i and d_i, in real and complex signal models, but a doubled number of registers are required by b_i in complex model. The control circuit remains approximately the same in both real and complex signal models.

Table 2 shows the available throughput of the five cases. Although the implementations in real and complex signal models achieve the same processing speed in terms of visited nodes per second (1.60 G in the four-nodes-per-cycle architecture and 3.08 G in the eight-nodes-per-cycle architecture), there are large gaps between the throughput in terms of coded bits per second, because the total numbers of visited nodes are quite different between real and complex signal models. Hardware efficiency is given in terms of throughput per area unit. We can see that either for 16-QAM or 64-QAM modulation, implementations in complex signal model achieve higher values with the same list size.

Communications performance and implementation efficiency of the five implementations of FSD are shown in Fig. 8. The BER performance is given through simulations in

Table 2: Throughput comparison among the five FSD implementations

	FSD 1	**FSD 2**	**FSD 3**	**FSD 4**	**FSD 5**
Visited nodes	116	64	456	256	448
Max. clock freq. (MHz)	400	400	384.6	384.6	384.6
Nodes/second (G)	1.60	1.60	3.08	3.08	3.08
Throughput (Mbps)	213.3	376.5	159.1	279.7	161.9
Efficiency (Mbps/KG)	8.81	13.69	1.66	2.83	1.16

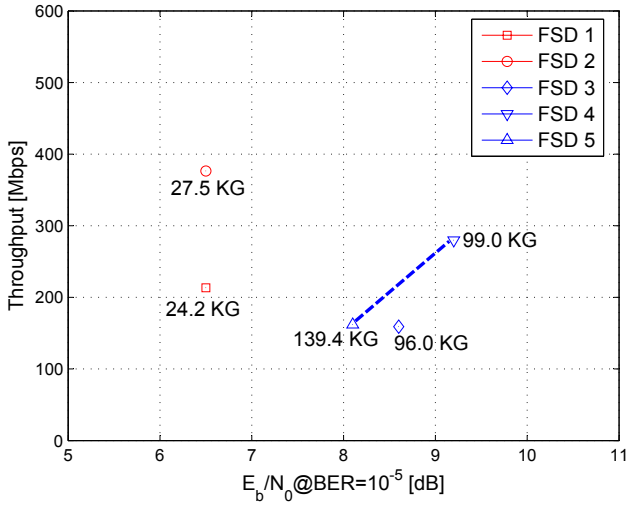

Figure 8: Performance and Silicon complexity of the five FSD implementations.

4×4 system, coupled with a four state, 1 / 3 code rate Turbo decoder, which executes 8 decoding iterations [30]. SQRD is applied to all the five cases and two iterations between soft-output FSD and Turbo decoder are performed.

For 16-QAM modulation and list size = 16, the SNR at BER = 10^{-5} is 6.5 dB for both real and complex FSD. The complex FSD achieves almost twice of the throughput compared with the real FSD, but occupies only 13% more area. It is obvious that the complex FSD is more efficient than the real FSD. For 64-QAM and list size = 64, the complex FSD achieves also almost twice of the throughput compared with the real FSD and occupies nearly the same area, however with a 0.5 dB SNR gap at BER = 10^{-5}. The performance gap can be compensated by increasing the list size, at the cost of increased area and decreased throughput.

The proposed LLR generator is also synthesized on the 0.13 μm CMOS technology. It occupies a Silicon area of 22 KG, at 500 MHz clock frequency [31]. In this implementation

Table 3: Comparison with several recently published SISO MIMO detectors

	[9]	**[6]**	**[32]**	**FSD 2**
Antenna	4×4	4×4	4×4	4×4
Modulation	16-QAM	16-QAM	16-QAM	16-QAM
Algorithm	K-best	STS	MMSE	FSD
Technology	0.13 μm	90 nm	90 nm	0.13 μm
Max. clock freq. (MHz)	200	250	568	400
Throughput (Mbps)	106.6	72@16dB	757	376.5
Area (KG)	97	96	410	49.5
Efficiency(Mbps/KG)	1.10	0.75@16dB	1.85	7.61

the accumulation stage costs 17 cycles and the comparison stage costs 16 cycles. Thanks to the pipeline architecture, the maximum throughput is determined by the stage with longer delay, *i.e.*, the accumulation stage. The achievable throughput is therefore (16×500)/18 = 444 Mbps, with an extra cycle to start a new procedure.

When combined with the four-nodes-per-cycle FSD architecture, a highly efficient SISO MIMO detector is available, which costs only 49.5 KG Silicon area for 4×4 system with 16-QAM modulation.

The implementation efficiency is compared with several recently published SISO MIMO detectors in Table 3. The FSD 2 implemented in complex signal model achieves the highest efficiency in terms of throughput per area unit. It achieves much higher throughput compared with the SISO K-best SD reported in [9] and the SISO STS reported in [6], which suffers variable throughput. The SISO MMSE detector achieves very high throughput at the cost of 410 KG Silicon area, which reduces its efficiency.

5 Conclusion

In this paper we improve a recently proposed four-nodes-per-cycle FSD architecture into complex signal model in order to verify the impacts on implementation efficiency and further extend the architecture into eight-nodes-per-cycle version to provide doubled

throughput for 64-QAM modulation. We conclude that the complex signal model is more efficient for FSD implementation compared with the real signal model in terms of throughput per area unit. For 16-QAM modulation, the complex FSD achieves nearly doubled throughput (376.5 Mbps) with the same BER performance and only 3 KG additional Silicon area compared with the real FSD. For 64-QAM modulation, thanks to the improved eight-nodes-per-cycle architecture, the complex FSD achieves 279.7 Mbps throughput. We also provide a parallel implementation of LLR generator with 444 Mbps throughput in 4×4 system with 16-QAM modulation. When combined with the LLR generator, a highly efficient SISO FSD is available.

References

[1] Paulraj, A.J., Gore, D.A., Nabar, R.U., and Bölcskei, H.: 'An overview of MIMO communications – a key to gigabit wireless'. Proc. IEEE, 2004, 92, (2), pp. 198–218

[2] TR25.876: 'Multiple-input multiple-output in UTRA (Release 7), 3GPP Std', 2006

[3] Agrell, E., Eriksson, T., Vardy, A., and Zeger, K.: 'Closest point search in lattices', IEEE Trans. Inf. Theory, 2002, 48, (8), pp. 2201–2214

[4] Burg, A., Borgmann, M., Wenk, M., Zellweger, M., Fichtner, W., and Bölcskei, H.: 'VLSI implementation of MIMO detection using the sphere decoding algorithm', IEEE J. Solid-State Circuits, 2005, 40, (7), pp. 1566–1577

[5] Cerato, B., Masera, G., and Viterbo, E.: 'Decoding the Golden code: a VLSI design', IEEE Trans. VLSI Syst., 2009, 17, (1), pp. 156–160

[6] Witte, E.M., Borlenghi, F., Ascheid, G., Leupers, R., and Meyr, H.: 'A scalable VLSI architecture for soft-input soft-output single tree-search sphere decoding', IEEE Trans. Circuit and Systems-II: Express Briefs, 2010, 57, (9), pp. 706–710

[7] Burg, A., Wenk, M., and Fichtner, W.: 'VLSI implementation of pipelined sphere decoding with early termination'. Proc. European Signal Processing Conf. (EUSIPCO 2006), Florence, Italy, September 2006

[8] Troglia Gamba, M., and Masera, G.: 'Look-ahead sphere decoding: algorithm and VLSI architecture', IET Communications, to appear

[9] Guo, Z., and Nilsson, P.: 'Algorithm and implementation of the K-best sphere decoding for MIMO detection', IEEE J. Sel. Areas Commun., 2006, 24, (3), pp. 491–503

[10] Barbero, L.G., and Thompson, J.S.: 'Extending a fixed-complexity sphere decoder to obtain likelihood information for Turbo-MIMO systems', IEEE Trans. Vehicular Tech., 2008, 57, (5), pp. 2804–2814

[11] Hochwald, B.M., and ten Brink, S.: 'Achieving near-capacity on a multiple-antenna channel', IEEE Trans. Commun., 2003, 51, (3), pp. 389–399

[12] Shin, W., Kim, H., Son, M., and Park, H.: 'An improved LLR computation for QRM-MLD in coded MIMO systems'. Proc. Conf. Vehicular Technology, Baltimore, USA, September 2007, pp. 447–451

[13] Myllylä, M., Juntti, M., and Cavallaro, J.R.: 'Implementation and complexity analysis of list sphere detector for MIMO-OFDM systems'. Proc. Conf. Signals, Systems and Computers, Pacific Grove, USA, October 2008, pp. 1852–1856

[14] Lee, J., and Park, S.-C.: 'Novel techniques of a list sphere decoder for high throughput'. Proc. Int. Conf. Advanced Communication Technology, Phoenix Park, Korea, February 2006, 3, pp. 1785–1787

[15] Myllylä, M., Juntti, M., and Cavallaro, J.R.: 'Implementation aspects of list sphere detector algorithms'. Proc. IEEE Global Telecommunications Conf., Washington, DC, USA, November 2007, pp. 3915–3920

[16] Barbero, L.G., and Thompson, J.S.: 'Rapid prototyping of a fixedthroughput sphere decoder for MIMO systems'. Proc. IEEE Int. Conf. Communications, Istanbul, Turkey, June 2006, 7, pp. 3082–3087

[17] Khairy, M.S., Abdallah, M.M., and Habib, S.E.-D.: 'Efficient FPGA implementation of MIMO detector for mobile WiMAX system'. Proc. IEEE Int. Conf. Communications, Dresden, Germany, June 2009, pp. 1–5

[18] Wu, B., and Masera, G.: 'A novel VLSI architecture of fixed-complexity sphere decoder'. Proc. Conf. Digital System Design, Lille, France, September 2010, pp. 737–744

[19] Berrou, C., Glavieux, A., and Thitimajshima, P.: 'Near Shannon limit error-correcting coding and decoding: turbo-codes'. Proc. Conf. Communications (ICC 93), Geneva, Switzerland, May 1993, pp. 1064–1070

[20] Gallager, R.G.: 'Low-density parity-check codes', IRE Trans. Inf. Theory, 1962, IT-8, (1), pp. 21–28

[21] Boher, L., Rabineau, R., and Helard, M.: 'Architecture and implementation of an iterative receiver for MIMO systems'. Proc. Int. Symposium on Turbo Codes and Related Topics, Lausanne, Switzerland, September 2008, pp. 96–101

[22] Burg, A., Felber, N., and Fichtner, W.: 'A 50 Mbps 4×4 maximum likelihood decoder for multiple-input multiple-output systems with QPSK modulation'. Proc. IEEE Int. Conf. Electronics, Circuits and Systems, Sharjah, United Arab Emirates, December 2003, 1, pp. 332–335

[23] Garrett, D., Davis, L., ten Brink, S., Hochwald, B., and Knagge, G.: 'Silicon complexity for maximum likelihood MIMO detection using spherical decoding', IEEE J. Solid-State Circuits, 2004, 39, (9), pp. 1544–1552

[24] Bölcskei, H., Gesbert, D., Papadias, C., and van der Veen, A.J. (Ed.): 'Space-time wireless systems: from array processing to MIMO communications' (Cambridge University Press, 2006)

[25] Nazar, G.L., Gimmler, C., and Wehn, N.: 'Implementation comparisons of the QR decomposition for MIMO detection'. Proc. Symposium on Integrated Circuits and System Design, New York, USA, September 2010, pp. 210–214

[26] Ienne, P., and Verma, A.K.: 'Arithmetic transformations to maximise the use of compressor trees'. Proc. IEEE Int. Workshop on Electronic Design, Test and Applications, Perth, Australia, January 2004, pp. 219–224

[27] Barbero, L.G., and Thompson, J.S.: 'Fixing the complexity of the sphere decoder for MIMO detection', IEEE Trans. Wireless Commun., 2008, 7, (6), pp. 2131–2142

[28] Jaldén, J., Barbero, L.G., Ottersten, B., and Thompson, J.S.: 'The error probability of the fixed-complexity sphere decoder', IEEE Trans. Signal Process., 2009, 57, (7), pp. 2711–2720

[29] Wübben, D., Böhnke, R., Rinas, J., Kühn, V., and Kammeyer, K.D.: 'Efficient algorithm for decoding layered space-time codes', Electronics Letters, 2001, 37, (22), pp. 1348–1350

[30] Martina, M., Nicola, M., and Masera, G.: 'A flexible UMTS-WiMax Turbo decoder architecture', IEEE Trans. Circuits and Systems II: Express Briefs, 2008, 55, (4), pp. 369–373

[31] Graziano, M., and Piccinini, G.: 'Statistical power supply dynamic noise prediction in hierarchical power grid and package networks', Integration, the VLSI Journal, 2008, 41, (4), pp. 524–538

[32] Studer, C., Fateh, S., and Seethaler, D.: 'A 757 Mb/s 1.5 mm^2 90 nm CMOS soft-input soft-output MIMO detector for IEEE 802.11n'. Proc. European Solid-State Circuits Conf. (ESSCIRC), Seville, Spain, September 2010, pp. 530–533