Chapter 5
VLSI Implementations of Sphere Detectors

Johanna Ketonen, Markus Myllylä, Yang Sun, and Joseph R. Cavallaro

5.1 Soft Detection

The multiple input multiple output (MIMO) detection problem of an uncoded system can be considered as a so-called integer least squares problem, which can be solved optimally with a hard-output maximum likelihood (ML) detector [1]. The ML detector solves optimally the so-called closest lattice point problem by calculating the Euclidean distances (EDs) between the received signal vector and points in the lattice formed by the channel matrix and the received signal, and selects the lattice point that minimizes the Euclidean distance to the received vector [2]. The ML detection problem can be solved with an exhaustive search, i.e., checking all the possible symbol vectors and selecting the closest point. The ML detector achieves a full spatial diversity with regard to the number of receive antennas; however, it is computationally very complex and not feasible as the set of possible points increases.

The received frequency domain (FD) signal can be described with the equation $\mathbf{y} = \mathbf{H}\mathbf{x} + \eta$, where $\mathbf{x} \in \mathbb{C}^N$ is the transmitted symbol vector, $\eta \in \mathbb{C}^M$ is a vector containing circularly symmetric complex Gaussian distributed noise with variance σ^2, $\mathbf{H} \in \mathbb{C}^{M \times N}$ is the frequency domain channel matrix containing complex Gaussian fading coefficients, and N is the number of transmit (TX) antennas and M is the number of receive (RX) antennas. The entries of $\mathbf{x}$ are chosen independently

J. Ketonen
Department of Communications Engineering, University of Oulu, Oulu, Finland

M. Myllylä
Nokia Networks, Nokia, Oulu, Finland

Y. Sun • J.R. Cavallaro (✉)
ECE Department, Rice University, 6100 Main St., Houston, TX 77005, USA
e-mail: cavallar@rice.edu

C. Chavet, P. Coussy (eds.), *Advanced Hardware Design for Error Correcting Codes*, DOI 10.1007/978-3-319-10569-7_5

from a complex QAM constellation Ω with sets of Q transmitted coded binary information bits $\mathbf{b} = [b_1, \ldots, b_Q]^{\mathrm{T}}$ per symbol.

The ML detector calculates the Euclidean distances (EDs) between the received signal vector $\mathbf{y}$ and lattice points $\mathbf{Hx}$, and returns the vector $\mathbf{x}$ with the smallest distance, i.e., it minimizes

$$\hat{\mathbf{x}}_{\mathrm{ML}} = \arg\min_{x \in \Omega^N} ||\mathbf{y} - \mathbf{Hx}||^2, \tag{5.1}$$

where $\mathbf{x}$ is the transmitted signal vector and $\mathbf{H}$ is the channel matrix. The ML detector performs an exhaustive search over all possible lattice points and the complexity is exponential in N.

The ML detector is optimal for uncoded systems, but for coded systems a posteriori probabilities (APP) for the decoder are required. Practical communication systems apply forward error correction (FEC) coding in order to achieve near capacity performance. The optimal way to process the spatially multiplexed and FEC coded data sequence would be to use a joint detector and decoder for the whole coded data sequence and decode the most probable data sequence. The complexity of the optimal receiver would be prohibitive as it depends on the length of the code block [3]. The optimal receiver is then approximated with an iterative receiver [4] with a separate soft-input soft-output (SfISfO) detector and soft in soft out (SISO) decoder, which exchange reliability information between the units. A structure of such a receiver is presented in Fig. 5.1.

The MAP detector provides the optimal APPs or log-likelihood ratios (LLR) [5] for the decoder. Given the interleaving of $\mathbf{b}$ and assuming the noise in the system is white Gaussian and the bits are approximately statistically independent, the a posteriori LLR for the transmitted bit k can be written as [3]

$$\begin{aligned} L_D(b_k|\mathbf{y}) &= \ln \frac{\Pr(b_k = +1|\mathbf{y})}{\Pr(b_k = -1|\mathbf{y})} \\ &= L_A(b_k) + \ln \frac{\sum_{\mathbf{b} \in \mathcal{L}_{k,+1}} \exp(-\frac{1}{2\sigma^2}||\mathbf{y} - \mathbf{Hx}||^2 + \frac{1}{2}\mathbf{b}_{[k]}^T \mathbf{L}_{A,[k]})}{\sum_{\mathbf{b} \in \mathcal{L}_{k,-1}} \exp(-\frac{1}{2\sigma^2}||\mathbf{y} - \mathbf{Hx}||^2 + \frac{1}{2}\mathbf{b}_{[k]}^T \mathbf{L}_{A,[k]})}, \end{aligned} \tag{5.2}$$

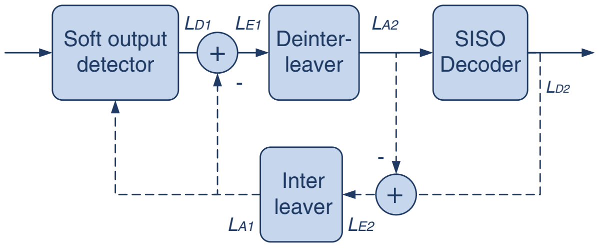

Fig. 5.1 The iterative receiver

where $\mathcal{L}_{k,+1} \bigcap \mathbb{B}_{k,+1}$ is a list of candidate points $\mathbf{x}$. $\mathbb{B}_{k,a}$ is the set of 2^{NQ-1} bit vectors having $b_k = a, a \in \{-1, 1\}$, $\mathbf{b}_{[k]}$ is a subvector of $\mathbf{b}$ without b_k, and vector $\mathbf{L}_{A,[k]}$ includes all L_A values except for b_k. The list $\mathcal{L}$ can be obtained by neglecting the insignificant elements in $\mathbb{B}$ such that the K candidate points in $\mathcal{L}$ include $\hat{\mathbf{x}}_{\text{ML}}$ and $2^{MQ} > K \geq 1$ [3]. This can be achieved for example with a list sphere detector (LSD).

The approximation of the logarithm in (5.2) can be calculated using a small look-up table and the Jacobian logarithm [6]

$$\text{jacln}(a_1, a_2) := \ln(e^{a_1} + e^{a_2}) = \max(a_1, a_2) + \ln(1 + \exp(-|a_1 - a_2|)). \quad (5.3)$$

The Jacobian logarithm in (5.3) can be computed without the logarithm or exponential functions by storing $r(|a_1 - a_2|)$ in a look-up table, where $r(\cdot)$ is a refinement of the approximation $\max(a_1, a_2)$. Max-log approximation further simplifies (5.2) when the refinement term is left out with negligible loss in performance. With these simplifications, $L_D(b_k|\mathbf{y}) - L_A(b_k)$ can be written as

$$\begin{aligned} L_E(b_k|\mathbf{y}) = & \max_{b \in \mathcal{L}_{k,+1}} \left\{ -\frac{1}{2\sigma^2} ||\mathbf{y} - \mathbf{H}\mathbf{x}||^2 + \frac{1}{2} \mathbf{b}_{[k]}^T \mathbf{L}_{A,[k]} \right\} \\ & - \max_{b \in \mathcal{L}_{k,-1}} \left\{ \frac{1}{2\sigma^2} ||\mathbf{y} - \mathbf{H}\mathbf{x}||^2 + \frac{1}{2} \mathbf{b}_{[k]}^T \mathbf{L}_{A,[k]} \right\}. \end{aligned} \quad (5.4)$$

5.1.1 Tree Search Algorithms

The tree search algorithms can be used to solve or approximate the hard output ML solution with reduced complexity compared to the full-complexity ML detector. They are based on preprocessing and tree search algorithms and their application to the MIMO detection problem has gained renewed attention in the literature during the last few decades [7]. The search over the lattice points can be performed with a tree structure due to the QR decomposition applied on the channel matrix. The tree search algorithm then aims to find the shortest path in a search tree formed by the MIMO channel matrix and the transmitted symbols, i.e., solves the exact ML solution or suboptimal solution depending on the algorithm search strategy. The algorithms in the literature are often divided into three categories according to the search strategy: the breadth-first (BF) search, the depth-first search (DF), and the metric-first (MF) search [8–10].

A class of algorithms, generally called sphere detectors (SD) [11–15], solve the ML solution with a reduced number of considered candidate symbol vectors. They take into account only the lattice points that are inside a sphere of a given radius. The condition that the lattice point lies inside the sphere can be written as

$$||\mathbf{y} - \mathbf{H}\mathbf{x}||^2 \leq C_0. \quad (5.5)$$

After QR decomposition of the channel matrix $\mathbf{H}$ in (5.5), it can be rewritten as $||\mathbf{y}' - \mathbf{R}\mathbf{x}||^2 \leq C_0^{'}$, where $C_0^{'} = C_0 - ||(\mathbf{Q}')^{\mathrm{H}}\mathbf{y}||^2$, $\mathbf{y}' = \mathbf{Q}^{\mathrm{H}}\mathbf{y}$, $\mathbf{R} \in \mathbb{C}^{N\times N}$ is an upper triangular matrix with positive diagonal elements, $\mathbf{Q} \in \mathbb{C}^{M\times N}$ and $\mathbf{Q}' \in \mathbb{C}^{M\times(M-N)}$ are orthogonal matrices.

The squared partial Euclidean distance (PED) of $\mathbf{x}_i^N$, i.e., the square of the distance between the partial candidate symbol vector and the partial received vector, can be calculated as

$$d(\mathbf{x}_i^N) = \sum_{j=i}^{N} \left| y_j^{'} - \sum_{l=j}^{N} r_{j,l} x_l \right|^2 , \tag{5.6}$$

where $i = N \ldots, 1$, $y_j^{'}$ is the jth element of $\mathbf{y}'$, $r_{j,l}$ is the j,lth element of the matrix $\mathbf{R}$, x_l is the lth element of the candidate vector $\mathbf{x}_i^N$, and $\mathbf{x}_i^N$ denotes the last $N-i+1$ components of vector $\mathbf{x}$ [15].

Hard output sphere detectors may cause significant performance degradation when used in a system with FEC. However, there are methods proposed in the literature to modify hard output detectors to give soft reliability information of the transmitted bits as an output. A tree search algorithm can be used to obtain a list of candidates $\mathcal{L}$ and their Euclidean distances which are used to calculate the APPs L_D of the coded bits in $\mathbf{b}_p$. The size of the candidate list and the bounding of the tree search define the trade-off between complexity and the quality of the soft output information. List detector algorithms continue the tree search until a defined list is obtained. LSDs can be used to approximate the MAP detector and to provide soft outputs for the decoder [3]. The algorithms can often be derived from the sphere detector algorithms with minor modifications.

A tree search detector structure is presented in Fig. 5.2. The channel matrix $\mathbf{H}$ is first decomposed as $\mathbf{H} = \mathbf{QR}$ in the QR-decomposition block. The Euclidean distances between the received signal vector $\mathbf{y}$ and the possible transmitted symbol vectors are calculated in the tree search block. The candidate symbol list $\mathcal{L}$ from the tree search block is demapped to a binary form. The tree search algorithm can be any algorithm that produces a list of candidate symbols, for example the LSD. The LLRs are calculated from the list of Euclidean distances in the LLR block. Limiting the range of LLRs reduces the required list size [16].

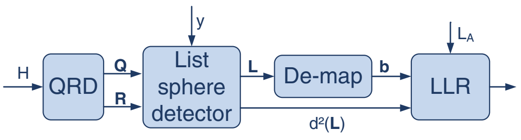

Fig. 5.2 The structure of the tree search detector

5.2 Breadth-First Detection

Breadth-first algorithms, such as the M algorithm [10] or the K-best Algorithm [17, 18] with sphere radius, extend the search in a layer-by-layer basis with multiple paths and always proceed in the depth direction of the tree. The algorithms always keep a constant number of candidate paths in each layer of the tree if no sphere radius constraint is introduced, but also require sorting of the candidate paths at each tree layer. The fixed complexity sphere decoder (FSD) [19] and the selective spanning with fast enumeration (SSFE) algorithm [20] also have a fixed complexity as they search over a fixed number of lattice points around the received signal. They both have a predefined number of nodes to be searched in the tree. Breadth-first algorithms guarantee a fixed number of visited nodes, which makes the algorithm very suitable for implementation. However, the breadth-first search strategy does not guarantee the ML solution and the search as such is inefficient in term of visited nodes especially with higher order modulation compared to the other tree search strategies.

5.2.1 K-Best Detection

The K-best algorithm [17] is a breadth-first search based algorithm, which keeps the K nodes which have the smallest accumulated Euclidean distances at each level. If the PED is larger than the squared sphere radius C_0, the corresponding node will not be expanded. The K-best algorithm without the sphere constraint can also be seen as the M-algorithm [10]. Here, $C_0 = \infty$, but a set the value for K is used instead, as is common with the K-best algorithms. The K-best LSD algorithm description is given as Algorithm 1. The main loop of the algorithm runs from $i = 1, \ldots, 2N$ in a real valued system, i.e., the real and complex parts of the signal are treated separately [15, 21].

The K-best tree search with no sphere constraint is illustrated in Fig. 5.3. A list size of two is assumed. The tree search proceeds level by level, expanding all the child nodes of each parent node. If the number of child nodes exceeds the list size,

Algorithm 1 The K-best LSD algorithm

```
Inputs: $\mathbf{Q}, \mathbf{R}, \mathbf{y}$, $C'_0$, $K$, P (modulation used, P-QAM)
Preprocessing: Calculate $\mathbf{y}'$
Algorithm:
for $i = 1, \ldots, N$
    1. Denote the partial candidate by $\mathbf{x}_{i+1}^N$.
    1.1 Determine all admissible candidate child nodes $x_i$
    (with given $C'_0$) and the corresponding PEDs $d(\mathbf{x}_i^N)$.
    1.2 Store the partial candidates and their PEDs
        to a temporary stack memory.
    2. Sort the partial candidates according
    to their PEDs
    3. Store the $K$ smallest PEDs and symbol
    vectors to the final list stack memory.
end
Give the candidates and their
EDs as outputs.
```

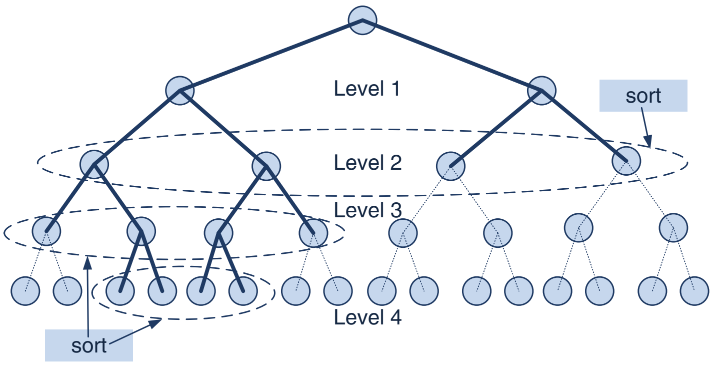

Fig. 5.3 The K-best tree search

sorting is performed to find the K nodes with the smallest PEDs. The tree search starts from the top of the tree on the first level in the figure. Both nodes are spanned, and on the second level, all the child nodes are spanned as well. Sorting is performed to find the two nodes with the smallest PEDs. The tree search continues until the fourth level is reached and the two leaf nodes with the smallest Euclidean distances are given as output. The breadth-first tree search can be modified to decrease the latency [22].

5.2.2 *Selective Spanning with Fast Enumeration*

The SSFE algorithm [20] can also be thought of as a breadth-first tree search algorithm. It can be also thought of as a fixed complexity detector. The algorithm spans each level of the tree based on the node spanning vector $\mathbf{m} = [m_1, \ldots, m_M]$. The number of spans for each node on a level is specified with the element of $\mathbf{m}$ corresponding to that level. As the spanned nodes are not discarded, the length of the final candidate list can be obtained by multiplying the elements of $\mathbf{m}$. For example, in a 2×2 antenna and 64-QAM system, the vector $\mathbf{m} = [64, 8]$ would lead to a final candidate list of 512. Here, a real valued system model is used. Such a system model simplifies the Euclidean distance calculation and the slicing operation as the closest constellation point selection can be done on a one dimensional axis.

The PED on each level i of the tree search can be calculated as

$$d_i(\mathbf{x}^i) = d_{i+1}(\mathbf{x}^{i+1}) + \left\| e_i(\mathbf{x}^i) \right\|^2, \tag{5.7}$$

where $d_{i+1}(\mathbf{x}^{i+1})$ is the PED from the previous level. The slicer unit selects a set of closest constellation points $\mathbf{x}^i$, minimizing

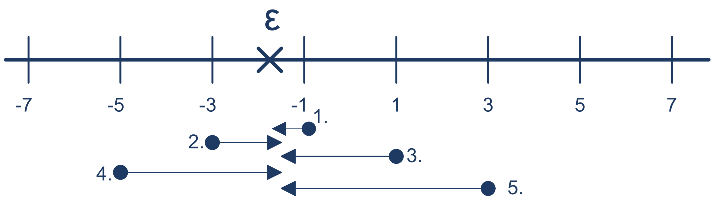

Fig. 5.4 The slicing operation in SSFE with 64-QAM

Algorithm 2 The SSFE algorithm

```
Inputs: $\mathbf{Q}, \mathbf{R}, \mathbf{y}$, $\mathbf{m}$, P (modulation used, P-QAM)
Preprocessing: Calculate $\mathbf{y}'$ and $\mathbf{h}_i = 1/\mathbf{R}_{i,i}$
Algorithm:
for $i = 1, \ldots, N$
1. Calculate $\varepsilon$ for each candidate in $\mathbf{x}^i$
2. Slice the $\mathbf{m}_i$ closest points
3. Calculate the PEDs to the sliced lattice points
end
Give the candidates and their EDs as outputs.
```

$$\left\|e_i(\mathbf{x}^i)\right\|^2 = \left\| y'_i - \sum_{j=i+1}^{M} r_{i,j}x_j - r_{i,i}x_i \right\|^2. \tag{5.8}$$

Minimizing $\left\|e_i(\mathbf{x}^i)\right\|^2$ is equivalent to the minimization of $\|e_i(\mathbf{x}^i)/r_{ii}\|^2 = \|(y'_i - \sum_{j=i+1}^{M} r_{i,j}x_j)/r_{i,i} - x_i\|^2$, where $\varepsilon = (y'_i - \sum_{j=i+1}^{M} r_{i,j}x_j)/r_{i,i}$. The closest constellation points based on ε are selected in the slicer unit.

The real valued axis for 64-QAM is shown in Fig. 5.4. The slicing order given ε is also depicted. If five constellation points are sliced, the slicer would select the constellation points in the order of $\{-1, -3, 1, -5, 3\}$. The process is similar to the Schnorr–Euchner enumeration (SEE) [23]. The SSFE algorithm could then be thought of as the M-algorithm combined with SEE. The SSFE algorithm does not require sorting, which makes it more attractive for implementation than the M-algorithm or the K-best detector. The SSFE algorithm is summarized as Algorithm 2.

5.2.2.1 Implementation Choices

The top level architecture of the K-best LSD for a 2×2 antenna system is shown in Fig. 5.5. The 4×4 antenna system LSD is based on the same architecture, but four more PED calculation blocks and sorters are added to the design. The architecture for the SSFE has a similar pipelined structure, where each level of the tree is processed separately.

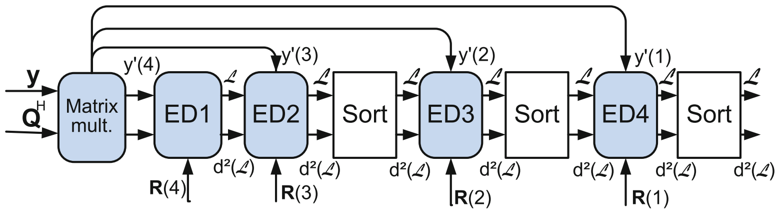

Fig. 5.5 The top level architecture of the 2×2 K-best LSD

The K-best LSD architecture is modified from [24]. A 2×2 and a 4×4 antenna system with a real signal model [25] is assumed. The received signal vector $\mathbf{y}$ is multiplied with matrix $\mathbf{Q}$ in the matrix multiplication block. Matrix $\mathbf{R}$ is multiplied with the possible transmitted symbols after the QRD is performed, i.e., when the channel realization changes. PEDs between the last symbol in vector $\mathbf{y}'$ and possible transmitted symbols are calculated in block PED1 in a 2×2 antenna system with $d(\mathbf{x}_4^2) = ||y_4^{'} - r_{4,4}^{'}||^2$. The resulting lists of symbols and PEDs are not sorted at the first stage. The distances are added to the PEDs $d(\mathbf{x}_3^2) = ||y_3^{'} - (r_{3,3}^{'} + r_{3,4}^{'})||^2$ calculated in the PED2 block. The lists are sorted and K partial symbol vectors with the smallest PEDs are kept. PED3 block calculates $d(\mathbf{x}_2^2) = ||y_2^{'} - (r_{2,2}^{'} + r_{2,3}^{'} + r_{2,4}^{'})||^2$, which are added to the previous distance and sorted. The last PED block calculates the PEDs $d(\mathbf{x}_1^2) = ||y_1^{'} - (r_{1,1}^{'} + r_{1,2}^{'} + r_{1,3}^{'} + r_{1,4}^{'})||^2$. After adding the previous distances to $d(\mathbf{x}_1^2)$, the lists are sorted and the final K symbol vectors are demapped to bit vectors and their Euclidean distance is used in the LLR calculation.

High level synthesis (HLS) was used to obtain the implementation results. Even though HLS tools have been developed for decades, only the tools developed in the last decade have gained a more widespread interest. The main reasons for this are the use of an input language, such as C, familiar to most designers, the good quality of results, and their focus on digital signal processing (DSP) [26]. HLS tools are especially interesting in the context of rapid prototyping where they can be used for architecture exploration and to produce designs with different parameters [27]. While the results may not always be as optimal as with hand-coded HDL, the tool allows experimenting with different architectures in a short amount of time. The complexity results can be close to the hand-coded ones with small designs [28]. There can be a bigger difference with large designs.

The implementation of algorithms was done by writing the architecture description with fixed-point ANSI C++ language and then applying the Catapult C Synthesis tool [29] to produce a register transfer level (RTL) description. After obtaining the RTL with the desired timing and complexity results, synthesis was performed with Synopsys Design Compiler specific tools to obtain the final complexity results. The algorithms in this section and in Sect. 5.3.4 were implemented with 0.18 µm complementary metal-oxide semiconductor (CMOS) ASIC technology for a 4×4 MIMO system with 16- and 64-QAM. The ASIC power estimation was

Table 5.1 Implementation results with 4×4 16-QAM

Receiver	Gates		Power		Detection rate
	Tree search	LLR	Tree search	LLR	
SSFE/SSFE 2 it.	135.2 k	19 k/34.6 k	488.9 mW	79 mW/158 mW	186 Mbps/163 Mbps
8-best/8-best 2 it.	97 k	17.3 k/33.1 k	341.5 mW	68.3 mW/140.5 mW	140 Mbps/126 Mbps
16-best	148.4 k	20.2 k	499.6 mW	79.2 mW	70 Mbps

Table 5.2 Implementation results with 4×4 64-QAM

Receiver	Gates		Power		Detection rate
	Tree search	LLR	Tree search	LLR	
SSFE/SSFE 2 it.	177.4 k	25.7 k/50.4 k	568.6 mW	110.5 mW/236.7 mW	269 Mbps/222 Mbps
8-best/8-best 2 it.	183.7 k	21.5 k/45.2 k	551.4 mW	87 mW/197.9 mW	210 Mbps/180 Mbps
16-best	217.2 k	24.5 k	717.3 mW	96.6 mW	105 Mbps

done with the Synopsys PrimeTime tool. Results for the enhanced tree search, other antenna configurations, adaptive systems, and other detectors can be found in [30, 31].

5.2.2.2 VLSI Implementation

The complexity and performance of two breadth-first tree search algorithms are compared. The complexity results for the SSFE and K-best detectors are presented in Table 5.1 for 16-QAM and in Table 5.2 for 64-QAM. Results for the LLR calculation are also given for a fair comparison of the two detectors. The detection rate of a receiver can be calculated as $\frac{QN}{D_{\text{rec}}}$, where Q is the number of bits per symbol, $D_{\text{rec}} = D_{\text{det}} + (D_{\text{LLR}} + D_{\text{dec}})N_{\text{iter}}$, D_{det} is the latency of the detector, D_{LLR} is the latency of LLR calculation, D_{dec} is the latency of the decoder, and N_{iter} is the number of iterations between the detector and the decoder. LLR calculation and decoding can be performed simultaneously and in a pipelined manner with detection and their latency does not have to be included in the throughput latency. In an iterative receiver, the throughput latency is determined by the minimum of D_{det} and $D_{\text{LLR}} + D_{\text{dec}}$. The receivers were designed to have a detection rate, which would be enough for the 3GPP Long Term Evolution (LTE) 20 MHz bandwidth.

The word lengths for the K-best LSD and LLR calculation are mainly 16 bits and computer simulations have been performed to confirm that there is no performance degradation [30]. The sorters are insertion sorters. The list size values of 16 and 8 are used in the implementation.

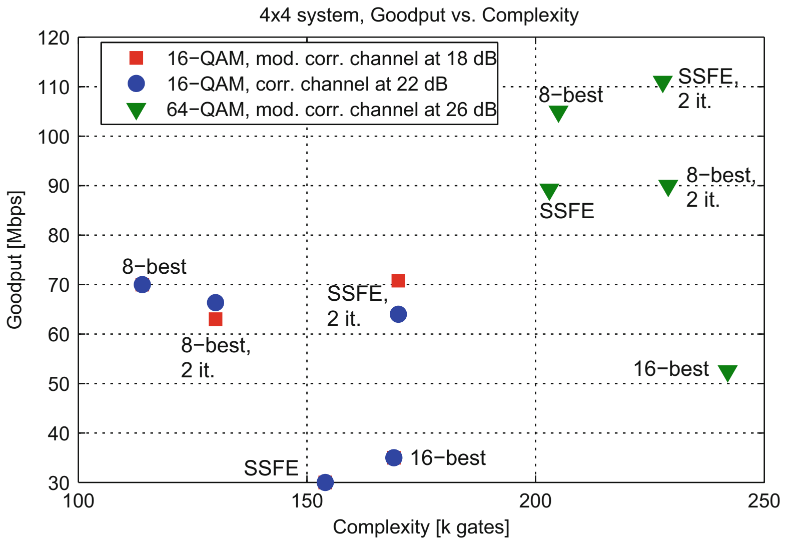

Fig. 5.6 Complexity-performance trade-off in a 4×4 antenna system

The SSFE list size is 12 and the node spanning vector is [3,2,2,1,1,1,1,1]. The clock frequency of the detectors was 280 MHz except for the 64-QAM SSFE where only a 269 MHz clock frequency was achieved. In the receiver with two global iterations, the tree search is performed only once and the complexity is the same as with one iteration. However, the LLR calculation is different in the two cases as the feedback from the decoder is used in the iterative detector. Decoding reduces the detection rate in the iterative receiver. The 8-best detector has a lower complexity and power consumption than the SSFE in the 16-QAM case, but the detection rate is also lower. The power consumption is also lower in the 64-QAM case, but the detection rate of the SSFE is higher.

The complexity-performance trade-off is illustrated in Fig. 5.6. The goodput, i.e., the minimum of the transmission throughput and hardware detection rate of information bits in a 20 MHz bandwidth with a 1/2 code rate, is compared to the hardware complexity. The figure then illustrates the communication performance of each detector compared to its complexity. The transmission throughput results were obtained with computer simulations in a realistic communication system model. The *K*-best with list size of 16 has a high complexity and low goodput. The goodput of the SSFE with two global iterations is close to that of the 8-best with one iteration with 16-QAM, but has a higher complexity. With 64-QAM, SSFE with two iterations achieves the highest goodput. Extra iterations do not bring any benefit with the *K*-best tree search as the detection rate is low. Even though the SSFE algorithm does not include sorting, the slicing operation induces extra complexity compared to the *K*-best algorithm and the difference between the two tree search algorithms remains small. The iterations between the detector and the decoder can improve the communication performance but at the same they increase the latency and complexity, resulting in a low overall gain. However, some pipelining and parallelization techniques can be used to improve the throughput [32].

5.3 Depth-First and Metric-First Detection Algorithm Implementations

In this section, we introduce some examples of soft-output depth-first and metric-first search based detection algorithms and their VLSI implementations [31]. The considered soft-output sphere detection algorithms are first presented in Sect. 5.3.1. Implementation trade-offs are then presented in Sect. 5.3.2, and the architectural choices in Sect. 5.3.3. Finally, the implementation results are presented in 5.3.4.

5.3.1 Algorithm Descriptions

The considered depth-first and metric-first search based LSD algorithms are introduced in this section.

5.3.1.1 Depth-First Algorithm

Depth-first algorithms are based on a sequential search and go through a variable number of nodes in the search tree depending on the channel realization and the signal to noise ratio (SNR). The algorithms explore the tree along the depth until the cost metric of the path is below a defined threshold called a sphere radius. They then return and pursue another unexplored path. DF algorithms are able to find the exact ML solution if the search is not bounded. The Pohst enumeration method is often considered to be the original sphere algorithm [12]. The algorithm search complexity is bounded by selecting a constant sphere radius, which limits the search in the tree to the most likely paths. More advanced adaptive sphere radius was introduced as the Viterbo–Boutros (VB) implementation [14], and the SEE [23] can be seen as even more efficient modification of the Pohst enumeration and VB implementation [11].

We consider a depth-first search based sphere detection algorithm called the SEE—LSD and it is listed as Algorithm 3. It is an extension of SEE-SD [23] to a LSD, and the algorithm continues the search until all admissible nodes have been checked and the required candidate list $\mathcal{L}$ has been obtained. The output candidate list $\mathcal{L}$ includes the most probable candidates, i.e., the candidates with the lowest ED. The sequential algorithm initially starts from the root layer and extends the partial candidate $\mathbf{s} = \mathbf{x}_N^N$ with the best admissible node determined by the SE enumeration. The search tree pruning loop in the algorithm extends the considered

Algorithm 3 $[\mathcal{L}]$ = SEE-LSD($\mathbf{y}', \mathbf{R}, N_{\text{cand}}, \Omega, N$)

```
Initialize set $\mathcal{L}$, and set $C_0 = \infty$, $m = 0$, $n_1 = 1$, $i = N$
Initialize $\mathcal{N}(\mathbf{s} = \mathbf{x}_N^N, d(\mathbf{s}) = 0)$
WHILE ($i \neq N$ and $n_1 \neq |\Omega|$) {
  IF ($n_1 = |\Omega|$) { Set $i = i + 1$, determine $n_1$ and continue with $\mathcal{N}(\mathbf{s} = \mathbf{x}_{i+2}^N, d(\mathbf{s}))$ }
  ELSE
    Determine the $n_1$th best node $x_i$ for $\mathbf{s}_c = (x_i, \mathbf{x}_{i+1}^N)$ and calculate $d(\mathbf{s}_c)$
    IF ($d(\mathbf{s}_c) < C_0$)
      IF ($\mathbf{s}_c$ is a leaf node, i.e., $i = 1$)
        1. Store $\mathcal{N}_F(\mathbf{s}_c, d(\mathbf{s}_c))$ in $\{\mathcal{L}\}^m$
        2. Set $m = m + 1$ or, if $\mathcal{L}$ is full, set $m$ according to $\{\mathcal{L}\}^m$ with max ED
           and $C_0 = \{d(\mathbf{s})\}^m$
        3. Continue with $\mathcal{N}(\mathbf{s} = \mathbf{x}_{i+1}^N, d(\mathbf{s}))$, $n_1{+}{+}$ and $i = 1$ if $n_1 + 1 \leq |\Omega|$
      ELSEIF ($i \neq 1$ or $n_1 + 1 = |\Omega|$) { Set $i = i - 1$ and $n_1 = 1$, and continue with
      $\mathcal{N}(\mathbf{s}_c, d(\mathbf{s}_c))$ }
    ELSEIF ($d(\mathbf{s}) \geq C_0$ and $i \neq N - 1$) { Set $i = i + 1$, determine $n_1$ and continue with
        $\mathcal{N}(\mathbf{s} = \mathbf{x}_{i+2}^N, d(\mathbf{s}))$ }
    ELSE {End the algorithm} }
```

partial candidate $\mathbf{s} = \mathbf{x}_{i+1}^N$ with the next best available child node in each iteration until the PED of the extended partial candidate exceeds the sphere radius C_0 or a leaf node $\mathbf{s} = \mathbf{x}_1^N$ is found. In the case of a leaf node $\mathbf{s} = \mathbf{x}_1^N$, the candidate information $\mathcal{N}(\mathbf{s}, d(\mathbf{s}))$, which includes the candidate $\mathbf{s}$ and the corresponding ED $d(\mathbf{s})$, is added to the final candidate list $\mathcal{L}$ if the ED $d(\mathbf{s})$ is lower than the current sphere radius C_0. The radius is always updated to be equal to the highest ED in the final list when the final candidate list is full and a new leaf node is found. If the extended candidate exceeds the C_0 or all the admissible nodes have been checked, the algorithm moves one layer higher and continues with the next best admissible node. The next best admissible node is determined based on the previously extended nodes.

5.3.1.2 Metric-First Algorithm

Metric-first algorithms are based on a sequential search method and the search always proceeds along a path with the best cost metric among the stored paths in the tree search [8, 33]. MF algorithms are based on Dijktra's algorithm [34, 35], which was originally used to solve the single-source shortest path problem for a graph. The application of metric-first algorithms for MIMO detection has been applied in [36–38]. MF algorithms find the exact ML solution and the search strategy is efficient in terms of visited nodes in the search tree, but requires storing and ordering of the paths studied [33].

The increasing radius (IR)—LSD is listed as Algorithm 4. The IR-LSD algorithm uses the metric-first search strategy and it is a modification of Dijkstra's algorithm [34] to a LSD algorithm. The algorithm is optimal in the sense of the number of nodes in the tree structure visited [33, 34]. The output candidate list $\mathcal{L}$ includes the most probable candidates, i.e., the algorithm always gives exactly the same output

Algorithm 4 $[\mathcal{L}] = \text{IR-LSD}(\mathbf{y}', \mathbf{R}, N_{\text{cand}}, \Omega, N)$

```
Initialize sets $\mathcal{S}$ and $\mathcal{L}$, and set $C_0 = \infty$, $m = 0$, $n_1 = 1$
Initialize $\mathcal{N}(\mathbf{s} = \mathbf{x}_N^N, d(\mathbf{s}) = 0, n_2 = 2, i = N)$
WHILE ($C_0 < d(\mathbf{s})$) {
  1. Determine the $n_1$th best node $x_i$ for $\mathbf{s}_c = (x_i, \mathbf{x}_{i+1}^N)^T$ and calculate $d(\mathbf{s}_c)$
  2. Determine the $n_2$th best node $x_{i+1}$ for father candidate
        $\mathbf{s}_f = (x_{i+1}, \mathbf{x}_{i+2}^N)^T$ and calculate $d(\mathbf{s}_f)$ if $n_2 \le |\Omega|$
  IF ($d(\mathbf{s}_c) < C_0$)
    IF ($\mathbf{s}_c$ is a leaf node, i.e., $i = 1$)
      1. Store $\mathcal{N}_F(\mathbf{s}_c, d(\mathbf{s}_c))$ in $\{\mathcal{L}\}^m$
      2. Set $m = m + 1$ or, if $\mathcal{L}$ is full, set $m$ according to $\{\mathcal{L}\}^m$ with max ED and
      $C_0 = \{d(\mathbf{s})\}^m$
      3. Continue with $\mathcal{N}(\mathbf{s} = \mathbf{x}_{i+1}^N, d(\mathbf{s}), n_1 + 1, 1)$ if $n_1 + 1 \le |\Omega|$
    ELSE { Store $\mathcal{N}_c(\mathbf{s}_c, d(\mathbf{s}_c), n_2 = 2, i - 1)$ in $\mathcal{S}$}
  IF ($\mathcal{N}_f$ calculated and $d(\mathbf{s}_f) < C_0$) { Store $\mathcal{N}_f(\mathbf{s}_f, d(\mathbf{s})_f, n_2 + 1, i)$ in $\mathcal{S}$}
  3. Continue with $\mathcal{N}$ with min PED from $\mathcal{S}$ and set $n_1 = 1$}
```

as the SEE-LSD algorithm. The algorithm always extends the partial candidate with the lowest PED in one extend loop. The algorithm operates in a sequential fashion; it initially starts from the root layer with partial candidate $\mathbf{s} = \mathbf{x}_N^N$ and determines the next best admissible node x_i at layer i with SEE. The child candidate is then defined as $\mathbf{s}_c = (x_i, \mathbf{x}_{i+1}^N)^T$. The algorithm also, if possible, extends the father candidate $\mathbf{s}_f = \mathbf{x}_{i+2}^N$ with the next best admissible node x_{i+1}. The SEE, which is used to determine the next best admissible node, requires the information of already extended nodes, and the information is defined as n_1 and n_2 for the considered candidate and its father candidate, respectively. The algorithm uses two memory sets for storing the candidates, the final candidate set $\mathcal{L}$ and the partial candidate set $\mathcal{S}$. In the algorithm search, the partial child candidate information $\mathcal{N}_{\mathcal{S}}\left(\mathbf{s}_c, d(\mathbf{s}_c), n_1\right)$ and the possible father candidate information $\mathcal{N}_{\mathcal{S}}\left(\mathbf{s}_f, d(\mathbf{s}_f), n_2\right)$ are stored to set $\mathcal{S}$ after each tree pruning loop. In the case the child candidate $\mathbf{s}_c$ is a leaf node and smaller than the current radius C_0, the candidate information $\mathcal{N}_{\mathcal{L}}\left(\mathbf{s}_c, d(\mathbf{s}_c)\right)$ is stored to the final list set $\mathcal{L}$. The sphere radius C_0 is updated when $\mathcal{L}$ is full and the candidate with the largest ED is replaced with a new leaf candidate. After storing the candidate(s), the algorithm finds the candidate information $\mathcal{N}_{\mathcal{S}}$ with the minimum PED $d(\mathbf{s})$ from set $\mathcal{S}$ and continues the algorithm if the PED is smaller than the current radius C_0. It should also be noted that $n_1 = 0$ is used in the tree pruning loop if the extended node is not a leaf node, and the n, which is read from $\mathcal{S}$, is used as n_2.

5.3.2 Implementation Trade-Offs

The algorithms presented in Sect. 5.3.1 are not as such directly feasible for hardware implementation without some modifications. In order to reserve the hardware resources for the algorithm to meet the given timing constraints, we need to determine the so-called worst case scenario and determine the algorithm complexity accordingly. The SEE-LSD and the IR-LSD, however, visit a variable number of nodes depending on the channel realization, and the implementation of these algorithms as such is not feasible for a system with a fixed latency requirement. A simple way to fix the complexity is to limit the maximum number of L_{node} nodes visited by the LSD algorithm [31]. If the sphere search is not completed within the defined L_{node}, the algorithm is stopped and the current final candidate list $\mathcal{L}$ is given as an output. Another more sophisticated alternative is to use a scheduling algorithm as, e.g., in [31, 39, 40]. The idea behind the scheduling algorithm is that, e.g., in an OFDM system, higher maximum limits L_{node} can be allocated for subcarriers with a difficult channel realization while subcarriers with easier channel realization can be allocated with lower limits.

The LSD algorithms are often assumed to apply a real equivalent system model [15, 21] especially in the implementation of the algorithms. However, complex valued signal models are also applied in the literature [3, 39]. The definition of the signal model does not affect the mathematical equivalence of the expressions, but it affects the lattice definition where the LSD algorithm search is executed. The real signal model was shown to be clearly the better choice to be applied with LSD algorithms [31], and we also consider it here.

5.3.3 Architectural Choices

Architectural design choices for the considered LSD algorithms are presented in this section.

5.3.3.1 SEE-LSD

A scalable architecture for the SEE-LSD algorithm, which consists of a tree pruning unit (TPU), a control unit (CNTR), and a memory unit, is shown in Fig. 5.7. The architecture operates in sequential fashion and prunes a single node in the search tree in each iteration. The TPU executes the tree pruning, and the CNTR determines the partial candidate for the next iteration and the possible final candidate to be stored in the memory unit. The problem of variable complexity is solved by applying an input variable L_{node}, which sets a maximum limit for the number of nodes to be pruned by the architecture as discussed in Sect. 5.3.2.

The SEE-LSD algorithm TPU microarchitecture is illustrated in Fig. 5.8. The TPU microarchitecture is divided into two sub-units that can be implemented with different levels of parallelism and pipelining. It should be noted that the SEE-LSD algorithm TPU microarchitecture has to be able to calculate the tree pruning for partial candidates in different search layers. Typically the TPU should be made as fast as possible with the proper parallelism and pipelining configuration as the latency of the unit directly affects the throughput of the SEE-LSD algorithm architecture. The first unit calculates the part of PED calculation that is independent

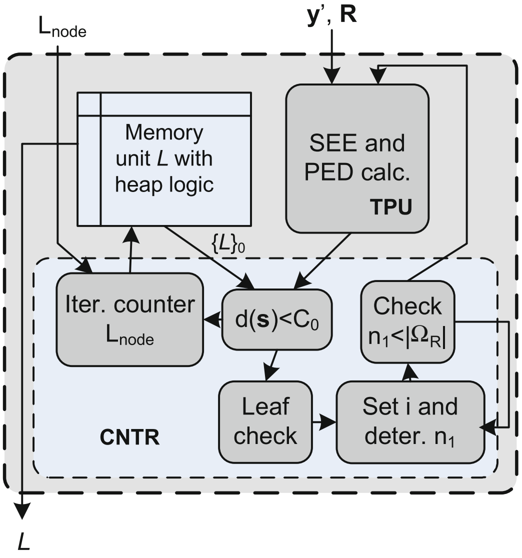

Fig. 5.7 A scalable architecture for the SEE-LSD algorithm

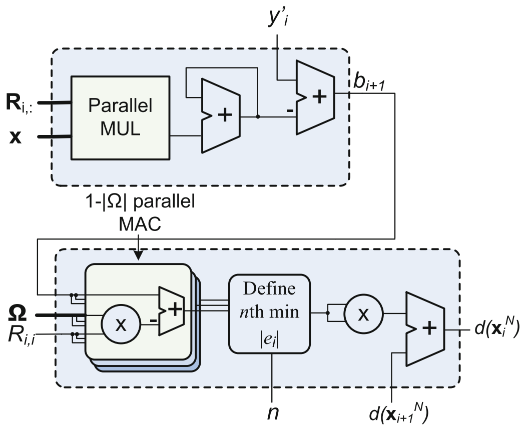

Fig. 5.8 The microarchitecture for the extension of the candidate

of the new symbol x_i in (5.6), where i is the current search layer. The second unit executes the SEE, i.e., determines the nth best node x_i, and calculates the PED of the extended partial candidate accordingly [31].

The memory unit is used to store the N_{cand} final candidates with the smallest EDs, which are found during the SEE-LSD algorithm tree search. The memory unit is designed as a binary heap [35, 41] data structure, which keeps the stored elements in order according to the selected cost metric. The memory unit $\mathcal{L}$ is implemented as max-heap, where the new element $\mathcal{N}_{\text{F}}(\mathbf{s}_c, d(\mathbf{s}_c))$ is always ordered in the heap as it is stored. The heap elements are kept in order so that the final candidate with the maximum ED is always at the top of the heap [35, 41].

The required control logic in CNTR unit for the SEE-LSD algorithm architecture is rather simple. The logic determines the next search level i and next admissible node n_1 for the next algorithm iteration based on the partial candidate, which was extended in the TPU. If the extended candidate is a leaf node and $d(\mathbf{s}) < C_0$, the final candidate is stored to the memory unit and the sphere radius C_0 is possibly updated. The CNTR unit also terminates the search after L_{node} iterations.

The SEE-LSD algorithm architecture operates in sequential fashion a total of L_{node} iterations, where the parameter L_{node} should be selected as suitable to provide the desired performance. The latency of the algorithm iterations consists of the latency of the CNTR and the latency of the TPU or the memory unit. The TPU and memory unit operations are designed to be executed in parallel, where the TPU is typically the slower unit as it includes more operations and the memory unit is executed only seldom. In order to maximize the throughput of the SEE-LSD algorithm architecture, the TPU should be implemented with proper parallelism and pipelining. The parameter L_{node} can also be lowered to increase the throughput with the cost of decreased performance. The SEE-LSD algorithm architecture is as such scalable for system configurations with different number of transmit antennas N and different constellation Ω.

5.3.3.2 IR-LSD

The IR-LSD algorithm architecture is shown in Fig. 5.9 and includes a TPU with two calculation modules, a partial candidate memory unit, a final candidate memory unit, and a control logic (CNTR) unit. In each iteration, the TPU executes the tree pruning for two partial candidates, and the CNTR determines the partial candidate for the next iteration and the possible final candidate to be stored in the

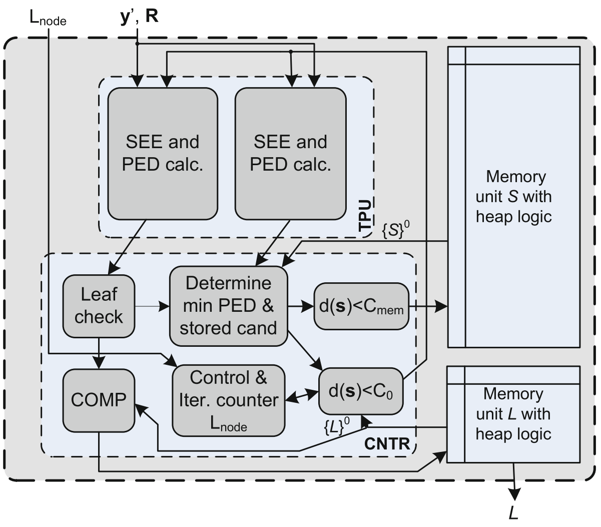
L_node
y', R
SEE and PED calc.
SEE and PED calc.
TPU
{S}^0
Memory unit S with heap logic
Leaf check
Determine min PED & stored cand
d(s)<C_mem
COMP
Control & Iter. counter L_node
d(s)<C_0
{L}^0
CNTR
Memory unit L with heap logic
L

Fig. 5.9 A scalable architecture for the IR-LSD algorithm

memory unit. The problem of variable complexity is solved by applying an input variable L_{node}, which sets a maximum limit for the number of nodes to be pruned by the architecture. The IR-LSD algorithm architecture TPU is similar to the TPU in the SEE-LSD algorithm architecture with two similar candidate extension modules, which execute the tree pruning for the new selected candidate and the corresponding father candidate in parallel. The latency of the parallel units, i.e., the parallelism and pipelining choices, should be designed to be as similar as possible for efficient design.

There are two memory units in the IR-LSD architecture: the partial candidate memory set $\mathcal{S}$ and the final memory set $\mathcal{L}_{\text{F}}$. The memory units are designed as binary heap [35, 41] data structures, which keep the stored elements in order according to the selected cost metric. The partial candidate memory set $\mathcal{S}$ is implemented as min-heap, where the elements $\mathcal{N}(\mathbf{s}, d(\mathbf{s}), n_2, i)$ are ordered so that the candidate with the minimum PED is always sorted to be at the top of the heap. The final memory set $\mathcal{L}_{\text{F}}$, which is similar to the memory unit in the SEE-LSD architecture, is implemented as max-heap, where the stored final candidates $\mathcal{N}(\mathbf{s}, d(\mathbf{s}))$ are sorted according to the ED. The size of the partial candidate memory $\mathcal{S}$ is equal to L_{node} elements. In practice, ordering of the partial memory elements might become a limiting factor in the IR-LSD algorithm implementation with a large L_{node}. A technique called as memory sphere radius C_{mem} is applied to decrease the amount of memory access [31].

The control logic unit includes an iteration counter for the IR-LSD algorithm architecture and determines the candidates to be stored in the memory units and to be used in the search in the next algorithm iteration. The candidate to be used in the TPU unit in the next iteration is determined as the candidate with minimum PED from the extended candidates $\mathcal{N}_c$ and $\mathcal{N}_f$, and the minimum candidate in partial memory $\{\mathcal{S}\}^0$. If either one of the extended candidates $\mathcal{N}_c$ or $\mathcal{N}_f$ is selected for the next algorithm iteration, $\{\mathcal{S}\}^0$ remains in the memory. Thus, unnecessary memory access is minimized as the candidates $\mathcal{N}_c$ and $\mathcal{N}_f$ are not directly stored to the memory. The extended partial candidate(s) to be stored in $\mathcal{S}$ are also conditioned with C_{mem} to minimize memory access. If the extended candidate $\mathcal{N}_c$ is a leaf node and $d(\mathbf{s}) < C_0$, the final candidate is stored to the memory unit and the sphere radius C_0 is possibly updated.

The IR-LSD algorithm architecture and its timing are designed to minimize the latency in one algorithm iteration by introducing parallel operations. The straightforward data flow mapping of algorithm would first extend the new candidates, then store them in memory units, and finally determine the new candidate for the next iteration. However, the data flow can be designed more efficiently to reduce the latency of one algorithm iteration. After the TPU extends the partial candidates in the current iteration, the control logic unit determines the new candidate for the TPU at the next iteration and the stored candidates for the memory units from the current iteration. The TPU and memory units are then executed in parallel, which decreases the latency significantly compared to the straightforward mapping of the algorithm. In order to maximize the throughput of the IR-LSD algorithm architecture, the TPU and partial memory unit should be implemented with proper parallelism and pipelining. Also the parameter L_{node} can be lowered to increase the throughput with the cost of decreased performance. The limit for the number of algorithm iterations L_{node} should be defined separately for different system configurations or according to the most complex supported configuration. A proper L_{node} value depends on the channel realization and on the search tree size, i.e., on the number of independent data streams and the constellation size $|\Omega|$.

5.3.4 VLSI Implementation Results

The SEE- and IR-LSD algorithm architectures were implemented for a 4×4 MIMO system with 16- and 64-QAM with the tools described in 5.2.2.2. The complexity results are given in area and in gate equivalents (GEs), where one GE corresponds to the area of a two-input drive-one NAND gate. The fixed-point word lengths were determined via computer simulations for a 4×4 MIMO–OFDM system and a maximum of 12 and 15 bits were found adequate for 16- and 64-QAM, respectively.

The SEE-LSD algorithm implementation is based on the architecture presented in Fig. 5.7. The SEE-LSD architecture TPU for 16-QAM was implemented with four parallel pipelined MULs in the first subunit and four parallel MULs in the latter subunit. The TPU for 64-QAM was implemented with four parallel pipelined MULs in the first subunit and eight parallel MULs in the latter subunit. Both algorithm implementations are done for output list size $N_{\text{cand}} = 15$. The synthesis results of the SEE-LSD algorithm implementation for the 0.18 µm CMOS technology are listed in Table 5.3. The IR-LSD algorithm implementation is based on the architecture presented in Fig. 5.9. The IR-LSD architecture TPU for 16-QAM was implemented, as in the SEE-LSD algorithm, with four parallel pipelined MULs in the first subunits and four parallel MULs in the latter subunits. The TPU for 64-QAM was implemented with four parallel pipelined MULs in the first subunit and eight parallel MULs in the latter subunit. Both algorithm implementations are done for output list size $N_{\text{cand}} = 15$. The memory unit $\mathcal{S}$ was implemented with dual port RAM to enhance the memory access. The maximum number of algorithm iterations is limited to 175 and 225 in the 16- and 64-QAM implementation, respectively. Thus, the memory unit size was 175×31 and 225×35 bits for the 16- and 64-QAM, respectively. The synthesis results of the IR-LSD algorithm implementation for the 0.18 µm CMOS technology are listed in Table 5.4.

Table 5.3 Synthesis results of the SEE-LSD algorithms for an SM system with $N = 4$

SEE-LSD	Area (mm^2)	kGEs	Latency	Power (mW)
16-QAM	0.13	10.6	13 cc/52 ns per it.	25
64-QAM	0.27	22.0	16 cc/64 ns per it.	38

Table 5.4 Synthesis results of the IR-LSD algorithms for an SM system with $N = 4$

IR-LSD	Area (mm^2)	kGEs	Latency	Power (mW)
16-QAM	0.31	25.4	14 cc/56 ns per it.	57
64-QAM	0.59	48.2	17 cc/68 ns per it.	90

5.3.4.1 Detection Rates

The results are applied to calculate detection rates of the algorithm implementations. The detection rate R_{det} denotes the amount of transmitted coded bits that the LSD algorithm implementation is able to detect in a certain time with a given complexity. The total detection rate R_{det} of the LSD algorithm implementation can be calculated as

$$R_{\mathrm{det}} = \frac{NQ}{\Delta_{\mathrm{tot}}} \mathrm{bits/s}, \tag{5.9}$$

where Δ_{tot} corresponds to the throughput time of the LSD algorithm implementation. The throughput time for the sequential search algorithm implementations, the SEE-LSD algorithm and the IR-LSD algorithm, is defined as $\Delta_{\mathrm{tot}} = \Delta_{\mathrm{it}} L^{\mathrm{it}}_{\mathrm{avg}}$, where Δ_{it} is the latency per algorithm iteration and $L^{\mathrm{it}}_{\mathrm{avg}}$ is the average number of executed algorithm iterations. Thus, the achievable detection rate R_{det} depends on the defined maximum limit for visited nodes L_{node}, which should be properly selected to meet the desired FER target with a given channel realization and SNR γ. Implementation results of a K-best-LSD are also added for comparison [31]. It should be noted that the IR-LSD algorithm implementation checks two nodes in one algorithm iteration and that the K-best-LSD algorithm implementation detection rate is fixed as the algorithm search goes through a fixed number of nodes with variable performance depending on the channel realization and SNR. Also it should be noted that the implementation of multiple parallel LSD algorithms can be used to achieve a higher detection rate.

The detection rates of the LSD algorithm ASIC implementations for 16- and 64-QAM in different channel environments are listed in Table 5.5. The listed SNR range is selected as the operating range of the LSD based receiver with a given configuration and channel environment. The detection rates of the SEE-LSD algorithm and IR-LSD algorithm implementations are lower at low SNR as more algorithm iterations are required to achieve adequate performance. The detection rates at high SNR correspond to cases where the minimum number of iterations provides adequate performance. The LSD algorithm implementations have different performances and complexities, and, thus, we also compare the efficiency of the implementations. The comparison is done with an algorithm work factor W_{alg}, which is calculated as a multiplication product between the used resources in terms of GEs and the implementation throughput time per subcarrier Δ_{tot}, and a smaller value reflects a more efficient implementation [42, 43]. The algorithm work

Table 5.5 Detection rates of the LSD algorithm ASIC implementations in different channel environments

$R_{\mathrm{det}}^{(\mathrm{asic})}$ (dB)	IR-LSD alg. (Mbps)	SEE-LSD alg. (Mbps)	K-best-LSD alg. (Mbps)
16-QAM, UNC, $\gamma = [13-19]$	$[4.14, 31.7]$	$[1.07, 34.2]$	62.5
16-QAM, CORR, $\gamma = [21-26]$	$[1.70, 31.7]$	$[0.35, 34.2]$	62.5
64-QAM, UNC, $\gamma = [20-25]$	$[3.71, 39.2]$	$[1.12, 41.6]$	93.8
64-QAM, CORR, $\gamma = [30-35]$	$[1.62, 39.2]$	$[0.30, 41.6]$	93.8

Table 5.6 Performance and work factor numbers of the LSD algorithm ASIC implementations in different channel environments

		IR-LSD alg.	SEE-LSD alg.	K-best-LSD alg.
16-QAM, UNC	W_{alg}	$[0.097, 0.013]$	$[0.158, 0.005]$	0.030
$\gamma = [13-19]$ dB	Perf.	Max-log—0.6 dB	Max-log—0.8 dB	Max-log—0.4 dB
16-QAM, CORR	W_{alg}	$[0.239, 0.013]$	$[0.472, 0.005]$	0.030
$\gamma = [21-26]$ dB	Perf.	Max-log—0.5 dB	Max-log—0.5 dB	Max-log—1.2 dB
64-QAM, UNC	W_{alg}	$[0.311, 0.029]$	$[0.473, 0.013]$	0.050
$\gamma = [20-25]$ dB	Perf.	Max-log—1.2 dB	Max-log—1.2 dB	Max-log—0.9 dB
64-QAM, CORR	W_{alg}	$[0.715, 0.029]$	$[1.757, 0.013]$	0.050
$\gamma = [30-35]$ dB	Perf.	Max-log—0.7 dB	Max-log—0.7 dB	Max-log—2.0 dB

factor values of the LSD algorithm ASIC implementations for 16- and 64-QAM in different channel environments are listed in Table 5.6. Also the performances of the implementations relative to the max-log-MAP detector are listed in Table 5.6.

All of the LSD algorithm implementations have advantages in certain channel environments and SNR values. The K-best-LSD algorithm implementation achieves rather good performance in the uncorrelated channel with a fixed W_{alg}, but the performance suffers in highly correlated channels. The algorithm work factor W_{alg} is best in low SNR values, but the performance cannot be tuned with the channel as in the sequential search algorithms. The SEE-LSD algorithm implementation is the most efficient in high SNR values, but is the least efficient in low SNR because of the algorithm search strategy. The IR-LSD algorithm implementation is more efficient at low SNR compared to the SEE-LSD algorithm implementation and more efficient at high SNR compared to the K-best-LSD algorithm implementation. Both sequential search algorithm implementations perform much better compared to the K-best-LSD algorithm implementation in highly correlated channels with the cost of additional complexity. The performance of the sequential search algorithms can also be tuned with the penalty of additional complexity according to the requirements.

We also calculated the required parallel LSD algorithm resources with 0.18 μm CMOS technology for a downlink receiver in a 3GPP LTE standard transmission with 20 MHz bandwidth and with $N_{\mathrm{used}} = 1{,}200$ subcarriers. We assume a 4×4

Table 5.7 The required LSD algorithm ASIC resources in the LSD detection of 3GPP LTE standard with 20 MHz BW and with $N_{\text{used}} = 1{,}200$ subcarriers

4×4 MIMO–OFDM, 16-QAM	Area (mm^2)	kGEs	Power (mW)
IR-LSD alg.	2.6–49.0	216–4,010	485–9,000
SEE-LSD alg.	1.0–100	83–8,140	198–19,200
K-best-LSD alg.	6.5	547	1,480

MIMO–OFDM system with 16-QAM, i.e., the LSD algorithm must be capable of the detection rate of 268.8 Mbps. The required 0.18 µm CMOS technology resources are scaled linearly from the LSD algorithm implementation results and are listed in Table 5.7. The required resources with IR-LSD algorithm and SEE-LSD algorithm implementations depend on the defined performance of the algorithms as discussed earlier in this section.

5.4 Trellis-Search Based MIMO Detection

In this section, we introduce a trellis-search based detection algorithm for iterative MIMO detection [44–46]. We use an unconstrained trellis structure as an alternative to the tree structure to represent the search space of a MIMO signal. We describe a trellis-based approximate Log-MAP algorithm as a replacement of the typically used Max-Log algorithm for iterative MIMO detection. We search the trellis to find a number of most likely paths for each trellis node and compute a log-sum of a number of exponential terms corresponding to a hypothesized transmitted bit value. Near-optimal performance can be achieved by choosing an appropriate number of surviving paths in the trellis-search process. The trellis-based detection algorithm is a very data-parallel algorithm because the searching operations at multiple trellis nodes can be performed simultaneously. The local search complexity at each trellis node is kept very low to reduce the overall processing time. Moreover, the trellis-based detector can support iterative MIMO detection by utilizing the a priori information from the outer channel decoder.

5.4.1 Trellis-Search Algorithm

The LLR computation requires calculations of two log-sums of $\frac{Q^{N_t}}{2}$ exponential terms. The brute-force implementation is too expensive. As a balanced trade-off between complexity and performance, we can use a reduced number (n) of exponential terms to approximate the Log-MAP algorithm as:

$$LLR(x_{k,b}) \approx \ln \sum_{n:x_{k,b}=+1} \exp\left(-\frac{1}{2\sigma^2} d(\mathbf{s})\right) - \ln \sum_{n:x_{k,b}=-1} \exp\left(-\frac{1}{2\sigma^2} d(\mathbf{s})\right), \quad (5.10)$$

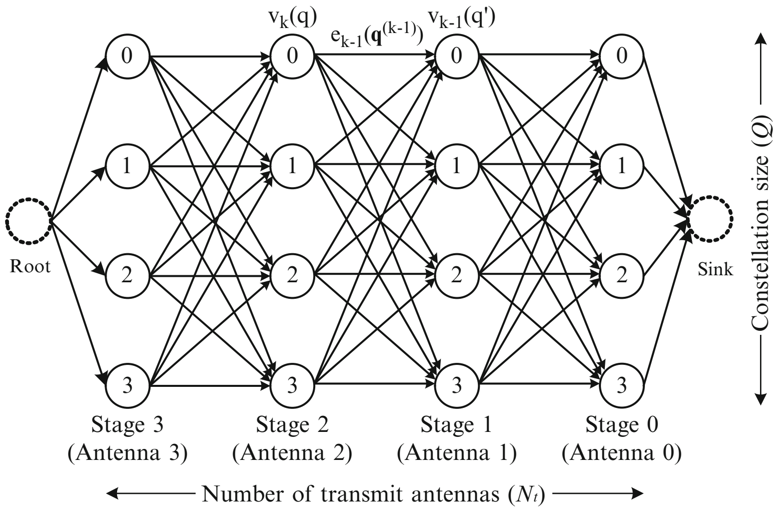

Fig. 5.10 Trellis model of a 4 × 4 4-QAM system

where the distance $d(\mathbf{s})$ is defined as:

$$d(\mathbf{s}) = \sum_{i=0}^{N_t-1} \left(|(\hat{\mathbf{y}})_i - (\mathbf{Rs})_i|^2 - \sigma^2 \sum_{j=0}^{B-1} x_{i,j} \cdot L_A(x_{i,j}) \right). \tag{5.11}$$

In the equation above, $\hat{\mathbf{y}} = \mathbf{Q}^H\mathbf{y}$, and $L_A(x_{i,j})$ is the a priori LLR for bit $x_{i,j}$. In order to implement (5.10), we must find n minimum distances $d(\mathbf{s})$ for each hypothesized transmitted data bit, i.e., $x_{k,b} = +1$ and $x_{k,b} = -1$. To realize this, we can use a trellis-search algorithm to find the n minimum distances.

5.4.2 *Trellis Model for Iterative MIMO Detection*

The search space of a MIMO signal can be represented with a compact trellis diagram. As an example, Fig. 5.10 shows the trellis diagram for a 4 × 4 4-QAM system. The trellis has N_t stages corresponding to N_t transmit antennas, and each stage contains Q different nodes corresponding to Q symbols of a complex constellation of the transmitted signal. In other words, the trellis is formed of columns representing the number of transmit antennas and rows representing values of a number of symbols with nodes at intersections. Each trellis node is physically mapped to a transmit symbol that belongs to a known modulation alphabet of the Q constellation symbols. Thus, any path through the trellis represents a possible vector ($\mathbf{s}$) of transmitted symbols. Because of the upper triangular property of the matrix $\mathbf{R}$, the stages of the trellis are labeled in descending order. The trellis is fully connected, so there are Q^{N_t} number of different paths from the root node to the sink node. The nodes in stage k are denoted as $v_k(q)$, where $q = 0, 1, \ldots, Q-1$.

To compute the distance metric in (5.11) using the trellis model, we define a weight function $e_{k-1}(\mathbf{q}^{(k-1)})$ for each edge between node $v_k(q)$ in stage k and node $v_{k-1}(q')$ in stage $k-1$ as:

$$e_{k-1}(\mathbf{q}^{(k-1)}) = \left| \hat{y}_{k-1} - \sum_{j=k-1}^{N_t-1} R_{k-1,j} \cdot s_j \right|^2 - \sigma^2 \sum_{b=0}^{B-1} x_{k-1,b} \cdot L_A(x_{k-1,b}), \quad (5.12)$$

where $\mathbf{q}^{(k-1)} = [q_{N_t-1} \, \ldots \, q_k \, q_{k-1}]^T$ is the partial symbol vector, s_j is the complex-valued QAM symbol $s_j = QAM(q_j)$, B is the number of bits per constellation point, and $L_A(x_{k-1,b})$ is the a priori information for data bit $x_{k-1,b}$ provided by the outer channel decoder. In the first iteration, $L_A(x_{k-1,b})$ is not available and is set to 0. Note that the weight function not only depends on nodes $v_k(q)$ and $v_{k-1}(q')$, but also depends on all the nodes prior to node $v_k(q)$. In other words, depending on how we traverse the trellis, the weight function will get different values. We further define a path weight as the sum of the edge weights along the path. Then the distance metric as defined in (5.11) can be considered as a path weight, which can be computed recursively by adding up the edge weights along the path from the root node to the sink node. If we define a (partial) path metric d_k as the sum of the edge weights along this (partial) path, the path weight is then computed recursively as:

$$d_{k-1}(q') = d_k(q) + e_{k-1}(\mathbf{q}^{(k-1)}), \quad (5.13)$$

where $d_k(q)$ and $d_{k-1}(q')$ are the path weights associated with nodes $v_k(q)$ and $v_{k-1}(q')$, respectively, and $e_{k-1}(\mathbf{q}^{(k-1)})$ is the edge weight between node $v_k(q)$ and node $v_{k-1}(q')$.

In the trellis diagram, each trellis node $v_k(q)$ maps to a complex-valued symbol s_k such that each path from the root node to the sink node maps to a symbol vector $\mathbf{s}$. With the trellis model, we transform the MIMO detection problem into a per-node shortest paths problem, which is defined as follows. For each node $v_k(q)$ in the trellis diagram, we find a list of L most likely paths from the root node to the sink node over the node $v_k(q)$. The L most likely paths refer to the paths with the L shortest distances or the lowest L path weights. For each node, we only keep the L most likely paths and will discard all the other paths to reduce the complexity. The detection is performed layer by layer. In the trellis model, a layer corresponds to a stage in the trellis. In each stage k of the trellis, there are Q nodes, where each node corresponds to a constellation point. For each node $v_k(q)$ in stage k, $q = 0, 1, \ldots, Q-1$, we must find L shortest paths through the trellis, which are denoted as $\lambda_k^{(l)}(q)$, $l = 0, 1, \ldots, L-1$. Then, altogether QL candidates in each stage k of the N_t stages of the trellis are used to compute the LLRs for data bits transmitted by antenna k as follows:

$$LLR(x_{k,b}) \approx \ln \sum_{(q,l):x_{k,b}=+1} \exp\Big(-\frac{1}{2\sigma^2}\lambda_k^{(l)}(q)\Big) - \ln \sum_{(q,l):x_{k,b}=-1} \exp\Big(-\frac{1}{2\sigma^2}\lambda_k^{(l)}(q)\Big). \quad (5.14)$$

With the trellis model, the detection problem now becomes a trellis-search problem. To detect a layer k, we need to search for L shortest paths for each node q in each stage k of the trellis diagram. The maximum theoretical value of the number L is Q^k, where $k = 0, 1, \ldots, N-1$ for the first stage, second stage, and etc., of the trellis. Practically, however, the number L should be kept small to reduce the complexity. The number L determines the detection performance: a larger L leads to better error performance. We will show later that even with a small L (such as $L = 2$ for $Q = 16$), the trellis-based detector can achieve good detection performance. To implement this algorithm, an exhaustive trellis-search approach would be very expensive. In order to reduce the search complexity, we use a greedy trellis-search algorithm that approximately finds the L shortest paths for each node in the trellis. In this search process, the trellis is first pruned by removing the unlikely paths. We refer to this pruning process as the "path reduction" process. In the path reduction process, the trellis is scanned from left to right, where each node retains the most likely L incoming paths using the local information it has so far. After the trellis is pruned, a second process, called the "path extension" process, is applied to extend the uncompleted paths so that each node will have L full paths through the trellis.

5.4.2.1 Path Reduction

Figure 5.11 illustrates a flow graph demonstrating a path reduction process. The path reduction process is configured to prune paths for each trellis node to a smaller number of surviving paths. The stages (columns) of the trellis are labeled in descending order, starting from stage $N_t - 1$ and ending with stage 0. Note that Fig. 5.11 illustrates only three successive stages, $k+1$, k, and $k-1$ among the N_t stages. As an example, we use a $Q = 4$, $L = 2$ case to explain the algorithm. In Fig. 5.11, each node receives $QL = 4 \times 2 = 8$ incoming paths from nodes in the previous stage of the trellis and, then, the $L = 2$ paths (the ones with the least cumulative path weights) are selected from the QL candidates. Next, the L survivors are expanded to the right so that each node will have the best QL outgoing paths forwarded to the next stage of the trellis. This process repeats until the end of the trellis. The path reduction process can effectively prune the trellis by keeping only L best incoming paths at each trellis node. As a result, each trellis node in the last stage of the trellis has L shortest paths through the trellis. However, other than the trellis nodes in the last stage, the path reduction process cannot guarantee that every trellis node will have L shortest paths through the trellis. These paths will be added as path extensions as described next.

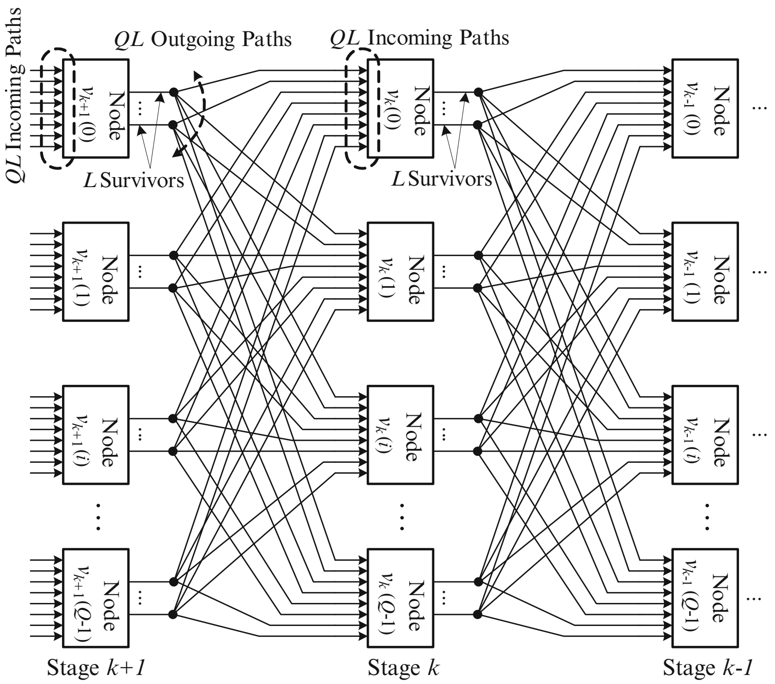

Fig. 5.11 Path reduction step

5.4.2.2 Path Extension

An objective of the trellis-based detection algorithm is to find L shortest paths for every node in the trellis. To achieve this goal, a path extension process is used after the path reduction process to fill in the missing paths for each trellis node. The goal is to extend the uncompleted paths so that each node will have L shortest paths through the trellis. The path extension is performed stage by stage (no path extension is required for the last stage), and node by node. Figure 5.12 is a flow graph demonstrating the path extension process. The path extension process is being demonstrated with respect to a node $v_k(i)$ in a stage k (i.e., the highlighted node in the figure). Note that all of the nodes in the same stage can be extended in parallel and independently.

As shown in Fig. 5.12, for a trellis node $v_k(i)$ (i.e., for the constellation point i in stage k), the path extension process first retrieves the $QL = 8$ outgoing path metrics computed in the path reduction step (at stage k), and then an extension process in stage $k-1$ is used to select the best $L = 2$ outgoing paths from $QL = 8$ candidates. Next, each of the $L = 2$ surviving paths is extended for the next stage of the trellis (stage $k-2$). Among the QL extended paths, only the best $L = 2$ paths are retained. This process repeats until the trellis has been completely traversed. As a result, the L shortest paths are obtained for node $v_k(i)$. Figure 5.12 shows a path extension

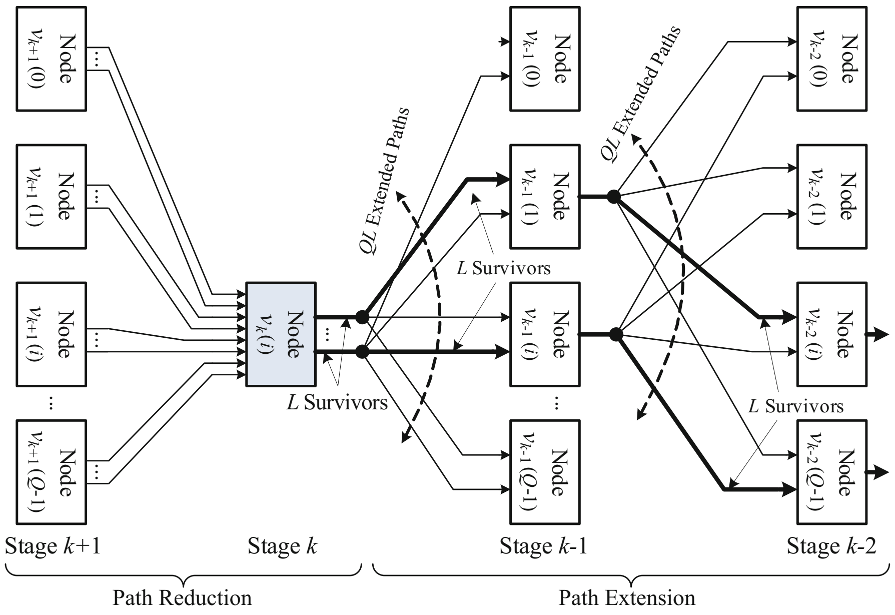

Fig. 5.12 Path extension step

process for one trellis node. The path reduction process is first performed until stage k of the trellis and next the path extension procedure is performed until the end of the trellis (stage 0). Note that the path extension process is to find the L best *outgoing* paths extending from a particular node.

5.4.2.3 LLR Computation

The most important feature of the trellis-based detection algorithm is that it will always guarantee that the bit LLR can be generated for every transmitted bit. For example, after the path reduction and the path extension processes are employed, every node $v_k(q)$ has successfully found L shortest paths or L minimum distances denoted as $\lambda_k^{(l)}(q)$, $l = 0, 1, \ldots, L-1$. We separate the LLR computation into two steps. A symbol reliability metric $\Gamma_k(q)$ is first computed for each node $v_k(q)$ as follows:

$$\Gamma_k(q) = \ln \sum_{l=0}^{L-1} \exp\Big(-\frac{1}{2\sigma^2}\lambda_k^{(l)}(q)\Big) = \overset{*}{\max_l}\Big(-\frac{1}{2\sigma^2}\lambda_k^{(l)}(q)\Big), \tag{5.15}$$

where the two-input $\max^*(\cdot)$ is defined as:

$$\overset{*}{\max}(a,b) \equiv \ln \sum (\exp(a) + \exp(b)) = \max(a,b) + \ln(1 + \exp(-|a-b|)). \tag{5.16}$$

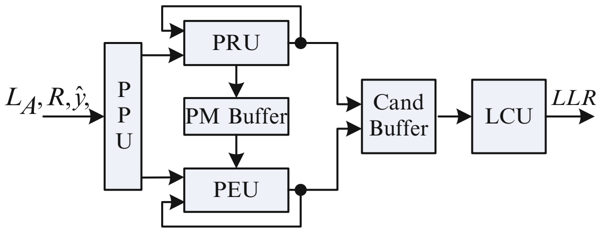

Fig. 5.13 VLSI architecture of the trellis-based iterative MIMO detector

Moreover, the n-input $\max^*(\cdot)$ for $n = 4, 8, 16$, etc., can be recursively computed based on the Jacobian algorithm. Then, the bit LLR is computed based on the symbol reliabilities $\Gamma_k(q)$:

$$\begin{aligned} LLR(x_{k,b}) &= \ln \sum_{q:x_{k,b}=+1} \exp(\Gamma_k(q)) - \ln \sum_{q:x_{k,b}=-1} \exp(\Gamma_k(q)) \\ &= \overset{*}{\max_{q:x_{k,b}=+1}}(\Gamma_k(q)) - \overset{*}{\max_{q:x_{k,b}=-1}}(\Gamma_k(q)). \end{aligned} \tag{5.17}$$

5.4.3 VLSI Architecture

Now we describe a high-speed VLSI architecture for the trellis-search based SfISfO MIMO detector. As a case study, we introduce a detector architecture with the surviving path number $L = 2$ for the 4×4 16-QAM system. Figure 5.13 shows the top level block diagram for the trellis-search MIMO detector. The detector consists of six main functional blocks: the path reduction unit (PRU), the path extension unit (PEU), the LLR calculation unit (LCU), the pre-processing unit (PPU), the path metric buffer (PM Buffer), and the candidate buffer (Cand Buffer). The PPU is used to pre-compute the initial path metrics and some constellation-dependent constant values that will be used by the PRU and the PEU. The PRU and the PEU are employed to implement the path reduction algorithm (cf. Fig. 5.11) and the path extension algorithm (cf. Fig. 5.12), respectively. The shortest path metrics found by the PRU and the PEU are stored in the Cand Buffer, which will then be used by the LCU to generate the LLR for each data bit based on (5.17). These blocks will be discussed in more detail in the following subsections.

Figure 5.14 shows the block diagram of PRU, which implements the path reduction algorithm. The PRU employs $Q = 16$ path calculation units (PCUs) and 16×2 minimum finder units (MFUs) to simultaneously process all the Q nodes in a trellis stage. This is a recursive architecture by reusing the hardware for processing nodes in different trellis stages. In Fig. 5.14, PCU i is used to compute the QL extended path metrics from node $v_k(i)$ to all the nodes in the next stage $k-1$. The extended path metrics are denoted as $\beta_{k-1}^{(l)}(i, j)$, where l is the surviving path

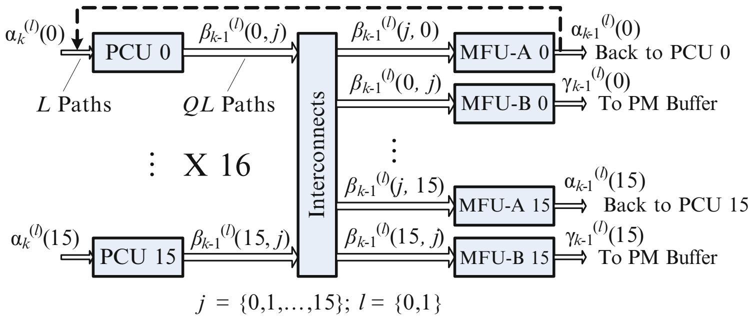

Fig. 5.14 Path reduction unit (PRU)

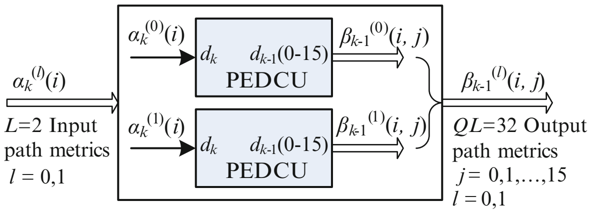

Fig. 5.15 Path calculation unit (PCU)

index ($l = 0, 1, \ldots, L-1$), i is the current node index, and j is the node index in the next stage ($j = 0, 1, \ldots, Q-1$). Next, the extended path metrics are gathered and sent to MFUs. In Fig. 5.14, MFU-A i is used to select the best L incoming paths to node $v_{k-1}(i)$, where the surviving path metrics are denoted as $\alpha_{k-1}^{(l)}(i)$, where $l = 0, 1, \ldots, L-1$. Then, these surviving paths are fed back to PCU i so that it can continue the processing for the next trellis stage. This operation is repeated until the trellis is completely traversed. MFU-B i is used to select the best L outgoing paths, denoted as $\gamma_{k-1}^{(l)}(i)$, from node $v_k(i)$ to any nodes in stage $k-1$. These best outgoing paths selected by MFU-B i will be stored into the path metric buffer (PM Buffer), which will be used later in the path extension process. Each PCU in Fig. 5.14 is used to compute $QL = 32$ path metrics in parallel. Figure 5.15 shows the block diagram of PCU which employs $L = 2$ PED calculation units (PEDCUs). For a given input path metric, or PED, d_k, one PEDCU needs to compute $Q = 16$ extended PEDs in parallel, denoted as $d_{k-1}(q)$, $q = 0, 1, \ldots, Q-1$. Figure 5.16 shows the hardware architecture for the PEDCU, which computes $Q = 16$ PEDs in parallel. Note that variables $R_{k-1,k-1}^2 |s_{k-1}(q)|^2$ and $\sigma^2 L_A(x_{k-1,b})$ are pre-computed in the PPU.

The MFU is used to select the best $L = 2$ path metrics from $QL = 32$ candidates. This type of $(32, 2)$ sorting can be done quickly by using a comparison tree. Note that the sorting cost of the trellis-based detector is much lower compared with the regular K-best detector which typically requires a larger (QK, K) sorting operation.

The PEU implements the path extension algorithm. As previously discussed, a path extension process is employed after the path reduction process to fill in the

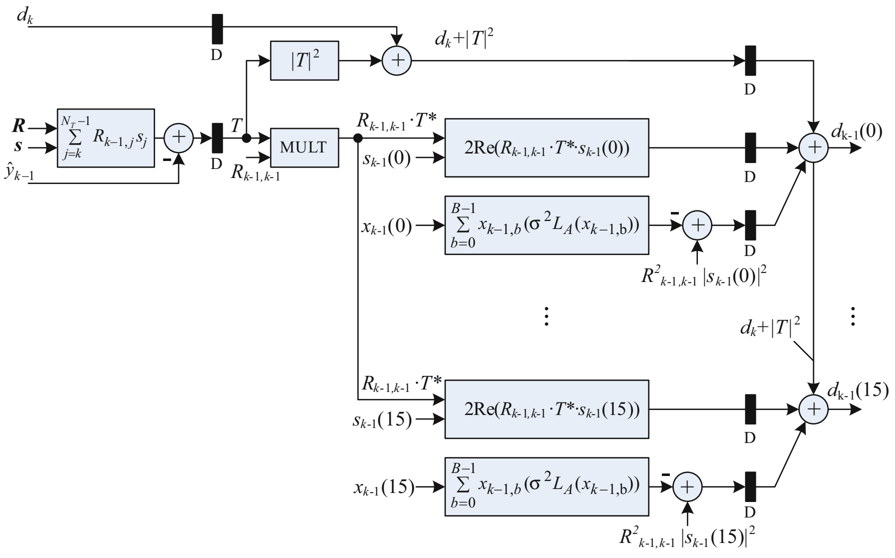
d_k
D
$|T|^2$
+
$d_k+|T|^2$
D
R
s
$\hat{y}_{k-1}$
$\sum_{j=k}^{N_T-1} R_{k-1,j}s_j$
+
-
D
T
MULT
$R_{k-1,k-1}$
$R_{k-1,k-1}\cdot T^*$
$s_{k-1}(0)$
$2\mathrm{Re}(R_{k-1,k-1}\cdot T^*\cdot s_{k-1}(0))$
D
+
$d_{k-1}(0)$
$x_{k-1}(0)$
$\sum_{b=0}^{B-1} x_{k-1,b}(\sigma^2 L_A(x_{k-1,b}))$
-
+
D
$R^2_{k-1,k-1}|s_{k-1}(0)|^2$
$d_k+|T|^2$
$R_{k-1,k-1}\cdot T^*$
$s_{k-1}(15)$
$2\mathrm{Re}(R_{k-1,k-1}\cdot T^*\cdot s_{k-1}(15))$
D
+
$d_{k-1}(15)$
$x_{k-1}(15)$
$\sum_{b=0}^{B-1} x_{k-1,b}(\sigma^2 L_A(x_{k-1,b}))$
-
+
D
$R^2_{k-1,k-1}|s_{k-1}(15)|^2$

Fig. 5.16 Partial Euclidean distance calculation unit (PEDCU)

missing paths for each node so that every node will have L shortest paths through the trellis. The PEU has a very similar architecture to the PRU. The PEU employs $Q = 16$ PCUs and $Q = 16$ MFUs so that it can simultaneously extend Q nodes in a certain trellis stage. The PEU has a recursive architecture. In each iteration, PCU i calculates the QL extended path candidates based on the L input path metrics, and then, the MFU i selects the best L paths from these QL extended path candidates. The initial L input path metrics are retrieved from the PM Buffer, and, then, the PEU performs the path extension operation recursively.

5.4.4 VLSI Implementation Results

As a case study, we have developed a trellis-search based iterative MIMO detector ASIC module for a 4×4 16-QAM MIMO system. The fixed-point design parameters are summarized as follows. Each element in the $\mathbf{R}$ matrix is scaled by $\frac{1}{\sqrt{10N_t}} = \frac{1}{\sqrt{40}}$, and this scaled R is represented with 11 bits signed data S2.9 (2 integer bits with 9 fractional bits). The received signal y is represented with 11 bits signed data S5.6. The path metrics (PMs) are rounded to 13 bits between computational blocks. The LLR values are represented with 7 bit signed data S5.2. With this configuration, the fixed-point simulation result shows about $0.1 \sim 0.2$ dB performance degradation compared to a floating-point detector. The trellis-search detector has a pipelined architecture, where the pipeline stages for the PRU and PEU are $T = 4$. To maximize the throughput, we can feed four back-to-back MIMO

Table 5.8 VLSI implementation results for 4×4 trellis-search MIMO detector

Clock frequency	Throughput (1 iter.)	Core area	Gate count	Technology
320 MHz	1.7 Gbps	1.58 mm^2	1097K	65 nm

symbols in four consecutive cycles, e.g., at $t, t+1, t+2, t+3$ into the pipeline to fully utilize the hardware. The processing times for the path reduction process and the path extension process are both $3T = 12$ cycles, i.e., the iteration bound is 12 cycles. Thus, we can feed another four back-to-back MIMO symbols into the pipeline at $t+12, t+13, t+14, t+15$, and so forth. Furthermore, we can overlap the path reduction process with the path extension process to hide the processing delay. As a result, the maximum throughput of the detector is $\frac{4\times 16\times fclk}{12} = \frac{16}{3} fclk$.

We have described the trellis-search detector with Verilog HDL and we have synthesized the design for a 1.08 V TSMC 65 nm CMOS technology using Synopsys Design Compiler. With a 320 MHz clock frequency, the detector can achieve a maximum throughput of 1.7 Gbps. Table 5.8 summaries the VLSI implementation results. From Table 5.8, one can observe that the trellis-search detector can achieve a very high data throughput (1.7 Gbps) while still maintaining a low area requirement (1.58 mm^2).

References

1. Hassibi B, Vikalo H (2005) On the sphere-decoding algorithm I. Expected complexity. IEEE Trans Signal Process 53(8):2806–2818
2. Paulraj A, Nabar RD, Gore D (2003) Introduction to space-time wireless communications. Cambridge University Press, Cambridge
3. Hochwald B, ten Brink S (2003) Achieving near-capacity on a multiple-antenna channel. IEEE Trans Commun 51(3):389–399
4. Hagenauer J (1997) The turbo principle: tutorial introduction and state of the art. In: Proceedings of the international symposium on turbo codes, Brest, France
5. Hagenauer J, Offer E, Papke L (1996) Iterative decoding of binary block and convolutional codes. IEEE Trans Inf Theory 42(2):429–445
6. Robertson P, Villebrun E, Hoeher P (1995) A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain. In: Proceedings of the IEEE international conference on communications, pp 1009–1013
7. Damen MO, Chkeif A, Belfiore J-C (2000) Lattice code decoder for space-time codes. IEEE Commun Lett 4(5):161–163
8. Murugan A, Gamal HE, Damen M, Caire G (2006) A unified framework for tree search decoding: rediscovering the sequential decoder. IEEE Trans Inf Theory 52(3):933–953
9. Anderson T (1984) An introduction to multivariate statistical analysis, 2nd edn. Wiley, New York
10. Anderson J, Mohan S (1984) Sequential coding algorithms: a survey and cost analysis. IEEE Trans Commun 32(2):169–176
11. Agrell E, Eriksson T, Vardy A, Zeger K (2002) Closest point search in lattices. IEEE Trans Inf Theory 48(8):2201–2214

12. Fincke U, Pohst M (1985) Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Math Comput 44(5):463–471
13. Pohst M (1981) On the computation of lattice vectors of minimal length, successive minima and reduced basis with applications. ACM SIGSAM Bull 15:37–44
14. Viterbo E, Boutros J (1999) A universal lattice code decoder for fading channels. IEEE Trans Inf Theory 45(5):1639–1642
15. Damen MO, Gamal HE, Caire G (2003) On maximum–likelihood detection and the search for the closest lattice point. IEEE Trans Inf Theory 49(10):2389–2402
16. Myllylä M, Antikainen J, Juntti M, Cavallaro J (2007) The effect of LLR clipping to the complexity of list sphere detector algorithms. In: Proceedings of the annual Asilomar conference on signals, systems, and computers, Pacific Grove, 4–7 November 2007, pp 1559–1563
17. Wong K, Tsui C, Cheng RK, Mow W (2002) A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels. In: Proceedings of the IEEE international symposium on circuits and systems, vol 3, Scottsdale, AZ, 26–29 May 2002, pp 273–276
18. Guo Z, Nilsson P (2006) Algorithm and implementation of the K-best sphere decoding for MIMO detection. IEEE J Sel Areas Commun 24(3):491–503
19. Barbero L, Thompson J (2008) Extending a fixed-complexity sphere decoder to obtain likelihood information for turbo-MIMO systems. IEEE Trans Vehicular Technol 57(5):2804–2814
20. Li M, Bougart B, Lopez E, Bourdoux A (2008) Selective spanning with fast enumeration: a near maximum-likelihood MIMO detector designed for parallel programmable baseband architectures. In: Proceedings of the IEEE international conference on communication, Beijing, China, 19–23 May 2008, pp 737–741
21. Guo Z, Nilsson P (2006) Algorithm and implementation of the K-best sphere decoding for MIMO detection. IEEE J Sel Areas Commun 24(3):491–503
22. Ketonen J, Myllylä M, Juntti M, Cavallaro JR (2008) ASIC implementation comparison of SIC and LSD receiver for MIMO-OFDM. In: Proceedings of the annual Asilomar conference on signals, systems, and computers, Pacific Grove, 25–29 October 2008, pp 1881–1885
23. Schnorr CP, Euchner M (1994) Lattice basis reduction: improved practical algorithms and solving subset sum problems. Math Program 66(2):181–191
24. Ketonen J, Juntti M, Cavallaro J (2010) Performance-complexity comparison of receivers for a LTE MIMO-OFDM system. IEEE Trans Signal Process 58(6):3360–3372
25. Myllylä M, Juntti M, Cavallaro JR (2007) Implementation aspects of list sphere detector algorithms. In: Proceedings of the IEEE global telecommunication conference, Washington, D.C., 26–30 November 2007, pp 3915–3920
26. Martin G, Smith G (2009) High-level synthesis: past, present, and future. IEEE Des Test Comput 26(4):18–25
27. Casseau E, Gal L, Bomel P, Jego C, Huet S, Martin E (2005) C-based rapid prototyping for digital signal processing. In: Proceedings of the European signal processing conference, Antalya, Turkey, 4–8 September 2005
28. Cong J, Liu B, Neuendorffer S, Noguera J, Vissers K, Zhang Z (2011) High-level synthesis for FPGAs: from prototyping to deployment. IEEE Trans Comput Aided Des Integr Circuits Syst 30(4):473–491
29. Calypto (2014) Catapult C synthesis overview. Technical report. http://calypto.com/en/products/catapult/overview
30. Ketonen J (2012) Equalization and channel estimation algorithms and implementations for cellular MIMO-OFDM downlink. Ph.D. dissertation, Acta Univ. Oul., C Technica 423, University of Oulu, Oulu
31. Myllylä M (2011) Detection algorithms and architectures for wireless spatial multiplexing in MIMO–OFDM systems. Ph.D. dissertation, Acta Univ. Oul., C Technica 380, University of Oulu, Oulu

32. Preyss N, Burg A, Studer C (2012) Layered detection and decoding in MIMO wireless systems. In: Conference on design and architectures for signal and image processing (DASIP), Karlsruhe, Germany, 23–25 October 2012, pp 1–8
33. Mohan S, Anderson JB (1984) Computationally optimal metric-first code tree search algorithms. IEEE Trans Commun 32(6):710–717
34. Dijkstra EW (1959) A note on two problems in connexion with graphs. In: Numerische Mathematik, vol 1. Mathematisch Centrum, Amsterdam, pp 269–271
35. Knuth D (1997) The art of computer programming. Volume 3: sorting and searching, 3rd edn. Addison-Wesley, Reading
36. Baro S, Hagenauer J, Witzke M (2003) Iterative detection of MIMO transmission using a list-sequential (LISS) detector. In: Proceedings of the IEEE international conference on communications, vol 4, pp 2653–2657
37. Xu W, Wang Y, Zhou Z, Wang J (2004) A computationally efficient exact ML sphere decoder. In: Proceedings of the IEEE global telecommunication conference, vol 4, 29 November – 3 December 2004, pp 2594–2598
38. Hagenauer J, Kuhn C (2007) The list-sequential (LISS) algorithm and its application. IEEE Trans Commun 55(5):918–928
39. Studer C, Burg A, Bolcskei H (2008) Soft-output sphere decoding: algorithms and VLSI implementation. IEEE J Sel Areas Commun 26(2):290–300
40. Burg A, Borgmanr M, Wenk M, Studer C, Bolcskei H (2006) Advanced receiver algorithms for MIMO wireless communications. In: Proceedings of the design, automation and test in Europe (DATE'06), vol 1, March 2006, 6 pp
41. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. MIT Press, Cambridge
42. Ullman J (1994) Computational aspects of VLSI. Computer Science Press, Rockville
43. Bajwa RS, Owens R, Irwin M (1994) Area time trade-offs in micro-grain VLSI array architectures. IEEE Trans Comput 43(10):1121–1128
44. Sun Y, Cavallaro JR (2009) High throughput VLSI architecture for soft-output MIMO detection based on a Greedy graph algorithm. In: ACM great lakes symposium on VLSI design, May 2009, pp 445–450
45. Sun Y, Cavallaro JR (2012) High-throughput soft-output MIMO detector based on path-preserving trellis-search algorithm. IEEE Trans Very Large Scale Integr (VLSI) Syst 20(7):1235–1247
46. Sun Y, Cavallaro JR (2012) Trellis-search based soft-input soft-output MIMO detector: algorithm and VLSI architecture. IEEE Trans Signal Process 60(5):2617–2627